Contents

Modify([Boolean]) Method

ModifyAll(Any, Any [, Boolean]) Method

Next([Integer]) Method

ReadConsistency() Method

ReadPermission() Method

RecordId() Method

RecordLevelLocking() Method

Relation(Any) Method

Rename(Any [, Any,...]) Method

Reset() Method

SecurityFiltering([SecurityFilter]) Method

SetAscending(Any, Boolean) Method

SetAutoCalcFields([Any,...]) Method

SetCurrentKey(Any [, Any,...]) Method

SetFilter(Any, String [, Any,...]) Method

SetLoadFields([Any,...]) Method

SetPermissionFilter() Method

SetPosition(String) Method

SetRange(Any [, Any] [, Any]) Method

SetRecFilter() Method

SetView(String) Method

TableCaption() Method

TableName() Method

TestField(Any) Method

TestField(Any, Boolean) Method

TestField(Any, Integer) Method

TestField(Any, BigInteger) Method

TestField(Any, Decimal) Method

TestField(Any, Guid) Method

TestField(Any, Text) Method

TestField(Any, Label) Method

TestField(Any, TextConst) Method

ConvertStr(String, String, String) Method

CopyStr(String, Integer [, Integer]) Method

DelChr(String [, String] [, String]) Method

DelStr(String, Integer [, Integer]) Method

IncStr(String) Method

InsStr(String, String, Integer) Method

LowerCase(String) Method

MaxStrLen(String) Method

MaxStrLen(Variant) Method

PadStr(String, Integer [, String]) Method

SelectStr(Integer, String) Method

StrCheckSum(String [, String] [, Integer]) Method

StrLen(String) Method

StrPos(String, String) Method

StrSubstNo(String [, Any,...]) Method

UpperCase(String) Method

Contains(Text) Method

EndsWith(Text) Method

IndexOf(Text [, Integer]) Method

IndexOfAny(Text [, Integer]) Method

IndexOfAny(List of [Char] [, Integer]) Method

LastIndexOf(Text [, Integer]) Method

PadLeft(Integer [, Char]) Method

PadRight(Integer [, Char]) Method

Remove(Integer [, Integer]) Method

Replace(Text, Text) Method

Split([Text,...]) Method

Split(List of [Text]) Method

Split(List of [Char]) Method

StartsWith(Text) Method

Substring(Integer [, Integer]) Method

ToLower() Method

IsDate() Method

IsDateFormula() Method

IsDateTime() Method

IsDecimal() Method

IsDefaultLayout() Method

IsDotNet() Method

IsDuration() Method

IsExecutionMode() Method

IsFieldRef() Method

IsFile() Method

IsFilterPageBuilder() Method

IsGuid() Method

IsInStream() Method

IsInteger() Method

IsJsonArray() Method

IsJsonObject() Method

IsJsonToken() Method

IsJsonValue() Method

IsNotification() Method

IsObjectType() Method

IsOption() Method

IsOutStream() Method

IsRecord() Method

IsRecordId() Method

IsRecordRef() Method

IsReportFormat() Method

IsSecurityFiltering() Method

IsTableConnectionType() Method

IsTestPermissions() Method

IsText() Method

IsTextBuilder() Method

IsTextConstant() Method

Development and Administration for Dynamics 365 Business Central

2/17/2021 • 5 minutes to read • Edit Online

Dynamics 365 Business Central is a complete enterprise resource planning (ERP) software solution for mid-sized organizations that is fast to implement, easy to configure, and simple to use. Right from the start, simplicity has guided — and continues to guide — innovations in product design, development, implementation, and usability. Users can customize and administrate their Business Central experience in the product. Other tasks for administration and development require a more specialized profile, and in this *Development and Administration* section, you can find more technical information, including information about developing for Dynamics 365 Business Central using the AL Language extension and Visual Studio Code.

Get started with development and administration documentation

The content in this section of the Docs.microsoft.com site is intended for people who are Business Central resellers, administrators, or developers, and people who want to get started with Business Central development or administration. If you are a functional consultant or user of Business Central, check out the Business Functionality Help instead.

Resources for an administrator

As a system administrator, IT professional, or superuser, you have tools available for you to configure, update, and maintain a Business Central solution. The tools and processes are somewhat different depending on whether Business Central is deployed online or on-premises. It also depends on whether you are an internal administrator, or you work for a partner.

The following table outlines recommended content for you to get started:

LINK	DESCRIPTION
Deployment of Dynamics 365 Business Central	Describes the difference between making Business Central available online and on-premises.
Administration of Business Central Online as an internal administrator	Learn about resources that are available to you as the system administrator, IT professional, or superuser of a Business Central customer.
Administration of Business Central Online as a partner	Learn about resources that are available to you as a Business Central reselling partner.
The Business Central Administration Center	Learn about the administration center and what you can do there.
Technical Support for Dynamics 365 Business Central	Learn about the tools that are available to you to help you troubleshoot your customers' Business Central.
Managing Technical Support	Learn about how to handle support requests.

LINK	DESCRIPTION
Administration of Business Central On-Premises	Learn about resources that are available to you as the system administrator, IT professional, or superuser of Business Central on-premises.
Configuring the Help Experience	Learn about your options for deploying Help for Business Central.

Resources for a developer

You can extend Business Central with add-on apps, vertical or horizontal solutions, and with integration to other products and services.

The following table outlines recommended content for you to get started:

LINK	DESCRIPTION
Developer learning paths on Microsoft Learn	Provides links to role-specific training.
Development in AL	Get an understanding of the basics and terms you will encounter while working in Visual Studio Code with the AL extension.
Getting Started with AL	Learn how to set up a development environment.
The SMB Opportunity for App Publishers	Learn about the business opportunity for building your app on top of Business Central.
Get Started with Building Apps	Learn how to get started as a partner.

Resources for a reseller

Enroll in the CSP program

The Cloud Solution Provider (CSP) program helps your company to be more involved in your customers' businesses, beyond reselling licenses. In CSP, you can choose to enroll as an *indirect reseller* or a *direct bill partner*.

In most cases, you will enroll as an *indirect reseller* and then work with an *indirect provider*, also referred to as a *distributor*, who then manages all interaction with Microsoft in terms of licensing and technology, so that you can focus on sales and support. If you decide to enroll as a *direct bill partner* in order to fully own the end-to-end relationship with both customers and Microsoft, make sure that you meet the eligibility requirements. For more information, see Enroll in the Cloud Solution Provider program in the Microsoft Partner Center content.

The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Some indirect providers (distributors) provide their resellers with a custom portal that optimizes and enhances the experience beyond the Partner Center. They can also provide indirect resellers with an API to automate some of the customer onboarding steps. Contact your indirect provider to find out more.

Both indirect resellers and direct bill partners can access and support their customers' Business Central by setting up a reseller relationship with them.

To service customers in a specific country, your partner company's Azure Active Directory (Azure AD) tenant and CSP account must be registered in the regional CSP market that covers that country. For more information, see Cloud Solution Provider program regional markets and currencies.

> **NOTE**
>
> When you buy Business Central offers on behalf of your CSP customers, the CSP offer must be available in both your own tenant's country and in your customer's tenant's country. For example, if your tenant is located in Slovakia and the customer's tenant is in Germany, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.
>
> Similarly, if your tenant is located in Germany and the customer's tenant is in Slovakia, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.

The following table outlines recommended content for you to get started:

LINK	DESCRIPTION
Get Started as a Reseller of Business Central Online	Landing page for readiness for resellers.
Business Central on Microsoft Learn	Provides links to role-specific training.
Enroll in the Cloud Solution Provider program	Describes the different models for selling in the Cloud Solution Provider (CSP) program so you can determine which works best with your business.

Other roles and profiles

Business Central supports the same profiles and roles as other Dynamics 365 services. If you see yourself as a solution architect or functional consultant, for example, Business Central has functionality and content for you. Look around in the Help & Support section here, for example, ramp up on features in the business functionality content, or learn how you can integrate this with other products.

If you are completely new to Business Central, we recommend you take the Get started learning path.

See Also

Dynamics 365 Business Central Business Functionality Help
Dynamics 365 Business Central on Microsoft Learn
FAQ for Dynamics 365 Business Central Development and Administration
Resources for Help and Support for Dynamics 365 Business Central
The Lifecycle of Apps and Extensions
Maintenance of AppSource Apps and Per-Tenant Extensions
Microsoft Dynamics 365 Business Central on the Dynamics 365 blog
Dynamics NAV Developer and ITPro Content

The SMB Opportunity for App Publishers

2/17/2021 • 4 minutes to read • Edit Online

Our mission is to empower every individual and every organization on the planet to achieve more. This is particularly key for the 78 million small and mid-sized businesses (SMB) worldwide that increasingly want focus on tech intensity to be more resilient, more capable of transforming their products and services to face the constant changes and challenges in the marketplace. To embrace growth, every organization will have to have a business application which breaks down the silos of data, processes and workflow in their operations. This enables employees to respond to daily challenges and opportunities with agility. Market research shows that the business applications opportunity for software companies in the SMB space is predicted to be 51 billion dollars by 2025. As a developer, you want to make sure you bet on the winning platform. The BizApps market growth in this area is 17%; however Dynamics 365 platform growth has been growing at 47%!

Dynamics 365 Business Central

Dynamics 365 Business Central is an all-in-one business management solution that connects operations for small to mid-sized business. It ensures business continuity with a cloud solution that connects sales, service, finance, and operations to help teams adapt faster and deliver result. Business Central provides app publishers with a modern business platform that you can easily connect to, extend and build on — with little to no code required.

Microsoft AppSource

Microsoft AppSource is part of Microsoft's commercial marketplace where customers can find, try and get business solutions apps. It is the launch pad for your joint go-to-market activities with Microsoft and a flywheel for business growth. Using launch promotion, demand generation, and joint sales and marketing, your offer portfolio on AppSource can be the centerpiece of your cloud business engine.

In November 2020, Microsoft AppSource had more than 3 million monthly active users and more than +20.000 apps. Specifically for Business Central, there are more than 1000 apps available, and the number is growing fast.

Go-to Market Scenarios

As an AppSource publisher you can focus on these scenarios with Business Central and Microsoft AppSource:

- Connect
- Extend, or
- Embed

Connect

Connect your existing online service with Business Central through a powerful API. Here are a few examples which of existing connect apps which you can find on AppSource:

- Square payments, which allow you take payments with a Square terminal.
- Certify which allows users to post expense reports using Certify.com
- Scaptify, which connects your Shopify store with Business Central

Learn more about the API to build connect apps: Getting Started Developing Connect Apps

Extend

You can extend the default capabilities in Business Central to add extra productivity features or industry functionality to fit the needs of your customer base. The possibilities are plentiful.

In countries where Business Central is not localized by Microsoft, you can extend Business Central based on local requirements to respond to the regulatory and or competitive needs of that market.

Here are a few examples of some apps that extend Business Central:

- E-Ship and E-Receive from Lanham Associates, which extends warehouse management with barcoding and labelling, scanning functionality and interfaces with carriers and weigh scales.
- Continia Document Capture, which is an end-to-end solution for document recognition, invoice approval and digital archiving
- Philippines localization from Pasi, which computes the withholding tax as mandated by the Philippine government for both customers and vendors.
- SwissSalary 365, which is a certified and flexible payroll app that's intuitive and easy to use for the Swiss market

Learn more on how to build your app: Getting Started with AL

Embed

Embed Business Central as an integral component in an end-to-end industry solution. Here are a few examples of embed solutions available on Microsoft AppSource:

- 4PS construct, an all-in one solution for construction. 4PS has specifically developed an integrated software solution based on the Microsoft platform and tailored for construction, civil engineering, mechanical and electrical, service and maintenance, house builders and equipment rental companies
- LS Central from LS Retail. The complete, unified commerce online platform to manage your entire retail and food service operations
- Dynamics Empire Online, cegeka-dsa's open, secure and innovative real estate solution.

Learn more about embed apps: Embed app overview

Consultancy services

An increasing number of business users find web more convenient for buying where they can research, deploy and manage apps. Therefore, next to providing Apps, a publisher can also market consultancy services on Microsoft AppSource to connect with buyers. The provided content in these offers could be assessments, briefings, workshops, proof of concepts and implementations. In general the services are typically fixed in scope and duration, they are offered at a fixed price or free and have a defined outcome.

Here are a few examples of Consultancy services provided by publishers:

- Unified Commerce Readiness Intro: 1-Hr Assessment In this 1 hour assessment, LS Retail assesses how a 4-day engagement helps them with implementing their Unified Commerce Cloud solution based on Microsoft Dynamics 365.
- NAV-X Commission Mgt Gold 4-Hr Implementation in this 4 hour implementation workshop, NAV-X supports the customers in implementing their commission management app.

Learn more about Consultancy Services.

See also

Get Started with Building Apps

Get Started with Building Apps

2/17/2021 • 8 minutes to read • Edit Online

Dynamics 365 Business Central is a business management solution that helps companies connect their financials, sales, services, and operations to streamline business processes, improve customer interactions and make better decisions. With this modern business platform, you can easily and quickly tailor, extend, and build applications so they fit your specific needs with little to no code development.

Build a line-of-business app, such as for a specific industry, process, or department such as HR, finance, marketing, or operations. Then, publish your app to the Microsoft commercial marketplace, where customers can find and try your app, and get in touch with you. For more information, see What is the Microsoft commercial marketplace?.

Learn how you can become a Business Central app publisher in six steps in this article.

Step 1: Become a partner

Becoming a Microsoft partner gives you access to the Microsoft resources needed to build, market, and sell your apps. You don't have to be a Microsoft partner to begin developing your apps. But all of the steps below are required to gain access to the programs that enable you to publish, market, and sell your apps for Business Central.

Obtain your work account

Your work account or work email is the email address provided to you by your company. This email is usually in the format `you@yourcompany.com`. More information on work accounts can be found here.

Join the Microsoft Partner Network

Microsoft Partner Network (MPN) membership unlocks our best resources to differentiate your business, take your product to market, and sell your solutions. To become a partner, you must join the Microsoft Partner Network (MPN), at which time you will be assigned an MPN ID. MPN membership is free to all partners; you can enroll in the MPN here.

Once signed up, you will get an MPN ID – your gateway to access all the membership resources and benefits for your partnership with Microsoft. There is no cost to obtain a MPN ID as a Network member, and with options to upgrade to an Action Pack subscription or work toward a competency, you can access even more benefits.

Set up your Partner Center account

Once you have joined the Microsoft Partner Network (MPN), you can set up your Partner Center (PC) account. The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Your Partner Center account provides you with access to pricing information, tools and services, and enables you to manage admin credentials for your company's work account. Partner Center is also where you can purchase or renew subscriptions to Microsoft Action Packs, create a business profile to receive and manage sales leads from Microsoft, and see if you qualify for co-selling opportunities.

Step 2: Register as a publisher

The first step to becoming a publisher is to register in Partner Center (PC). PC is where you submit your apps for publication, promote your apps, and manage your offers. To begin the registration process, you must complete

these steps. One of our team members will follow up to help you complete your registration. Once registered, you can access PC.

For more information, see Partner Center Account.

PartnerSource Business Center (PSBC) account

Developing apps requires you to be known as Business Central developer and requires you to have a unique development license file with a specific object range.

To obtain an object range for developing a Business Central, you must first have access to PartnerSource Business Center (PSBC). You have access to PSBC if you have one of the following agreements:

- An active Partner Registration Agreement (PRA)
- A Registered Solution Program Addendum (RSPA)

The relevant contract can be requested through your local Regional Operations Center (ROC) Contracts and Agreements Team below:

- mbscon@microsoft.com if you are based in Europe, the Middle East, or Africa
- mbsagree@microsoft.com if you are based in the Americas
- mbslques@microsoft.com if you are based in the Asia Pacific region.

Step 3: Your unique app specifications

Requesting an object range

When you develop an app for Business Central, you must request access to an object range that holds a certain number of objects for your solution. To avoid overlap between objects used in different solution, each partner is assigned a number of objects in a unique object range. For example, a partner is assigned the object range 70,001,000 – 70,001,999. The object range gives them 1000 numbered objects that they can use to develop Business Central solutions.

Depending on where you will deploy your Business Central solution, online or on-premises, you can use different licensing methods and object ranges.

There are currently two available ranges that you can request. Both have some characteristics to keep in mind:

- *RSP Object Range* (ID range 1,000,000-69,999,999)

  This object range is tied to the RSP Program details.

  > **IMPORTANT**
  > We currently advise new publishers to *not* request an RSP object range

- *App Object Range* (ID range 70,000,000-74,999,999)

  This object range was originally designed just for apps in the Microsoft commercial marketplace to be used in Business Central online.

  > **IMPORTANT**
  > We currently advise new publishers to request an app object range.

Currently, you can implement apps developed in both the *RSP range* and the *app object range* in Business

Central online and on-premises, as well as partner-hosted.

You can request an object range by downloading the object range request form here. After completion, send them to your Regional Operational Center (ROC) for processing:

- mbscon@microsoft.com if you are based in Europe, the Middle East, or Africa
- mbsagree@microsoft.com if you are based in the Americas
- mbslques@microsoft.com if you are based in the Asia Pacific region.

Downloading your development license file

After your Regional Operational Center has processed your Agreements and Object Range Request forms, download your company's unique developer license from PartnerSource Business Center. Find it in the license key configuration section under the developer tools section.

Register your unique prefix or suffix

In your extension, the name of each new application object must contain a prefix or suffix. This rule applies to all objects. Email d365val@microsoft.com with a prioritized list of three-letter affixes of your choice. You must also submit your MPN ID and the publisher name that you will use for the app.

Step 4: Getting access to preview bits

Get access to preview builds by joining Microsoft Collaborate.

In Microsoft Collaborate, you get access to a set of Business Central builds:

- The current major version
- An upcoming major version
- Daily builds

You must have the following prerequisites to register on Microsoft Collaborate:

- Azure Active Directory (Azure AD).

  > **NOTE**
  > If you have Microsoft 365, then your company most likely has Azure AD

- Azure AD Global Administrator permission

  > **NOTE**
  > To find out if your company has an Azure AD account, check with your system administrator.

Step 4 A: How your Global Administrator must register for Collaborate

Only your company's Global Administrator can start the onboarding to Collaborate. They must register at https://aka.ms/Collaborate, choose the **Get Started** action, and then complete the registration form.

The administrator can then add the relevant colleagues.

Optional Step: Add your coworkers to Microsoft Collaborate

To add coworkers:

1. Sign in to Microsoft Collaborate with your Global Administrator account at aka.ms/Collaborate.
2. Choose the  icon in the top-right corner of the page, click on account settings, and choose **user**

management.

3. Choose the grey **ADD USERS** button, and leave the default choice to **Add existing users** as-is. Now you can search for the user(s) that you want to add to Collaborate. To add them you need to select them from the menu, and then click the grey **ADD SELECTED** button.

4. You have successfully added your coworkers to Collaborate. Users can now sign in to Microsoft Collaborate using the following link: aka.ms/Collaborate

Step 4 B: Getting access to the available builds and engagements

Once you have successfully registered on Microsoft Collaborate, Microsoft must assign you to the right programs and engagements before you can see the preview bits. Contact Dyn365BEP@microsoft.com and provide them with information about the relevant users. the following table illustrates the type of information that you must submit:

PUBLISHER DISPLAY NAME	MPN ID	FIRST NAME	LAST NAME	WORK ACCOUNT EMAIL
Contoso	12345	Eugenia	Lopez	Eugenia.Lopez@Contoso.com
Contoso	12345	Quincy	Watson	Quincy.Watson@Contoso.com

After sending the email, expect a response from Microsoft within 1-2 business days.

Step 5: Resources while you develop your solution

Find below some guiding resources on how to develop your apps for Business Central.

- Microsoft Learn

  Learn new skills and discover the power of Microsoft products with step-by-step guidance. Start your journey today by exploring our learning paths and modules.

- Microsoft Docs

  Find The developer and administration content on Microsoft Docs

- Join the conversation

  In the dedicated Yammer network, join the conversation on developing apps

- Join the monthly Office hour calls

  Join the monthly Office hour calls to learn more about a hot topic

- Get coaching from experts

  Need help with developing your apps? There is a community of ISV Development Centers specialized in Business Central ready to engage with you.

Set up Azure DevOps for your development processes

Optionally, use the CI/CD workshop document at https://aka.ms/cicdhol if you want to do it yourself. Alternatively, choose between vendors offering tools or services around DevOps.

Talk to one of the ISV Development Centers for guidance.

Step 6: Publish your app in the Microsoft commercial marketplace

Once your app is ready for submission, you can list your app in the Microsoft commercial marketplace by submitting it in Partner Center. For more information, see Create a Dynamics 365 Business Central offer.

Before you submit, we recommend that you review the technical validation checklist and marketing validation checklist. The two articles list all requirements that you **must meet before you submit** an app for validation. If you do not meet these mandatory requirements, your extension will fail validation

See also

The SMB Opportunity for App Publishers
The Lifecycle of Apps and Extensions for Business Central
Update Lifecycle for AppSource Apps FAQ
Maintain AppSource Apps and Per-Tenant Extensions in Business Central Online
Microsoft Responsibilities for Apps on Business Central online

The Lifecycle of Apps and Extensions for Business Central

2/17/2021 • 6 minutes to read • Edit Online

When you build an app or extension to Business Central and get that published to AppSource, it becomes an app like so many others - the app itself can be updated, and the platform that it sits on, Business Central online itself, will also get updated. But what happens after your app gets published?

When your app has passed all of our validations and has gone live to App Source, customers can install your extension and use it for their business. But you are expected to keep it compliant with the service and update it if something changes.

The following sections describe the different upgrade scenarios that we have seen play out as we update Business Central. For more information about your responsibility for keeping your app updated and the resources that are available to you, see Maintain AppSource Apps and Per-Tenant Extensions.

Scenario 1: Business Central service update

You don't need to make any bug fixes, feature adds, or app changes to your app. It continues to work fine without any interaction on your part.

Impact of service updates

The monthly service upgrades to Business Central do not impact your app. Your app just gets moved along and no upgrade code from your app needs to get used. Business Central itself gets upgraded on your tenant, and once complete, the customer sees no difference with your app.

Scenario 2: App update

You (our partner) add some features to your app and also some minor bug fixes. The app is submitted for validation. The app passes validation and gets checked into the service. This is now the active app for any new tenants and also for existing tenants that have never had your app installed before

Impact of app updates

Customers can either do an uninstall and then reinstall on their own, or they can ask their partner do it on their behalf from the Extension Management window within Business Central. Otherwise, they would have to wait until our every 6-month major release. That is the only time we do a force upgrade of extensions (except for critical bug hotfix extension updates)

Scenario 3: Reported bugs in your app

You (our partner) has various customers report some bugs that are impacting their usage of the app. The bugs aren't critical but they are important. The partner makes the fixes in the app and resubmits for validation. The app passes validation and gets checked into the service. This is now the active app for any new tenants and also for existing tenants that have never had your app installed before

Impact of bugs

We still do not force the upgrade of this app to this latest version on all of the tenants. Some tenants may not be using the functionality that includes this bug and continue to work fine on the current version of the app. Therefore, you should work directly with all of the impacted customer tenants to uninstall and reinstall to get the latest app version that contains fixes for the bug.

Scenario 4: Critical bug in your app

If a bug which leads to core functionality being broken or data loss/corruption/misrepresentation is found in the application, and the issue prevents customers from performing time-critical tasks, the validation and deployment of the application can be prioritized. The partner must create a support ticket for this case and they must immediately provide a fixed app for validation through Partner Center. The validation team makes this a top priority and does validation as soon as possible. If the fixed application passes validation, it will be checked into the service and will become available for environment administrators to install.

Scenario 5: Microsoft feature breaks your app

Microsoft has to break your app file for a needed Business Central core change. Some reasons for breaking could be security, bugs in the underlying code, high priority feature adds, and so on. Keep in mind, we do our very best to not break your app through our changes. We try and find proper ways of doing the changes without breaking your app. However, if we can't find a proper (non-breaking) way, then we could break your app. This won't be as likely in a minor update release (unless a security change is required on our part and that is the change that breaks you), but it can be more likely in our major (every 6-month) releases.

Impact of breaking changes

Here is our process when this takes place:

- First of all, Microsoft will not make a breaking change in the production environment at any point. Therefore, existing tenants are not expected to see this breaking change occur.
- When we make a breaking change, we do it in a build branch that is for a future release (monthly service minor or major release)
- We notify the partner in advance and give the partner ample time to fix their app, get it validated, and have it ready
- The fixed app will already be in our service and slotted as required for when your tenant is to be moved to the Business Central release that has the app breaking change
- As a result, the customer (tenant owner) should never see their Business Central break. Because the tenant gets moved from one monthly service update of Business Central to another, the tenant is being upgraded to the release of ours that breaks the specific app. However, our service detects that there is a new required version of that app (your fixed version). Therefore, we auto install the fixed version of the app for the tenant

Conclusions

You're responsible for your app. You own the process of updating the app and providing upgrade code if the schema changes between versions of the app.
If a customer uninstalls your app, and then installs it again later, then when they install the app the second time, they get the latest version from AppSource.

How Microsoft handles your app

When Microsoft upgrades a tenant with a service update, your app is tested against the new service version. If the app breaks, Microsoft rolls back to the previous healthy state. Your customer never learns that anything was about to break.

When a tenant uninstalls and reinstalls an extension via the Extension Management page or AppSource, there is platform logic that determines whether an *Install* or an *Upgrade* must take place. We detect which version of the extension the tenant previously had installed and perform the appropriate action. Therefore, the result of manually uninstalling/installing the extension is the exact same as an automated upgrade.

Additionally, there will not be any data loss during uninstall, install, or upgrade actions. Data for extensions is stored in its own tables in the tenant database. Before an extension gets installed, it first get synchronized on the tenant database. This step is implicit and happens automatically when a tenant installs an extension. This

synchronization process creates the database tables for the extension. Once the extension is installed and the tenant is using it, extension-specific data will get stored in these tables.

When an extensions gets uninstalled, these tables do not get removed. Therefore, when the extension gets reinstalled (or upgraded), the data is still available. You do not need to worry about data loss for choosing the uninstall/install route. However, do keep in mind that if any actions are being performed on the tenant while the extension is uninstalled, the extension's events and such will not be firing, and your app may miss the creation of new data. Try to perform the uninstall/install while the tenant is not online.

For more information, see When apps or PTEs cannot be updated by Microsoft.

See Also

Publishing and Installing an Extension
Retaining table data after publishing
Upgrading Extensions
Add your App to AppSource
Checklist for Submitting Your App
Upgrading AppSource Apps in Production
Maintain AppSource Apps and Per-Tenant Extensions

Update Lifecycle for Customizations of Business Central Online

2/17/2021 • 2 minutes to read • Edit Online

When you create a tenant-specific customization, or an extension that is scoped to a single Business Central environment (often referred to as per-tenant extensions), you must take the lifecycle of the extension into consideration. For more information about the extension lifecycle and events that can cause incompatibilities between the extension and base application, see The Lifecycle of Apps and Extensions for Business Central.

You are responsible for your extension. You own the process of updating the extension and providing upgrade code if the schema changes between versions of the Business Central base application. When an update is available for your Business Central environment, all extensions, both AppSource extensions and tenant customizations, must be compatible with the next version of the base application before the update can be installed on the environment. You are responsible for ensuring that your extensions are compatible with the update version.

In this article, we describe the process for ensuring update compatibility for tenant customizations.

Automated Extension Validation

An automated service validates all tenant customizations before an update is marked for a scheduled update of an environment. This validation compares the extension's dependencies with the updated version of the Business Central application to see if any changes caused incompatibilities with the new version.

If the validation service discovers any tenant customizations that are not compatible with the update, an email notification is sent to the tenant administrators listed on the **Notification recipients** tab of the Business Central administration center. For more information, see Managing Tenant Notifications.

> **IMPORTANT**
> At least one email address must be specified as a notification recipient to receive the update notifications. If you do not speciy an email address, you will not be notified of updates and other changes to the tenant.

The email notification provides information on the incompatible extension, detail on which properties must be updated, and steps to bring the extension into compatibility.

Update Failure Notification

If a Business Central environment has an extension that is not compatible with the update version, the update cannot be applied. If an update fails due to an incompatible extension, a notification is sent to the tenant administrators listed on the **Notification recipients** tab of the Business Central administration center.

The notification is similar to that provided by the automated extension validation. It provides information on the incompatible extension, detail on which properties must be updated, and steps to bring the extension into compatibility.

Automatic Extension Removal

The publisher of an extension must maintain compatibility with the new release of Business Central. An extension that is not compatible with the update within 90 days of the first notification of incompatibility will be

removed, and then the environment will be updated.

See Also

Retaining table data after publishing
Upgrading Extensions
Updating Environments

Maintain AppSource Apps and Per-Tenant Extensions in Business Central Online

2/17/2021 • 7 minutes to read • Edit Online

As a partner, keeping your apps and per-tenant extensions (PTEs) up to date is your responsibility. Business Central is regularly updated with major and minor releases. These updates provide customers with a business application that is always compliant, secure, and enriched with new platform and application functionality. Often customers choose Business Central because of this promise of having an always up-to-date business solution.

To not break this promise, developers that bring apps to Microsoft AppSource, and resellers that provide PTEs to respond to the unique needs of customers, have a responsibility to align their code to the Microsoft release rhythm.

An inability for Microsoft to update tenants because of publishers incompatible code causes serious disruption in the service and must be avoided since it impacts the trustworthiness of the service and customer satisfaction.

Resources

To help app publishers keep up with their update responsibilities, Microsoft provides the following resources:

- Release plans about what's new and planned

  For more information, see Dynamics 365 release plans.

- Access to pre-release bits

  Business Central partners have access to the next major, next minor, and daily pre-release bits in Docker. These bits can be used to test apps against upcoming updates.

- Information about what will be deprecated

  With all Business Central releases, Microsoft controls and regulates breaking changes with major releases and communicates upcoming breaking changes at least one year in advance. If developers missed this above info, the compiler in Visual Studio Code also warns for potential controls that will become obsolete in future versions and how to deal with them.

- Policy definitions and terms

  The publisher agreement and the commercial marketplace certification policies describe the responsibilities of app providers for how to publish and maintain apps in the Microsoft monthly rhythm.

  When a PTE gets installed, the publisher also agrees to the terms to keep that code current and updatable.

- Training and coaching

  Microsoft provides a set of tools, training, and documentation to help partners find the info they need to keep up with these responsibilities on continuous integration and continuous deployment. External providers, including ISV Development Centers, MasterVARs, and training centers, can provide in-person training and coaching.

- Service notifications

  Business Central online will support app and PTE publishers with extra warnings about potential technical incompatibility. If publishers respond to these notifications in due timing and avoid incompatibilities

repeatedly, Microsoft will stand with these publishers to help where needed. If a publisher includes a telemetry key in their app, then, starting with 2020 release wave 2, Business Central also provides publishers with telemetry about upgrade failures that happen in production because of issues in the publisher's upgrade code.

If publishers lack to keep their code updatable, they risk that ultimately their apps or PTEs will be removed from the customer's tenant, and this will most likely result in important data not being captured as it should. For apps, this also means removal from the marketplace.

Since resellers are the first line contact point for customers, they carry responsibility to explain what it means to load code in a customer's environment. The best way is to explain this is with terms.

We advise these terms include topics like intellectual property rights, upgrade responsibilities, associated costs to keep code updatable, support options, data privacy, and so on.

Pre-release publisher support to keep apps and PTEs compatible and up-to date

Publishers have several tools available for them to keep their code in good shape. Not least, a Public Preview release is made available approximately one month before the announced release date for a major release. In that Public Preview release time frame, Business Central will automatically test and notify publishers of existing apps and PTEs running in production on technical incompatibility with the upcoming release.

> **IMPORTANT**
> Microsoft tests code based on technical compatibility. As the publisher, you are still responsible for all functional and logical validation.

When apps or PTEs cannot be updated by Microsoft

This section describes the processes that are initiated during and after upgrade attempts of code provided by publishers of apps or PTEs.

- **T1 – T30**: Microsoft alerts administrators, resellers, and ISVs

  Shortly after a service update of Business Central online (Day **T**), Microsoft will initiate daily updates attempts on all tenants. In these update attempts, the publisher's provided upgrade code is triggered and run. These attempts run repeatedly in a time frame of approximately one month until the upgrade is successful. For more information, see Major Updates of Business Central Online.

  With every unsuccessful upgrade attempt, stakeholders will receive notifications. Customers and their reselling partners can follow these notifications in the Business Central administration center.

  ISVs who provide third-party AppSource apps might not be listed in the customer's admin center. The reseller will in most cases have worked with the ISV to test compatibility, but after two weeks (Day **T+15**) of failed upgrade attempts, the Microsoft AppSource team will also send the app provider a warning message that action within the next few days is required.

  This message will explain that if they fail to respond correctly, their app will be removed from AppSource at Day **T+30**.

- **T30 – T60**: Microsoft alerts the customer

  After one month of failed upgrade attempts (Day **T+30**), the customers will be notified again that apps or PTEs are incompatible with the new version of Business Central, and that no further automatic upgrade attempts will be planned until further notice. Although the publisher's code continues to run on an

outdated version of Business Central online, the customer must work with their reseller to resolve these issues immediately so that the tenant can be updated. Next to messages in the Business Central administration center, all users in the customer's tenant will also get more active warning about the incompatibilities when they use the product in the browser or their mobile device.

For AppSource apps, if no appropriate action or follow-up was taken by the publisher since the release (**T - T+30days**), the app will be removed from AppSource. This means no new customer will be able to install the app in a new tenant. The main reason for removing an app from the marketplace is to ensure that no new customers will be affected by incompatibilities with the latest version of the base product, Business Central.

If the publisher wants to have their app available again, they must mitigate all existing incompatibility issues and go through the full validation process again.

If the source of the incompatibility has been resolved by the publisher, they'll have to submit a support request to schedule a new set of upgrade attempts for any tenants that are blocked because of this incompatibility. They'll also have to work with their resellers to inform them about the compatibility resolution.

- **T60 – T150**: Microsoft initiates the customer wind-down period

  If the incompatibility issues are not resolved at **T+60**, and the publisher remained unresponsive to the request to resolve the incompatibility, Microsoft may choose to send out a wind-down communication to the customer about removal of the publisher's code.

  This communication will share that the code from the publisher will be removed in *90* days (**T+150**).

  During this wind-down time, the customer and their reselling partner are fully responsible for finding a solution on how to proceed in this situation. If the customer decides to leave Business Central, or decides to use another publisher, they can access their data by exporting the database, use RapidStart Services, or copy data to Excel. For more information, see Exporting Your Business Data to Excel in the business functionality content.

  Microsoft may also choose to remove all existing apps by this publisher from AppSource and block the publisher from publishing new apps for Business Central.

  If this wind-down period is initiated, and the customer was able to fix the incompatible issues with their reseller and potential publishers, it will be at Microsoft discretion if the publisher's code will be removed from Business Central or not.

Get notified about incompatibilities by Microsoft

It is crucial for you to keep contact details correctly up to date. We advise you to use global team aliases instead of individual mail addresses. Here are the mail addresses that we'll use in the above process:

- PTE publishers

  The mail addresses specified in the Business Central administration center; this could be both a customer user and a partner user.

- App publishers

  The mail addresses specified during App publication in the partner center during app publication as support and engineering contact details.

- Customers

  The mail addresses specified in the Business Central administration center; this could be both a customer and a partner user.

See also

The Lifecycle of Apps and Extensions
Update Lifecycle for Customizations
Microsoft Responsibilities for Apps on Business Central online
Technical Support for Business Central online
Sending Extension Telemetry to Azure Application Insights

Microsoft Responsibilities for Apps on Business Central online

2/17/2021 • 2 minutes to read • Edit Online

Business Central online is a cloud service for small to medium-sized businesses that is built on and for Microsoft Azure. Business Central brings together the business management solution, business intelligence, infrastructure, computing, and database services in a single offering that enables organizations to run horizontal or industry-specific apps from Independent Software Vendors (ISVs), without the hassle of managing infrastructure.

The Business Central online model distinguishes specific roles and responsibilities for partner-provided vertical solutions, system integrators, resellers, and Microsoft throughout the life cycle of the service. Microsoft maintains the Business Central service by deploying, actively monitoring, and servicing the customers' production tenants that are running on the service. This includes allocating the required system infrastructure to run the service and proactively communicating to customers about the service's health (which is done through the Service Health dashboard in the Microsoft 365 Admin Portal).

Microsoft responsibilities in the Business Central service include:

AREA	RESPONSIBILITIES
Infrastructure	• Storage and database capacity management • High availability and disaster recovery • Platform security and compliance • Infrastructure capacity, scaling in response to demand • Infrastructure management and deployment • Data center networking, power, and cooling
Base application and platform	• Availability and security • Diagnostics, patches, updates, hotfixes, and updates • Monitoring and first-line support for partners building apps
Lifecyle Services portal (EmbedApp program)	• Development, deployment, and support of the portal functionality • High availability and disaster recovery • Monitoring, updating and patching • First-line support for partners

See Also

Get Started as a Reseller of Business Central Online
Maintain AppSource Apps and Per-Tenant Extensions in Business Central Online
Administration of Business Central Online
Technical Support for Dynamics 365 Business Central

Components and Capabilities

2/17/2021 • 2 minutes to read • Edit Online

When you build an app for Business Central, be it an Embed App or an AppSource app, you must be aware of which components and deployment steps are provided by Microsoft and which you must provide.

Components

Base application

The base application is the Business Central application provided by Microsoft, customized and extended to fit the needs of a market segment that an app wants to serve. Major releases and cumulative updates (CUs) of the base application are publicly available on Microsoft Download and as artifacts for use with Docker.

Pre-released versions of the base application are available as artifacts for participants of the "Ready and Go" program via Microsoft Collaborate for use with Docker. Although we recommend always using the latest version of the base application, partners can choose any version they need. The only requirement is that the partner makes sure that the base application version that they base their app on is available in Business Central online production environments.

AppSource apps must specify the dependency of a specific base application in the settings for their app. In contrast, the base application is an optional part of an Embed App package. If the partner has implemented all required functionality in their library extensions, they do not have to include the base application itself with the Embed App. Instead they should specify, in the metadata of the Embed App, which version of the Business Central base application they are targeting, just like an AppSource app.

Microsoft recommends all partners to move towards a model where the code-customization of the base application is not used. Keep track of new capabilities in the base application and platform in the release plans.

Platform

Partners who build apps for Business Central online must make sure that it is compatible with a supported version of Business Central platform.

The Microsoft platform is updated in the following way:

- Minor updates

  Microsoft will ship minor platform updates monthly and major platform updates every six months. Minor updates can include bug fixes and improvements which should not affect the compatibility of the platform with the previous version of the application. In rare situations, partners may be asked to recompile their solution to work with a minor update of a Business Central platform.

- Major updates

  Major updates will include changes that can require partners to perform a technical upgrade to make their application work with the new version of the platform. For more information, see the Dynamics 365 release plans

Ecosystem

Business Central online is part of a rich ecosystem of other Microsoft and 3rd party services, which partners and customers can decide to take advantage of.

The following integration capabilities of the Business Central can be considered:

INTEGRATION CAPABILITY	STATUS
Dynamics 365 API endpoint	Available if the base application objects are unchanged
Outlook Add-in	Available
Power BI	Available if the customer has a Power BI license
Power Automate	Available if the customer has a Power Automate license
Power Apps	Available
Microsoft 1st party integration apps included with Business Central (Yodlee, Quick Books, OCR, AMC, and others)	Available, but in many cases the partner must set up a separate agreement with these service providers
Azure machine learning	Available if the partner has an Azure ML subscription
Integration with Dynamics 365 Sales	Available
Microsoft Graph	Currently under evaluation
Accountant Hub	Available in specific countries

See Also

Microsoft Responsibilities
Embed App Overview

Add your App to AppSource

2/17/2021 • 2 minutes to read • Edit Online

AppSource is a market place where partners can provide marketing details, such as descriptions, whitepapers, or videos about their app for Business Central.

Embed App partners can choose to promote themselves and their Embed App on AppSource.

> **IMPORTANT**
>
> Unlike all other apps, solutions that are part of the Embed App program are not uploaded to AppSource itself. Instead, the app package is uploaded, deployed and tested via Lifecycle Services. AppSource in this case is used for the marketing purposes, not as a repository of Apps.
>
> All other apps do submit their app to App Source. For more information, see Technical Validation Checklist and Marketing Validation Checklist.

See Also

Get Started as a Reseller of Business Central Online
Build Your Business on Dynamics 365 Business Central
Marketing Validation Checklist
Technical Validation Checklist
Embed App Overview

Marketing Validation Checklist

2/17/2021 • 3 minutes to read • Edit Online

The storefront details on AppSource is the first impression that prospects get regarding your offer. First impressions last, so make sure to invest some time in developing the content on the storefront, so it gives a good impression from the beginning. Failing to do so will jeopardize the hard work you put in, when developing your offer, likely leaving the prospect confused or looking elsewhere. Accordingly, we recommend you put in the time, effort and due diligence when developing this content.

You use Partner Center to submit your offer to AppSource. In the following, you can find information on all the marketing-related items that you need to fill out in Partner Center prior to submitting your app to AppSource. Follow this marketing validation checklist and get your app passed on the first submission.

What do I need to know before I begin?

You need to have the following three items in mind, when you are creating your storefront and marketing material.

ITEM	REQUIREMENT	DETAILS
Branding	Make sure to read the branding guidelines carefully before you start referencing the product name.	Read more
Language	Discover what to consider for each storefront detail when it comes to language.	Read more
Microsoft images	Make sure not to use any Microsoft images (e.g., Business Central icon or Dynamics 365 logo).	Read more

How do I fill out the marketing section in Partner Center?

The following section walks you through the marketing-related components, that is, *offer setup*, *properties*, *offer listing*, and *availability*, which you need to fill out in Partner Center prior to submitting your app to AppSource.

Offer Setup

ITEM	REQUIREMENT	DETAILS
App type	Read more about the types of apps you can submit to AppSource.	Read more
Contact type	How do you want potential customers to interact with your offer?	Read more
Test drive	If you choose “free trial” as the contact type of your offer, you need to enable a test drive.	Read more

ITEM	REQUIREMENT	DETAILS
Customer leads	Provide connection details to the CRM system where you would like to send customer leads.	Read more

Properties

ITEM	REQUIREMENT	DETAILS
Categories	Choose a primary, a secondary, and up to four subcategories.	Read more
Industries	Choose up to two industries for your offer.	Read more
App version	Specify the version number of your offer.	Read more
Terms and conditions	Outline the terms and conditions that the customer must accept before they can use your offer.	Read more

Offer listing

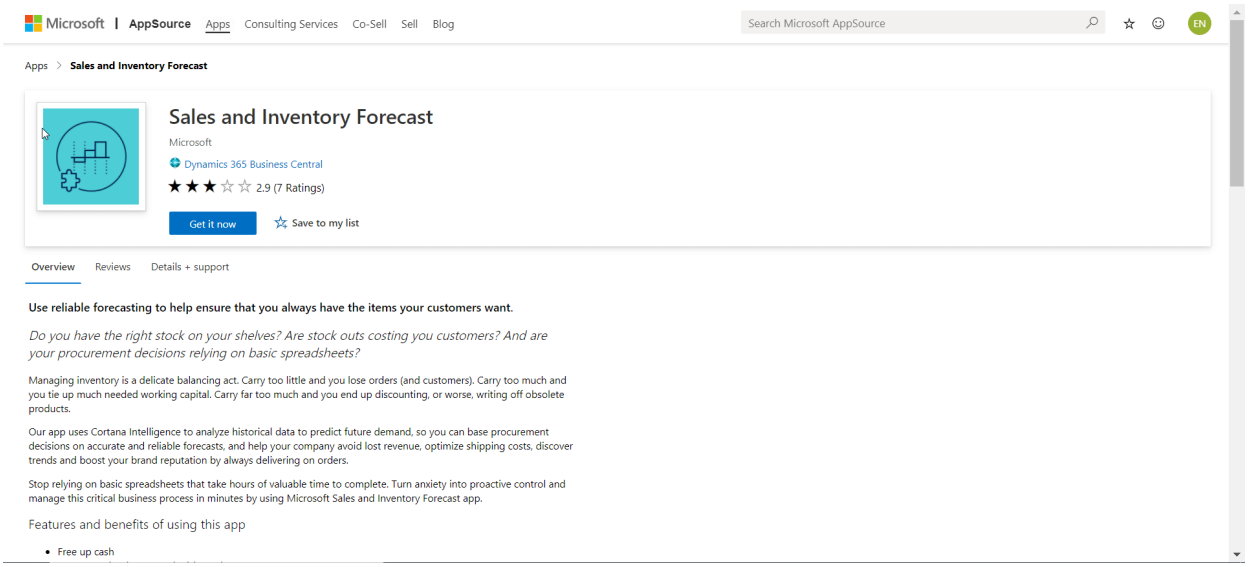
ITEM	REQUIREMENT	DETAILS
Offer name	Enter a descriptive name for the offer.	Read more
Offer summary	A single sentence summarizing the purpose or function of the offer.	Read more
Description text	Make an elaborate and compelling description that outlines the benefits and usage scenarios of your offer (include supported editions, countries, and languages).	Read more
Search keywords	Add search keywords to help users find your offer when they search in the marketplace.	Read more
Products your app work with	Add specific products that your app works with.	Read more
Help link	The link to your offer's learning resources.	Read more
Privacy policy link	The link to your offer's privacy policy.	Read more
Support link	The link to your offer's support page.	Read more
Supporting documents	Add supporting sales and marketing assets such as white papers, brochures, checklists, or PowerPoint presentations.	Read more

ITEM	REQUIREMENT	DETAILS
Logos	Upload a large size logo file with dimensions between 216 pixels x 216 pixels and 350 pixels x 350 pixels.	Read more
Screenshots	Provide a minimum of 3 screenshots that showcase your offer.	Read more
Videos (optional)	Add up to 4 videos that demonstrate your offer. These should be hosted on an external video service.	Read more

Availability

ITEM	REQUIREMENT	DETAILS
Markets (countries)	Choose the markets that your app is available in (make sure that it resembles the supported countries paragraph in the description text).	Read more
Hide key	The hide key is a token that is used to view the preview of your offer in AppSource before going live.	Read more

How does my offer look when it's live on AppSource?



Marketing Validation FAQ

2/17/2021 • 2 minutes to read • Edit Online

Where do I state the countries, editions and languages that my offer supports?

You are required to state the countries, editions and languages that your offer supports in the very bottom of your offer's description text. You can use the following format:

Supported Editions:

The app supports the Essentials and Premium Editions of Microsoft Dynamics 365 Business Central.

Supported Countries:

Canada, Mexico and United States

Supported Languages: This app is available in English (United States) and Spanish (Mexico).

Do you have any tips and tricks for what I should write in the description text?

Yes. We have detailed guidelines and good advice about that here.

How do I refer correctly to the product?

When you mention the product, both throughout your description text, as well as in your marketing material, you need to refer to the product, in the following way:

First mentions: Microsoft Dynamics 365 Business Central

Secondary mentions: Dynamics 365 Business Central

Subsequent mentions: Business Central

Therefore, you cannot use any abbreviations such as "MS Dyn 365 BC" or "Microsoft Dynamics NAV".

What are the requirements for my offer's help and support page?

You are required to submit two distinct pages for support and help i.e. they cannot be the same. You can see what you have to include on both pages in the table below.

HELP	SUPPORT
Learning material such as FAQs, step by step guides, video tutorials, webinars etc.	At least two contact options (e.g. e-mail, phone, chat) and a defined SLA for how much time it takes before you answer to support inquires.

Can I use the Microsoft Dynamics 365 Business Central logo?

No, you cannot use the Business Central logo, as it's a Microsoft trademarked logo. However, you can use the "Get it from Microsoft AppSource" badge, which you can find here.

See Also

Marketing Validation Checklist

Technical Validation

2/17/2021 • 7 minutes to read • Edit Online

Below you will find a checklist of all requirements that you **must meet before submitting** an extension for validation. You will also find a description of how the Business Central Validation team is performing technical and manual validation and how you can implement a validation pipeline to perform the same technical validation yourself.

Technical Validation Checklist

If you do not meet these mandatory requirements, your extension will fail validation. To get code validation helping you bring your extension package to AppSource, you can enable the **AppSourceCop** code analyzer. For more information, see Using the Code Analysis Tool.

REQUIREMENT	EXAMPLE/GUIDANCE
Develop your extension in Visual Studio Code.	Developing AL Language extensions
The app.json file has mandatory settings that you must include. Here you can also read more about dependency syntax and multiple countries per a single app syntax.	Mandatory app.json settings
Coding of `Date` must follow a specific format (**no longer region-specific**)	Use the format `yyyymmddD`. For example, `20170825D`.
Remote services (including all Web services calls) can use either HTTP or HTTPS. However, HTTP calls are only possible by using the HttpRequest AL type.	Guidance on HTTP use
Only JavaScript based Web client add-ins are supported. The zipping process is handled automatically by the compiler. Simply include the new AL `controladdin` type, JavaScript sources, and build the app.	Control Add-Ins
The .app file must be digitally signed.	Signing an APP Package File
The user scenario document must contain detailed steps for all setup and user validation testing.	User Scenario Documentation
Set the application areas that apply to your controls. Failure to do so will result in the control not appearing in Dynamics 365 Business Central.	Application Area guidance
Permission set(s) must be created by your extension and when marked, should give the user all setup and usage abilities. A user must not be required to have SUPER permissions for setup and usage of your extension.	Exporting Permission Sets Managing Users and Permissions
Before submitting for validation, ensure that you can publish/sync/install/uninstall/reinstall your extension. **This must be done in a Dynamics 365 Business Central environment**.	How to publish your app

REQUIREMENT	EXAMPLE/GUIDANCE
Thoroughly test your extension in a Dynamics 365 Business Central environment.	Testing Your Extension
Do not use `OnBeforeCompanyOpen` or `OnAfterCompanyOpen`	Replacement Options
Include the proper upgrade code allowing your app to successfully upgrade from version to version.	Upgrading Extensions
Pages and code units that are designed to be exposed as Web services must not generate any UI that would cause an exception in the calling code.	Web Services Usage
You are required to prefix or suffix the **Name** of your fields and objects. This eliminates collision between apps.	Prefix/Suffix Guidelines
We strongly recommend you are using automated testing, using the AL Test Toolkit. You are not required to include the test package with your extension.	Testing the Advanced Sample Extension
DataClassification is required for fields of all tables/table extensions. Property must be set to other than `ToBeClassified`.	Classifying Data
You must use the Profile object to add profiles instead of inserting them into the **Profiles** table.	Profile Object
Use `addfirst` and `addlast` for placing your actions on Business Central pages. This eliminates breaking your app due to Business Central core changes.	Placing Actions and Controls

Technical validation performed by the Business Central validation team

The primary responsibility of the technical validation is to ensure that the Business Central online service is stable and that the apps can be installed and run without destabilizing the service.

The technical validation is for a large part automated and will validate the steps described in the technical checklist above through some pipelines.

The submitted apps will be extracted and investigated following this list:

1. The apps are investigated. All dependencies must be included in the submission. We will lookup prior versions of the apps in the depot. If your app has a dependency on a third party app in AppSource, you should not include this, we will locate it in the depot. **Any unresolved dependencies will cause the submission to be rejected.**
2. If the version numbers haven't changed and the countries haven't changed, the validation is skipped and **the apps will not be updated.**
3. App.json is investigated for mandatory fields. **If mandatory fields are missing, the submission is rejected.**
4. Affixes for the submission are located. **If affixes haven't been registered or cannot be located, the submission is rejected.**
5. Business Central Artifacts are located for the version the apps is submitted for (*Current*, *NextMinor*, or

NextMajor).

6. For every country in the submission list, we perform the same validation:
   - A sandbox container based on the Business Central Artifacts with the right country version is created.
   - Any dependency apps not included in the submission are installed. **If any installation fails, the submission is rejected.**
   - In order of dependencies, all apps in the submission are tested using AppSourceCop analyzer. For more information, see AppSourceCop Analyzer
     - If any **breaking changes are identified, the submission is rejected.**
     - If mandatory affixes **are not included on all object names, the submission is rejected.**
   - In order of dependencies, all prior versions of the apps are published and installed. **If any installation fails, the submission is rejected.**
   - In order of dependencies, all new versions of the apps are published and upgrade is run (apps must be digitally signed, else they won't install). **If any installation/upgrade fails, the submission is rejected.**
   - A simple connection test is run; opening a role center and check simple actions and pages. If the connection test fails, **the submission investigated and potentially rejected.**
7. If all country validations succeed and no errors are found then **the submission is accepted.**

> **IMPORTANT**
> Microsoft recommends that all partners are performing the same checks as described above before submitting apps for validation to maximize chances of validation success.

Running technical validation yourself

With the latest version of BcContainerHelper, you can run a single command, which should perform the same validation steps and give you a good indication of whether your apps will pass validation or not:

```
$validationResults = Run-AlValidation `
    -licenseFile "path/url to license file" `
    -validateCurrent `
    -installApps @( "path/url to your foreign dependencies, apps which will not be part of the validation
(or blank if this is the first)" ) `
    -previousApps @( "path/url to your previous version of the .app files (or blank if this is the first)" )
`
    -apps @( "path/url to the new version of the .app files" ) `
    -countries @( "countries you want to validate against (f.ex. us,ca)" ) `
    -affixes @( "affixes you own (f.ex. fab,con)" ) `
    -supportedCountries @( "supported countries (f.eks. us,ca)" )
$validationResults | Write-Host -ForegroundColor Red
```

All array parameters can also be specified as a comma-separated string. For more information, you can also check this blog post Run-AlValidation and Run-AlCops.

Please include app and all library apps in both previousApps and apps and please include all countries on which you want to validate.

> **NOTE**
> The Run-AlValidation cannot see whether the affixes to specify have been correctly registered with Microsoft using your MPN ID and app publisher name, please make sure registration is in place.

IMPORTANT

The Computer on which you run this command must have Docker and the latest BcContainerHelper PowerShell module installed and be able to run Business Central on Docker.

If you are having issues with Business Central on Docker, you might be able to find help here: https://freddysblog.com/2020/10/12/troubleshooting-business-central-on-docker.

You can use https://aka.ms/getbc?artifacturl=bcartifacts%2fsandbox%2f%2fus%2flatest to create an Azure VM, which has all prerequisites installed to run Business Central on Docker.

NOTE

Microsoft recommends that all partners set up DevOps processes to ensure that this validation process happend automatically and regularly.

You can find resources for how to set up a build pipeline, which performs all these steps here: https://aka.ms/cicdhol and you can find sample repositories, performing these steps here:

- https://dev.azure.com/businesscentralapps/HelloWorld.AppSource (for Azure DevOps)
- https://github.com/BusinessCentralApps/HelloWorld.AppSource (for GitHub Actions)

Manual validation performed by the Business Central validation team

The primary responsibility of the manual validation is to ensure that the apps are working as described.

Manual validation is not done on all submissions. They will be done as sample tests.

For manual validation, we spin up a container with the right artifacts (same as used during technical validation) and the necessary apps are installed. Rapidstart packages needed for the manual test are installed.

The manual test validation document is run manually and if the document doesn't match the app functionality the submission is rejected.

IMPORTANT

Microsoft recommends that all partners are performing the manual validation as the last check before submitting for validation.

This can be done either in online sandbox environments or in sandbox docker containers.

See Also

Developing AL Language extensions

How to Make Compelling Videos

2/17/2021 • 9 minutes to read • Edit Online

Why use video? It is well worth investing time and resources to create marketing videos for your app, it is taken seriously in a business environment.

Reasons why video is a superior medium

- Videos offers a very rich, stimulating communication medium that engages multiple senses.
- Video engages the mind and triggers emotions, which makes it more compelling than text-based content.
- Our brains have an easier time processing visual stories than bullet points or straight facts.

A recent Demand Gen survey indicated that 58% of B2B buyers consume video content, while Hyperfine media states that 59% of executives would rather watch video than read text. Also, 50% of executives look for more information after seeing a product/service in a video.

Speak to Specific Personas in your videos

You should create a video for each of the three core personas in the company:

- WHY persona: Owner/executive/leadership
- HOW: Business line manager
- WHAT: IT buyer, User

A horizontal generic message that attempts to speak to everyone will likely not reach anyone in an emotionally engaging way. Wasting a prospect's time by requiring him/her to listen to irrelevant data or information will only create frustration and lead him/her to form a negative bias towards your company.

Choose the video format that is relevant for the audience that you want to target

Video type 1: "Why" video

How to set up "Why" videos

- Recommended length: 60-90 seconds
- Purpose:
  - Your video should clearly communicate WHY prospects need to buy your solution now.
- Focus:
  - Make sure the prospect is the hero of the story, not you or your company. Prospects are not interested in hearing about your company at this stage. They are simply trying to determine if what you offer is of value to THEM.
  - Your video should speak to the principal challenges and goals of your core decision-maker persona.
  - Describe the desired end state they will achieve by using your app.
  - A client/customer speaking about the benefits they received from your app is far more credible and compelling than anyone from your organization.
    - Don't only rely on "features" to acquire new customers.

How to speak to a WHY persona in a video

- Target audience:
  - Owner/executive/leadership
  - They have limited time and financial resources as well as many competing priorities and resource requirements
  - You need to elevate the discussion to a strategic level, where you highlight market share, competitiveness, profitability, differentiation, revenue loss, and more.
- Message:
  - The question you must answer beyond a doubt is WHY should they invest the time and money to buy your app? What will they get out of it?
  - Why should they spend money on a new system now? Can't they put it off?
  - The WHY messaging teaches people something and it is industry specific and results oriented, as well as being memorable. It engages the emotional/limbic brain and leads to meaningful action.

Video type 2: "How & What" video

How to set up "How and What product videos"

- Recommended length: Up to 3 minutes.
- Purpose:
  - This video goes into greater depth communicating the main benefits of your app as well as HOW you solve your prospects' problems. You can include some WHAT content. • Focus:
  - Demonstrating the proof of your claims is critical during this video.
  - Show very specific dashboards or visually show how you address prospect challenges.
  - If possible, use contrast to create desire and a sense of urgency. For example, you could show a complex, ugly data-filled forecast spreadsheet next to a beautiful visual dashboard stating "your sales forecast before and after."

How to speak to a HOW persona: (Business line manager)

- Target audience: Business Line Manager
  - HOW focuses on the operational benefits your solution will provide and HOW your organization will support the implementation.
  - Speaking to the HOW persona starts to separate you from the pack. • Message: o HOW content is VISUAL in nature and ACTION oriented. It allows your prospects to identify with you at a FUNCTIONAL business level and to Add-on with you. It provides evidence that your organization has relevant industry experience. Tribal acceptance increases, while risk decreases.
  - HOW messaging begins to appeal to the limbic brain because it is focused primarily on emotional business pains and problems.

How to speak to a WHAT persona: (IT buyer, User)

- Target audience: IT-buyer, User
  - WHAT people are often tasked with finding a solution and are important influencers in the decision, but they are not the financial decision makers, and their opinions are easily overturned by HOW and WHY people in the organization.
  - Therefore, don't invest all of your marketing time, money, and effort into providing content just for them.
- Message:
  - You need to survive the WHAT inquisition and provide information about product-related features, functionality, and data so that prospects clearly understand your solution offering.
  - However, this will seldom trigger an emotional response and, therefore, it is likely there will be little or

no emotional engagement with your content.

- WHAT content is binary. WHAT content is a commodity. WHAT content is boring. Logical WHAT content is a necessary evil because many prospects initially go looking for it, but stopping here means remaining relevant only to WHAT personas.

Video type 3: "Getting started" video

How to set up "Getting started videos"

- Recommended length: 2–3 minutes maximum.
- Purpose: This video should prove it is quick and easy to get up and running with your app.
- Target: What personas (Users, It buyers)

Video type 4: "Customer testimony" video

How to set up "Customer testimonial videos" - Recommended length: Up to 2 minutes

- Purpose:
  - Social reinforcement: Customer stories are the best proof of gain.
- Focus:
  - A story coming directly from your client in the form of a testimonial is stronger than having your prospects take your word for it. If prospects see that other similar people orcompanies have already purchased your solution, then their natural response will be to more readily accept it as a solution for themselves.

Video tips

How to structure your video and practical things to keep in mind when producing videos

How to structure the flow in your video?

- Gain immediate attention in the first 10 seconds of the video Stimulate curiosity by include a hook phrase/comment that will elude to solving a pain point. Ask questions about the prospects' core business challenges or ask about something they would like to do but can't accomplish today.
- Highlight the prospects' problems: Use an empathetic approach when describing their current situation and demonstrate that you understand their current business challenges. They must relate to this if they are to continue watching.
- Give them new learning Teach them something they don't know. Demonstrate you have expertise and knowledge about their business or industry that they might not. Show you can offer strategic value to them.
- Paint a picture of a desired outcome they would love to have or state they crave to experience Highlight the benefits, rewards, and value they will enjoy after they purchase from you. Include both what it looks like and how it will feel.
- Prove what you're saying is true Prospects don't trust us when we say our products are great. Include objective and credible proof in the form of data, charts, graphs, quotes, statistics, or testimonials as evidence of your claims.
- Ask them to take action Include a call to action at the end of all videos. When viewers watch your videos, they should feel inspired to take the next step towards purchasing. Tell them what to do next and include an interactive link to the next step in the buying cycle. Use scarcity to compel them to action. Provide a timelimited offer or, for example, say it is "only for the first 20 customers".

Practical things to keep in mind when producing and distributing your video

Does and don't when producing your video

- Don’t make the video too long As our attention span is 8 seconds the ideal length of video is 90 seconds (minimum 30 seconds/maximum 2 minutes).
- Add interactivity where possible Overlay text, charts, animation, questions etc. Visually call out key messages.
- Make sure your audio is high quality.
- Make your video easily shareable
- Enable your video to be shared on multiple media. Track views and attention span. Observe and measure viewer patterns so that you can learn from prospects’ actual behaviors and then improve future content.

How to make a good narrative that speak to the right persona in the right way?

- Your narrative should have a beginning, middle, and end.
  - Lead with a story, not with your app or the technology.
  - Don’t turn your videos into a product pitch.
  - You’ll build more brand affinity and trust by shedding light on a problem your prospects care about rather than by pitching your solutions to them directly.
- The brain is on alert at the beginning of the video and at the end.
  - Make sure the first and last ten seconds are compelling, memorable, and interesting.
- Speak directly to a particular persona in the second person.
  - Do not talk about them in the third person, and avoid using terms like “our clients” and “companies”; instead, use “you” language as often as possible.
  - Use a lot of industry specific vocabulary, terminology, and visuals. If possible, film onsite at a customer’s location rather than in your office or in a studio.
- Speak to a particular persona:
  - Do not try to appeal to everyone at once, as you may not fully engage anyone with this approach.
  - Keep your delivery casual and authentic to instill trust. Speak directly to the prospect as if you were having a fireside chat
  - The prospect should be the hero of the story, i.e. do not speak about you and your company.
- Ask rhetorical questions that stimulate pain and anxiety in your prospects in order to demonstrate that you understand their business problems.
  - For example: Are your margins decreasing? Having cash flow problems because you can’t collect payments sooner than 90 days? Had another large write off? Lost an important customer recently due to a late delivery?
- Use visual and auditory language to help the prospect imagine a new possible future.
  - For example: “imagine seeing” , “picture yourself”, or “ how would you like to hear your clients say…” and so on.
- Use contrast whenever possible.
  - Compare prospects’ experience now versus what it could be after the implementation of your solution.
  - Call out your competitive differentiators while anchoring your solution in prospects’ minds so that they can compare all others against the bar you set.

How to make a good narrative that speak to the right persona in the right way?

- Where possible, use tangible, concrete language.
  - Include quantifiable proof in the form of data or visual pictures.
  - No vague claims like “transform your business with the cloud”. This is an emotionless statement.
- Providing customer references and testimonials is much more compelling and effective than selling your company or product yourself.
  - Let others speak for you. A customer testimonial video will always be more believable and compelling than a video of you saying the same thing.

- Surprise and delight them.
    - Use humor to make them smile. We take ourselves and our problems too seriously. Be warm, memorable, and unique.

Guideline on Creating an Effective Sales Landing Page for Your App

2/17/2021 • 10 minutes to read • Edit Online

Building a landing page that drives a successful buying transaction

Microsoft will drive qualified traffic to AppSource. Though, once a prospect becomes aware of your app, it will be your job to guide them through to a successful buying transaction. Deliberately mapping and architecting the buying journey is critical to ensure a high level of engagement and conversion. Only presenting your app's features and functionality, or just providing a free trial, will not ensure prospects will become buyers. For this you need to have a good landing page that is built to help you capture attention, accelerate your customer acquisition process, and drive buying behavior. The recommendations on this page will help you do so.

Examples of how other partners have implemented our best practices

To inspire you in creating a good landing page for your app, two of our valued partners, LS Retail and Industry Built, have offered to provide a sample of what a best practice landing page for a Microsoft Dynamics 365 Business Central partner could look like.

Have a look at their app landing pages and use them as inspiration to build your own landing page:

- Industry Built's Build Food app
- LS Retail's LS Express Start app

In the following checklist, we have "broken down" the elements, on their landing pages in order to showcase best practices on design and messaging. More specifically, we are looking into layout and structure elements, content elements, visual elements, anxiety reducing elements and support elements.

Additionally, we have provided specific recommendations on how to apply these elements to help you increase conversion and maximize the effectiveness of your product's sales landing page.

We urge you to review and implement these best practices on your landing page – in so doing you will contribute in providing the Microsoft community of customers with a consistent buying experience across publishers.

Layout and structure elements

ELEMENT	DESCRIPTION	EXAMPLE
Company	Include the company logo on the page	
App name & app logo	Include a visual logo of your product name and a one sentence positioning statement.	
Top menu choices	Use clean, straightforward and descriptive menu options.	

ELEMENT	DESCRIPTION	EXAMPLE
Search box	Include a search box so visitors can quickly find what they are looking for.	
Emotional tribal anchor photos	Visuals create an emotional Add-onion. The brain skims over non-emotional photos.	
Visual	Make your page easy to scan, with lots of strong visual imagery.	

Logo

- The upper-left corner of the landing page is the most valuable section of the entire landing page.
- Place your company logo in this location.

If you need help formulating a positioning statement, try the value proposition generator located at here.

- There should ideally be 5 or fewer choices; do not include more than 7 options.
- The menu text should state what the prospect gains if they click on the menu item
- The text should be written from their perspective, not yours.

Recommended menu items:

- How to Buy, Benefits Gained, Why Us, and Contact.

The upper-right corner of the page is usually an ideal spot

- Faces evoke more emotion than landscapes or machines, and so on.
- Include a happy customer that looks similar to your prospect in terms of age, demographic, and industry, and which shows them dealing with the issues that your prospect can relate to.
- Try not to use stock photos of people or objects.

Engagement

- Too much text forces the brain to skim, skip, and exit. Text engages the logical, analytical brain, but not the emotional brain.
- Keep it clean and straightforward in terms of design and layout. Use lots of pictures, graphs, and screen shots to enhance engagement.

Content elements: Text and messaging

ELEMENT	DESCRIPTION	EXAMPLE	
Include a headline question	Get your prospects' attention by asking them a compelling pain-based question that they can relate to.	"Struggling to manage your ingredient inventory and fretting over allergens?"	

ELEMENT	DESCRIPTION	EXAMPLE	
	You want the prospect to mentally say "YES" as often as possible and to peak their curiosity enough to read more.		
	Your questions should be intriguing and customer-centric.		
	In general, 8 out of 10 people will read headline copy, but only 2 out of 10 will read the rest.		
Microsoft Dynamics 365 product description	Somewhere on the landing page, make sure you include the standard Microsoft Dynamics 365 Business Central product description provided by Microsoft *This is a requirement because your product is adding value to and building on this foundational solution.*	*Insert this paragraph:* Microsoft Dynamics 365 Business Central is a comprehensive business management solution for small and medium-size businesses (SMBs) that have outgrown their basic accounting software. From day one, this new application makes ordering, selling, invoicing, and reporting easier and faster. Dynamics 365 Business Central is deeply integrated with Microsoft 365 and includes built-in intelligence, so it is easy to use and helps users make better business decisions.	
Messaging (Address their pains)	Pain is a strong motivator of action.		
	- Identify 1-3 key sources of the client's most prominent pain early on the page.		
	- Call out the fears that are likely to be holding them back.		
	- Your landing page text and messaging should predominantly focus on the pain the prospect is experiencing, and NOT the features of your product or service.		

ELEMENT	DESCRIPTION	EXAMPLE	
Clearly demonstrate to your prospects that you genuinely understand their industry and unique business problems.	- Describe the business challenges they are facing now and the ways their revenue growth, margins, productivity and so on, are being negatively impacted by not taking action now.		

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Product benefits)	Paint a clear, visual and desirable picture of what is possible.	
	Describe the most significant benefits and rewards that your prospect will realize after purchase.	- For example, "Save time and money (benefits) by having a system that does all the tracking and calculations for you (features)."
	- Don't only list features and app functionality, start with the benefit first, then you can follow with the features.	
	- Paint a picture of a possible experience the prospect will immediately desire.	
Clearly articulate a compelling desired outcome	- If possible, use industry-specific language and vocabulary to resonate with your prospect deeply.	
	- Choose a particular persona to speak to directly.	
	- Engage prospects by speaking directly to them using first person "you" language.	

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Prove your claims)	Include specific calls-to-action on your app page.	"Reduce how long it takes to set up your recipes in the morning from 1 hour to 10 minutes."
	Don't make general and abstract claims.	

- Use data as often as possible to support your statements.

If you make specific claims, support your claims with proof, while Quantifying impacts and gains.

- The more specific and concrete your promise of value is, the better.
- Abstract concepts such as "more efficiency, more productivity, transform your business" are not emotionally

impactful or convincing and do not compel a prospect to act.

Target market - If you support multiple countries or languages, this is a key selling feature. • Find a way to show this visually.

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Compelling call-to-action)	Include specific calls-to-action on your app page	

- This can be your free trial; a time-limited special price; a scheduled walk-through demonstration; and so on.
- The words "free" and "save" are highly emotional words in the English language, so they should be used.
- Use bright colors, such as orange, yellow, or red, to call attention to your buttons.

Button text should use benefit language rather than descriptive language.

- For example, instead of "Download" write "Click here to start saving money now."
- Try not to send prospects away from your page – always have an embedded next step in your call to action that brings them back to your landing page.

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Create a sense of urgency by teaching the prospects)	Help your prospect gain a sense of urgency to buy by teaching them one thing about how they can be more efficient or profitable now.	Your bakery profitability will decrease over the next five years due to an increase of 3% in the cost of key inputs, such as wheat and sugar. Want to know five key strategies that can help you mitigate this challenge? Click here to find out how to preserve your profit margin

- Show them how their performance in one key business area is below that of their competitors.
- For an example you can provide a quick online self-assessment, a top-10 tips blog post, and much more. Visual elements

ELEMENT	DESCRIPTION	EXAMPLE	
Pictures (Differentiation comparison images)	Show them, don't tell them	Show the before and after state.	

- This is a visual image of how your prospects do things now versus how they will be able to do it in the future.
- You are not telling them but showing them using a visual.

ELEMENT	DESCRIPTION	EXAMPLE
Compelling proof screen shots	Visually demonstrate all the claims that you are making.	Quickly and easily view inventory items

- Graphic dashboards are the most effective method.
- Zoom in on the main benefit-related features.
- Make sure it is readable, and the benefit is obvious.
- Include a caption.
- Data should be industry specific so that it resonates with the viewer.

You want prospects to see how their data/process would look in your system.

ELEMENT	DESCRIPTION	EXAMPLE
Videos (Tell your story using videos not text)	Include as many videos as possible.	

- Videos have a much higher level of engagement and viewing time and convey much more than you can ever say with words.

Include at least one customer testimonial video on your app landing page.

- Your client should speak specifically about the pains they had before and the benefits they gained after, not product features. It should be all about your customers, not you.

Include one product demonstration video.

- See the video best practices https://aka.ms/ReadyToGo.

Elements that reduce anxiety and risk, while increasing trust

ELEMENT	DESCRIPTION	EXAMPLE
Customer testimonials	Don't sell your product; let your customers do that for you.	

- Social proof is more credible and trustworthy to prospects. The purpose of testimonials is to reduce the buyer's anxiety and fear.

Your testimonials should answer the following questions:

- "Will this work for my situation?"
- "What benefit will I really get if I buy this?"
- "Is this going to be too hard?"
- "How long is this going to take?"
- "Can I trust this company?"

ELEMENT	DESCRIPTION	EXAMPLE
Reduce risk	Prospects are afraid of being scammed and taken advantage of on the internet. They are naturally cautious and highly suspect.	Source: Microsoft.com

- You want to convert prospects to buyers.
- Make it easy for them to buy, while reducing their anxiety. Transparency is the key to building trust.
- Make sure that you include a link to a BUY NOW page, which includes full pricing details.
- Give them a compelling offer they cannot refuse. Offer a time-limited trial or special pricing discount if they buy in 30 days.
- Use scarcity to compel action. Offer a 100% money-back guarantee.

We recommend providing three offerings, optimized for three different customer segments. For more recommendations on pricing, see the pricing guide located at https://mbspartner.microsoft.com/BFI/Topic/64

ELEMENT	DESCRIPTION	EXAMPLE
Live chat	Include live chat, with a photo of one of your team members smiling at an appropriate time to increase conversion, such as when a prospect clicks the back button on your pricing page.	

- Include their name if possible to build trust.

Support elements: Interactivity and contact options

ELEMENT	DESCRIPTION	EXAMPLE
SHORT lead capture form	Include a lead capture form on your page.	

- Only ask for their name and email address, you can get the rest later.
- Your forms should not have more than 4 or 5 fields to fill out. You have not yet earned the right or enough trust to ask for too much information at this point.

Most lead capture forms are way too long, demanding, and intimidating, and have low completion rates.

> **NOTE**
> Nobody has the time or is willing to fill out an annoying form, which is of no value to them, especially if it is purely self-serving from your standpoint.

ELEMENT	DESCRIPTION	EXAMPLE
Contact	Provide prospects with different contact options based on their readiness to interact with you.	

- Ideally, include a phone number and an email address with an employee photo.
- This alone could double your conversion rate.

ELEMENT	DESCRIPTION	EXAMPLE
AppSource app page link & social share	Include a link back to your listing on AppSource, so the prospect can return when ready.	Return to AppSource

- Also, enable visitors to share and forward your app with others!

ELEMENT	DESCRIPTION	EXAMPLE
Close them! Add a get started button	Include a very specific call-to-action button with the option to buy or try.	

Embed App Overview

2/17/2021 • 4 minutes to read • Edit Online

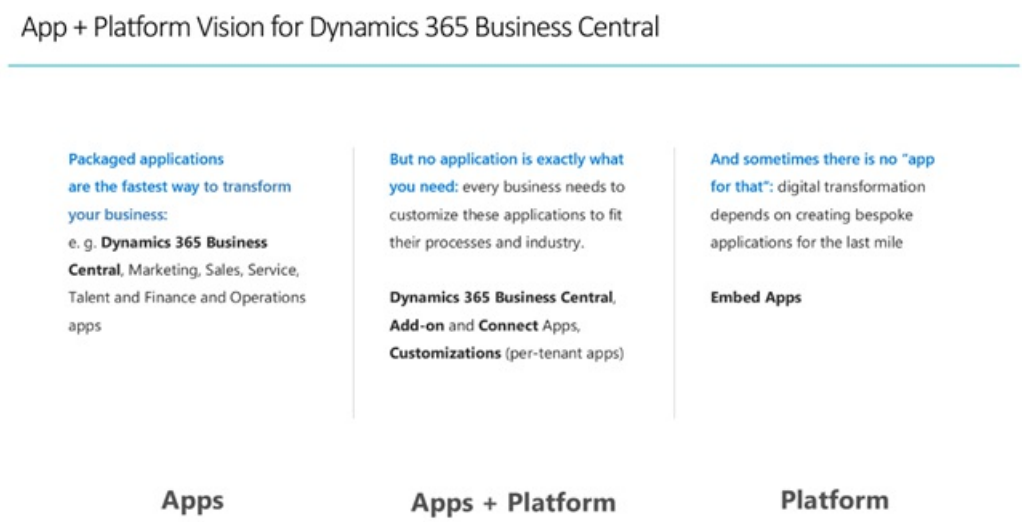
Embed App is a term that defines an end-to-end solution meeting the specific needs of a vertical or micro-vertical industry.

Dynamics 365 Business Central plays a vital role in the Embed App, as Business Central is embedded as an integral part of the overall solution.

Some examples of an Embed App include:

- A Dentist solution
- A Real Estate Agent solution
- A Food Processing solution

An Embed App refers to what is being provided to a given customer segment, unrelated to how the solution is being implemented or architected. An Embed App can be built using AL, in other words extension, code-customization, and a combination of extensions and code-customization.



Components

On a high level, an Embed App is a package that consists of the following parts:

- Library extensions

  This is the functionality of the Embed App that is implemented by the ISV partner in a form of extensions.

- Third party extensions

  These are add-on extensions coming from other ISVs that contribute to and enhance the Embed App. The extensions are validated to be compatible by the Embed App owner.

- Extended metadata

  This includes additional Embed App properties that are specific to this type of app and not otherwise available for other types of apps (see the list below).

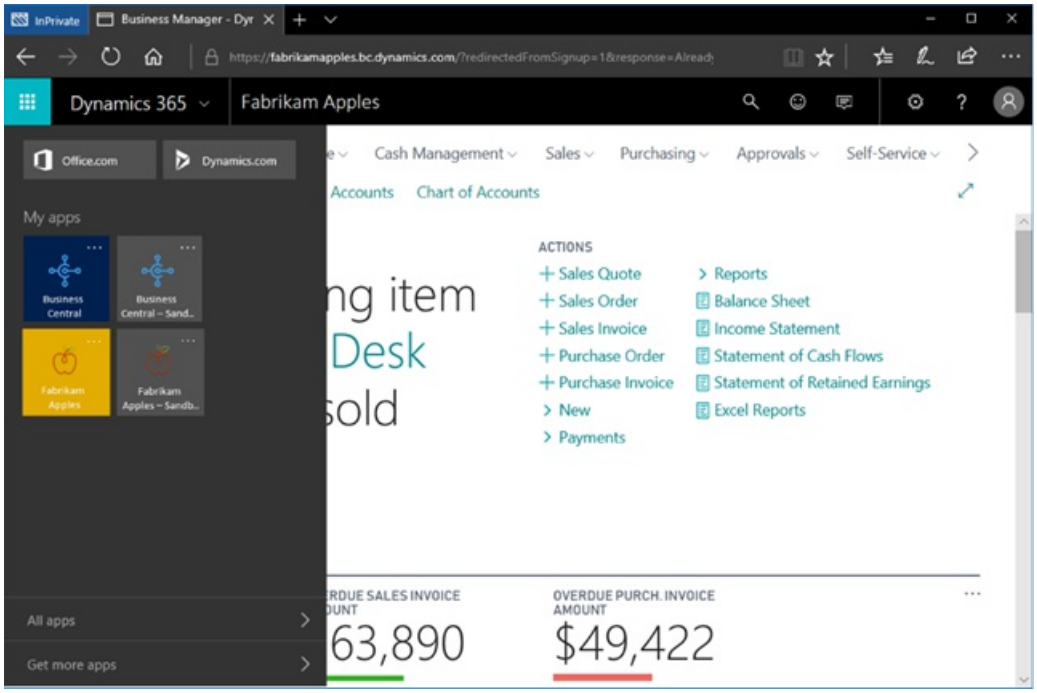
- Base application and tenant template (optional)

The following capabilities are only available for the Embed App and not for other types of Business Central apps (Connect and Add-on).

Partner Branding

The Embed App will promote the partner's brand in several places:

- Web Client and Web Service URLs
  - Client: `https://[application name].bc.dynamics.com`
  - Web Services: `https://[application name].api.bc.dynamics.com`
- Name, image, and icon on the provisioning page of the Fixed Client Endpoint
- Splash screen of the client
- Title bar of the browser tab (for example, "Fabrikam Apples")
- A dedicated product tile, icon, and short marketing description in the Dynamics shell (https://home.dynamics.com)
- In-product messages (such as pop-up errors, warnings, notifications)



Exclusivity

The partner can control which third party apps can be installed for their Embed App.

- Safe listing of the 3rd party apps - no other apps will be possible to install, except the ones explicitly approved by the partner
- App install/uninstall controlled by the partner
- The partner can choose to allow a customer to install other extensions from the AppSource, but this will be an explicit partner decision, not the default behavior

Code-customizations of the base application

Partners can choose to bring their own code-customized base application as an Embed App for several reasons:

- Shortening time-to-market ("lift and shift" approach).

  The partner's current solution is a significantly customized version of the Dynamics NAV application and it will require substantial time and effort to migrate it into extensions. A partner can lift their solution as-is (upgraded to a supported platform) to Business Central service and start offering it to their new and prospective customers. Then, they can gradually start moving their functionality into extensions to achieve the benefits that come with the extension model. The earliest version of the Business Central

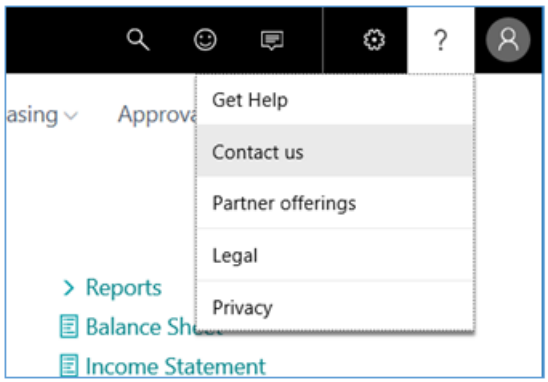
application you can bring to the service is version 16.

- Usage of .NET interoperability and custom assemblies.

  Partners that use .NET interoperability in their current application to address multiple business scenarios. Although extensions today allow a number of these scenarios to be implemented in AL, they don't and cannot cover for all possible scenarios of .NET usage. Therefore, the partner can choose to import the required .NET add-ins into the Add-ins table of the base application, and these add-ins will automatically be deployed into the environment where they will be running.

Additional settings (metadata)

- An Embed App is the property of the partner, so the customers of the Embed App must be able to find the partner's own legal, privacy, contact, community and feedback links (not Microsoft links) when they work with the app:



- Safe listed domains for embedding Embed App pages into other web sites, including SharePoint ("frame ancestors")
- Target version of Dynamics 365 Business Central platform
- Target version of Dynamics 365 Business Central base application (if not included with the Embed App)
- Azure KeyVault account for storing application secrets, such as accounts for connecting to 1-3rd party services
- Base application + tenant template. This is an optional component of an Embed App. The partner can choose to include it or simply specify which version of the Dynamics 365 Business Central base application the Embed App should use as a base application.

At this stage, within the extensions and base application, the partner can work in their own Object ID range.

Platform version availability

Microsoft is going to make new versions of the Business Central platform available to Embed App partners through the Lifecycle Services portal (LCS). The partner will then have to pick the platform they want to use for deployment of their solution.

Deploying versions

When a partner deploys a solution through the LCS portal, they can pick from the last three available versions of the platform (minor and major). Every newly released minor or major platform update will be added to the list and simultaneously one older version will be removed form that list.

Any existing deployments, running on platform versions that are older than 3 updates, will enter a grace period of 30 days. And after that, if the deployment is not upgraded, it will be moved out of the standard SLA.

See Also

Microsoft Responsibilities

Qualification and Onboarding of Partners to the Embed App Program

2/17/2021 • 2 minutes to read • Edit Online

If you as a software development partner are interested in developing an app that meets the description of Embed App in Embed App Overview, your app must meet qualification requirements. The main criteria for Embed App onboarding at this point are:

- Partner must provide all day, every day support. Their customers do not get to call Microsoft support. They always call the partner.
- Partner must live up to the standards for user assistance and documentation that are outlined in Dynamics 365 Business Central User Assistance Model. Good user assistance is part of Embed App success and uptake, and it also lowers the work load on their support center.
- Partner is committed to stay on the supported platform.
- Partner provides an SLA for requests from Microsoft towards the partner
- Partners must have a support agreement with Microsoft

Apps with a code-customized base application will have additional volume (number of users) requirements associated with it.

When an Embed App is qualified for onboarding to Business Central online, the partner must provide the following information about the Embed App to the Microsoft's onboarding team:

- The name of the application to be used for the client and web service URL. For example, an ISV for the *fabrikamapples* app would provide the following information:
  - Client: `https://fabrikamapples.bc.dynamics.com`
  - Web Services: `https://fabrikamapples.api.bc.dynamics.com`
- Entitlements for their application objects, per license type. For more information, see Licensing in Dynamics 365 Business Central.

The rest of the information will be included with the Embed App package that the partner deploys through LifeCycle Services.

See Also

Embed App Overview
Microsoft Responsibilities

Embed Apps Deployment

2/17/2021 • 2 minutes to read • Edit Online

This article provides an overview of the process for deploying an Embed App to the Business Central Online service. The deployment involves the following tasks:

TASK	DESCRIPTION	SEE...
Create a deployment package	The deployment package is a .zip file that contains components required for any Embed app, including: • ISV branding elements, like images, icons, names, and so on • An application database and tenant template databases • A manifest-json file that provides additional metadata and deployment instructions.	Creating Deployment Packages for Embed Apps
Upload your apps to the App Repository	The apps are the individual extensions that make up your application. Apps are delivered as .app files. You upload your apps into the App Repository by using the App Management API.	App Management API
Upload the deployment package to Microsoft Lifecycle Services (LCS)	Once you have the deployment package, and your apps have been uploaded, you can deploy your package to an LCS project.	Managing Embed Apps in Microsoft Lifecycle Services

See Also

Embed App Overview
Qualification and Onboarding
Managing Embed Apps in Microsoft Lifecycle Services

Creating Deployment Packages for Embed Apps

2/17/2021 • 13 minutes to read • Edit Online

To deploy an Embed App, you'll need a deployment package and your apps. This article describes how to create a deployment package for an Embed App.

The apps are the individual extensions that make up your application. Apps are delivered as .app files, which you upload to the App Repository via the App Management API. For information about this task, see App Management API.

Once you've created the deployment package and uploaded your apps, you then upload the deployment package to a registered project in Microsoft Lifecycle Services.

Deployment package overview

Creating a deployment package prepares the Embed app code and data for deployment on Business Central Online service. The deployment package is a .zip file that contains components required for any Embed app.

At a high level, a deployment package consists of the following components:

- ISV branding elements, like images, icons, names, and so on
- An application database and tenant template databases as BACPAC files
- A manifest-json file that provides additional metadata and deployment instructions

The deployment package has the following structure:

```
[FOLDER] branding/
  - favicon.ico
  - splash.png (or .gif)
  - splash_narrow.png (or .gif)
  - header.png (or .gif)

[FOLDER] databases/
  - app.bacpac
  - tenant.bacpac

manifest.json
```

The following sections explain how to build the deployment package.

Creating the database BACPACs using Docker

We strongly recommend you run Business Central on Docker using the BCContainerHelperPowerShell module to prepare the BACPAC files.

> **IMPORTANT**
> You must only use the sandbox artifacts (Get-BcArtifactUrl -type sandbox) for your online deployments.

With the Docker container, you can:

- Attach your own database and publish necessary extensions into the application database
- Synchronize the schema of the tenant template database with the schema defined in the application

- Ensure that all objects are compiled
- Test your solution before creating the deployment package.

For more information about these and other things you can do, see BCContainerHelperPowerShell module.

About the databases

Business Central Online service runs exclusively in multi-tenant mode. Each customer environment is a separate database, but sharing the same application. So the deployment package has to include two databases, which are compiled into BACPAC format:

- App.bacpac - includes the application database. The application database contains your application data, like permissions, entitlements, report layouts, and so on.
- Tenant.bacpac - includes the tenant template database, which may also contain initial tenant data.

Requirements for the application database

REQUIREMENT	CONDITION	DESCRIPTION
Running on a supported Business Central platform	Always	The Business Central platform is updated monthly. You're responsible for updating the Embed App to operate with the updated version of Business Central. The Embed App must run on a supported build (platform) of Business Central, namely, the current version of Business Central or one of the two immediately preceding versions. The immediately preceding versions can be both minor and major versions of Business Central. Earlier versions of Business Central platform are out of support and can't be used for Embed app deployments.
Unique Base Application ID, Name, and Publisher	If any change is applied to the Microsoft Base Application objects	Create and assign your own unique App ID for it in the app.json file. Also, change the Publisher property and the Application name to your own in the app.json file. You can't use the original values of the Microsoft Base Application unless you've extracted all your customizations from that app into your own app(s).
Use System Application	Always	Your Embed app must be built on top of the Microsoft System Application. The System Application must not be customized and must be used as a dependency for your customized or non-customized Base Application.

REQUIREMENT	CONDITION	DESCRIPTION
Use `Microsoft_Application.app` file	If you use customized Base Application	Use `Microsoft_Application.app` to encapsulate references to your own customized Base Application, instead of referencing that app as a direct dependency in each extension separately. This will also allow other apps (like AppSource apps) to refer to your solution via the `Microsoft_Application.app`. For more information about the `Microsoft_Application.app`, see The Microsoft_Application.app File.
Solution prefix or suffix	Always	Register a prefix or suffix to be used by your solution. Apply the prefix or suffix to all objects, while you're still preparing your apps on-premise. The prefix or suffix must be used for all your app objects and also for your tables, controls, fields, actions, and groups that remain inside of Microsoft's Base Application. There's no need to apply this prefix or suffix to Microsoft objects and fields in the customized Base Application. Fore more information about requesting and using prefixes and suffixes, see Benefits and Guidelines for using a Prefix or Suffix. Once your apps are deployed to Business Central service, it won't be possible to rename fields and objects, because it would be a breaking change. Breaking changes aren't permitted in the online version of Business Central. While still preparing your apps on-prem, you can use the Sync-NAVApp cmdlet with `-Force` parameter to add prefix or suffix to your objects and fields.

REQUIREMENT	CONDITION	DESCRIPTION
Entitlements	Always	It's important that the application database also contains Microsoft unmodified records in the following tables: • Membership Entitlement • Entitlement Set • Entitlement No users can sign in to your service if these tables are empty. You can't add new records or modify existing records in these tables. For more information about these tables, see Business Central entitlements explained. The entitlements can be copied from the standard Business Central Demo Database available in sandbox Docker containers. Make sure you update the entitlements when you uptake a new version of Business Central application. Consult the entitlements documentation and parameters provided for the `Clean-BCContainerDatabase` and `New-BCContainer` commands included with the BcContainerHelperPowerShell module.
Permissions	Always	The application database must contain Microsoft unmodified records in the following tables: • Permission Set • Permission You can't modify the existing records in these tables. If you need to add your own permission sets and permissions for the objects used by your own solution, include these with your extension instead. For more information, see Exporting Permission Sets. Don't include customer-specific permission sets and permissions in your database, these permissions must be added within the customer environment instead. For more information, see how to create or modify a permission set in the Business Central Application help.

REQUIREMENT	CONDITION	DESCRIPTION
Custom .NET add-ins	If your solution uses custom .NET Server components (.NET add-ins)	Import components into the **Add-in** table in the application database. You register the control add-ins, import and manage the files by using the New-NAVAddin cmdlet, Set-NAVAddin cmdlet, Get-NAVAddin, or Remove-NAVAddin cmdlets from the Business Central Administration Shell. **IMPORTANT** Using custom .NET Server components option is being deprecated. So refactor your code to remove custom .NET components as soon as you can. For example, use Azure Functions instead.

Requirements to the tenant template database

While you are testing your Embed app solution in a Docker container, you can publish and install all the apps you need for your customers. When you're done testing, we recommend you to uninstall (including removing the data) and unpublish all apps, including Microsoft apps, before you export the application and tenant template databases.

Instead of keeping the apps pre-published and pre-installed in your databases, they must be uploaded to the service by using the App Management API and listed in the `[apps]` section of the `manifest.json` file. This way the apps will be automatically published and installed later, during deployment, following the list you provide in the `manifest.json` file that you include with your deployment package.

We recommend you to use RapidStart packages to populate the new companies with the demo data or initial setup data.

Exporting application and tenant databases

You export the databases to BACPACformat by usingthe `Export-BCContainerDatabasesAsBacpac` command from the BcContainerHelperPowerShell module.

We strongly recommend using this command for creating the BACPAC files you're planning to deploy to the Business Central service. To simplify exporting the data, the command also does a number of clean-up stepson the databases. It cleans up sessions,database connection, list of tenants, imported license, and more. It also verifies that the schemas of the application and tenant databases are synchronized, which is essential for deployment.We keep enhancing this commandwith more cleanup and validation steps as we discover BACPAC-related issues with deployments. So, remember to update the BcContainerHelperPowerShell module for every new iteration.

Providing deployment instructions in the manifest.json

The manifest.json file, supplied within the deployment package, contains important properties used by the deployment routine to configure the application version according to the Embed app ISV requirements.

SETTING	TYPE	DESCRIPTION
manifestSchemaVersion	String, `<major>.<minor>`	Schema version of the manifest. Current manifest schema version is 3.0.

SETTING	TYPE	DESCRIPTION
id	GUID	Unique ID of the Embed solution. Used for informational purposes, not used in the runtime. As a best practice it's recommended to use the Id of your main app in this field.
name	String	Short marketing name of the app. Used for informational purposes, not used in the runtime.
description	String	Short marketing description of the app. Used for informational purposes, not used in the runtime.
publisher	String	Name of the ISV organization that owns the Embed app. Used for informational purposes, not used in the runtime.
version	String. `<major>.<minor>.<build>.<revision>`	Version of the deployment package being uploaded. This version is displayed in the Application Version list in LCS, it has no other purpose or application in the runtime. Remember to update this version for every new deployment, so you can distinguish the deployments in the LCS UI.
branding	Object	Object that contains paths to the branding elements of the ISV included with the deployment package. See the `"branding"` section below.
databases	Object	Object that contains paths to the application and tenant template BACPAC files of the ISV solution included with the deployment package. See the `"databases"` section below.
links	Object	Object that contains paths to various URLs to be used in the Business Central Web Client. See the `"links"` section below.
apps	Array	Array that contains the list of apps and their dependencies, which are uploaded by the ISV via the App Management API. See the `"apps"` section below.

"branding"

SETTING	TYPE	DESCRIPTION

SETTING	TYPE	DESCRIPTION
productName	String	Official marketing name of your Embed app. This name will be displayed in the title bar of the Business Central Web Client.
productNameShort	String	Short version of Embed app marketing name. It's displayed in the areas of the UI where full Product Name can't be displayed.
marketingName	String	Longer (full) marketing name of your Embed app.
images	Object	Object that contains paths to the branding images included with the package. See the `"images"` section below.

"images"

SETTING	TYPE	DESCRIPTION
appleTouchIconPath	String	Format: PNG. Dimensions: 114x114
faviconPath	String	Format: ICO. Dimensions 16x16, 32x32, 48x48 & 64x64. Used on the Browser tab and bookmark icon.
headerPath	String	Formats: PNG, GIF. Dimensions: 558x107. Displayed on "Minimal Layout" pages, like the Microsoft Office add-in sign-in; Authentication / Permission errors ("You don't have access to Business Central", "You've successfully signed out.")
splashLandscapePath	String	Formats: PNG, GIF. Dimensions: 2048x272. Splash screen on devices with large screen space, for example, full screen browser, tablet in landscape mode.
splashPortraitPath	String	Formats: PNG, GIF. Dimensions: 960x320. Splash screen on devices with limited screen space, for example, small browser, phones, tablets in portrait orientation.

"databases"

SETTING	TYPE	DESCRIPTION
applicationBacpacPath	String	Path to the application database (.bacpac) included with the deployment package.

SETTING	TYPE	DESCRIPTION
tenantTemplateBacpacPath	String	Path to the tenant template database (.bacpac) included with the deployment package.

"links"

SETTING	TYPE	DESCRIPTION
baseHelpUrl	String	Link to your online Help server or web page.
baseHelpSearchUrl	String	Link to your online Help server or web page.
communityUrl	String	Link to your community blog page.
feedbackUrl	String	Link to your product feedback page.
legalUrl	String	Link to your product legal and privacy terms.
signInHelpUrl	String	Link to your sign-in help URL.
comingSoonUrl	String	Link to your product announcements page.
blogUrl	String	Link to your product blog page.
contactSalesUrl	String	Link to your sales page.

"apps"

This section of the manifest.json file must list all the apps used by your solution and their dependencies. Even if the dependency app belongs to a different publisher, for example Microsoft, you must include it. This list will be used by the deployment routine to publish and install these apps to your service and for your environments.

SETTING	TYPE	DESCRIPTION
id	String	Unique ID of the app. Use the same Id as listed in the app.json file.
name	String	Name of the app. Use the same Name as listed in the app.json file.
publisher	String	Name of the ISV organization that owns the app. Use the same Publisher as listed in the app.json file.

SETTING	TYPE	DESCRIPTION
initialVersion	String	Minimum required and compatible version of the app. If `"allowedUpdates"` is set to `"none"`, then the exact version of the app you specify must be available in the App Repository. In this case, a full initial version number must be specified. For other `"allowedUpdates"` values, it's enough that an app with a higher or equal [build].[revision] part of the version is available in the App Repository. You don't need to specify a full version number -- [major].[minor] and [major].[minor].[revision] are also accepted. This applies to the App versions uploaded either by you or by other publishers who own the apps.

SETTING	TYPE	DESCRIPTION
allowedUpdates	Enum	Allowed values: none, hotfix, minor, all. You can set up rules that determine which app version to use for the new signups. You can also specify whether updating of the installed app is allowed. • `"none"`: It won't be possible to apply any new versions to this app inline (only side by side). Only the application version that exactly matches the specified initial version number will be used for the new signups. • `"hotfix"`: You allow hot fixing this app with newer hotfix versions, that is, versions with a higher build and revision numbers than in the `"initialVersion"`, but with the same major and minor version. New signups and environments you upgrade to this deployment will automatically get the latest available hotfix version of the app installed. • `"minor"`: Customers can view and install newer applicable minor (or hotfix) versions of the app via the Business Central Administration Center. New signups and environments that you upgrade to this deployment will automatically get the latest available hotfix version of the app installed. Minor updates won't be automatically installed. They must be installed by the newly signed-up customers using the Business Central Administration Center. • `"all"`: Customers can view and install all newer applicable versions of the app via the Business Central Administration Center. New signups and environments that you upgrade to this deployment will automatically get the latest available hotfix version of the app installed.
blockUninstall	Boolean	Allow or disallow the app to be uninstalled using the Extension Management page.

SETTING	TYPE	DESCRIPTION
publishOnly	Boolean	Specifies whether to only publish (`true`) that app to the service. Customers can then install it manually from the Extension Management page. If `false` , the app is automatically installed for all new environments and environments updated to this deployment.

Sample manifest

Sample of the manifest.json file.

```
{
  "manifestSchemaVersion":"3.0",
  "id": "201e5067-99cc-4974-900a-b50bd4fbe777",
  "name": "Fabrikam Rentals",
  "description": "Harness the power of the most feature-rich office furniture rentals solution from
Fabrikam.",
  "publisher": "Fabrikam",
  "version": "16.1.50043.50321",
  "branding": {
    "productName": "Fabrikam Rentals",
    "productNameShort": "Rentals",
    "marketingName": "Fabrikam Rentals",
    "images": {
      "appleTouchIconPath":  "branding/appletouch.ico",
      "faviconPath":  "branding/favicon.ico",
      "headerPath":  "branding/header.png",
      "splashLandscapePath": "branding/web-wide.png",
      "splashPortraitPath":  "branding/web-narrow.png"
    }
  },
  "databases": {
    "applicationBacpacPath": "databases/app.bacpac",
    "tenantTemplateBacpacPath": "databases/tenant.bacpac"
  },
  "links": {
    "baseHelpUrl": "https://help.fabrikam.com/help/",
    "baseHelpSearchUrl": "https://help.fabrikam.com/help/",
    "communityUrl": "https://fabrikam.com/au/contact",
    "feedbackUrl": "https://fabrikam.com/au/contact",
    "legalUrl": "https://fabrikam.com/au/legal",
    "privacyUrl": "https://fabrikam.com/au/privacy",
    "signInHelpUrl": "https://fabrikam.com/au/contact",
    "comingSoonUrl": "https://go.microsoft.com/fwlink/?linkid=2047422",
    "blogUrl": "https://go.microsoft.com/fwlink/?linkid=2076643",
    "contactSalesUrl": "https://go.microsoft.com/fwlink/?linkid=828707"
  },
  "apps":  [
                {
                    "id":  "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
                    "initialVersion":  "16.1",
                    "name":  "System Application",
                    "publisher":  "Microsoft",
                    "allowedUpdates":"hotfix",
                    "publishOnly":  false,
                    "blockUninstall":  true
                },
                {
                    "id":  "201e5067-99cc-4974-900a-b50bd4fbe777",
                    "initialVersion":  "16.1.5",
                    "name":  "Fabrikam Rentals Add-on",
                    "publisher":  "Fabrikam",
```

```
                    "allowedUpdates":"minor",
                    "publishOnly":  false,
                    "blockUninstall":  true
                },
                {
                    "id":  "201e5067-99cc-4974-900a-b50bd4fbe777",
                    "initialVersion":  "16.1.50043.50321",
                    "name":  "Fabrikam Rentals",
                    "publisher":  "Fabrikam",
                    "allowedUpdates":"minor",
                    "publishOnly":  false,
                    "blockUninstall":  true
                }
            ]
}
```

Creating the deployment package zip file

After you have all the components of the deployment package ready, you can package them into a .zip archive. The deployment package (.zip) can't include any additional subfolders and files and must have the following structure:

```
[FOLDER] branding/
  - favicon.ico
  - splash.png (or .gif)
  - splash_narrow.png (or .gif)
  - header.png (or .gif)

[FOLDER] databases/
  - app.bacpac
  - tenant.bacpac

manifest.json
```

Now, you can deploy your package to Business Central Online service. See Managing Embed Apps in Microsoft Lifecycle Services for more information.

See Also

Embed App Overview
Qualification and Onboarding
Managing Embed Apps in Microsoft Lifecycle Services

Managing an Business Central Embed App in Microsoft Lifecycle Services

2/17/2021 • 7 minutes to read • Edit Online

Microsoft provides essential functionality within Microsoft Lifecycle Services collaboration portal (LCS) to support qualified ISVs in managing the Embed App based on Business Central online.

Creating LCS project

In LCS, you should create a project for each Embed App and each country you would like to deploy to Business Central service. You provide the list of countries where you want to run your Embed app during onboarding. You can only deploy your solution to the country that is already supported in the Business Central online service. Find the list of supported countries here: https://aka.ms/bccountries.

Before you can create a project, you need to unlock a corresponding Private Preview feature. Once you sign in to LCS, select the **Preview feature management** action. Then on the Preview feature management page, select the "+" action to add a new preview feature using a preview code. In the **Preview code** field, enter the code you received from Microsoft during onboarding. You should now see the "Microsoft Dynamics 365 Business Central (SaaS)" feature on the list of the private preview features on this page.

> **IMPORTANT**
>
> Even though the project will be created by one user, every user you are planning to add to your LCS project will have to activate the "Microsoft Dynamics 365 Business Central (SaaS)" feature using the same preview code you received from Microsoft. Without this activation, the users you add to your LCS project will run into permissions issues when trying to open the LCS project that you created.

Navigate back to the main page and start creating a new project by selecting the "+" action. Choose **Migrate, create solutions, and learn** category as the purpose of the project. On the next page, provide the project name.

> **NOTE**
>
> You will have to create a separate project for each country, even if your Embed app is going to be exactly the same. Therefore we recommend that you use the name of your Embed app appended by the country code as the name of your project, for example "Fabrikam Rentals, DK", "Fabrikam Rentals, US" and so on.

Next, provide a short description for your project (optional), select **Microsoft Dynamics NAV** as the product name and **Microsoft Dynamics 365 Business Central (SaaS)** as the product version. Select the **Create** button to complete creation of the project.

After you create the project, send its ID to the e-mail alias specified on the main project page for safe listing purposes. You can find the project ID in the URL displayed in your browser, such as `https://lcs.dynamics.com/V2/NavProjectDashboard/[projectID]`

Once your LCS project ID has been safe listed by Microsoft, you can start performing deployments of your solution to Business Central service.

Deploying the Embed App

Prerequisites

Before deploying the Embed App, complete the following tasks:

1. Create the deployment package for the Embed App.

   The deployment package is a .zip file that contains components required for any Embed app. For more information, see Creating Deployment Packages for Embed Apps.

2. Upload your apps into the App Repository

   Upload new versions of your own apps into the App Repository using the App Management API. Don't upload Microsoft apps or apps from other ISVs. The Microsoft team regularly uploads Microsoft apps and AppSource apps to this repository, so you don't have to. For more information, see App Management API.

Upload the deployment package

Once you have the deployment package and uploaded the apps, you can upload the Embed App deployment package into the **Asset library** within your LCS project.

1. In your LCS project, select the **Asset library** action to navigate to the Asset library page.
2. Select the **+** action to open the **Upload Software deployable package file** wizard.
3. Provide the **Name** and **Description** (optional) for your package.

   The asset name you provide will be displayed on the Application Version list inside your project.

4. Select the **Add the file** action to select and upload the deployment package.
5. Wait for the package to get uploaded, then select the **Confirm** button to close the upload wizard.

Start the Embed App deployment

Now, start deploying the package you uploaded in the **Asset library** by doing the following steps:

1. Go to the **Rings** section of your project.
2. Select **Maintain** menu, then select **Deploy** action to open the **Deploy package** wizard.
3. Select the package you want to deploy in the **Package name** field.
4. Select the **Target platform**, then select **Deploy** button to start the deployment.

The deployment routine will pull the apps that are listed in the `[apps]` section of the `manifest.json` file in your deployment package from the App Repository. It will then publish and install the apps based on the instructions you provided in the `manifest.json` file.

After the environment is successfully provisioned, its status on the list of application versions will be set to **Deployed**. To allow sign-ups to be directed to this application version, you select the **Use for new tenants** check box. You can only have one application version marked this way at a time.

> **IMPORTANT**
>
> The Business Central platform is updated monthly. You are responsible for updating the Embed App to operate with the updated version of Business Central. The Embed App must run on a supported build of Business Central, for example, the current version of Business Central or one of the two immediately preceding versions. The immediately preceding versions can be both minor and major versions of Business Central. The list of supported versions is displayed in the LCS portal on the **Deploy package wizard** in the **Target platform** field. The versions, which aren't displayed on that list are out of support. You customers running on these versions can't be serviced at the service levels set forth in the published Service Level Agreement.

Onboarding customers and creating environments

Partners who support an Embed App based on Business Central online can onboard customers in two ways:

- Using the self-service IW sign-up – for acquiring a free evaluation version of the app.
- Through the Microsoft Partner Center Cloud Solution Provider (CSP) program by contacting the partner - for acquiring a paid production version of the Embed App.

Tenant provisioning is happening automatically (just-in-time) on the first attempt to sign in to the solution.

Navigating to `https://businesscentral.dynamics.com` will trigger provisioning of the Business Central tenant, while navigating to `https://[application name].bc.dynamics.com` will trigger provisioning of the tenant running on the *application name* application.

Self-Service (IW) sign-up - evaluation

The Embed App partner can choose to allow customers to use a self-service sign-up (also known as IW sign-up and viral sign-up) for their Embed App. In that case, the partner must prepare a sign-up URL that will redirect the Microsoft 365 sign-up flow to their application URL. The sign-up URL must have the following format:

```
https://signup.microsoft.com/signup?sku=6a4a1628-9b9a-424d-bed5-4118f0ede3fd&ru=https%3A%2F%2F[application name].bc.dynamics.com%2F%3FredirectedFromSignup%3D1
```

The partner can then pass the URL to their customers, either from the partner's own marketing page or in a welcome e-mail, for example.

To work with an Embed App, the customers would use a URL that looks something like this:

- Client: `https://[application name].bc.dynamics.com`
- Web Services: `https://[application name].api.bc.dynamics.com`

In contrast, to work with Business Central, they would use these URLs:

- Client: `https://businesscentral.dynamics.com`
- Web Services: `https://api.businesscentral.dynamics.com`

Once you've established the reseller relationship with the customer and added Business Central subscriptions with required number of licenses for them, you must use your own branded Embed app URL to sign in their environment and Business Central Administration center.

To create a new production environment for your customers, go to this URL:

```
https://[your application family].bc.dynamics.com/[Customer's Azure AD Tenant ID]/Production
```

To open your customer's Business Central Administration center, go to this URL:

```
https://[your application family].bc.dynamics.com/[Customer's Azure AD Tenant ID]/admin
```

Each environment that you signed up for the Embed App is then displayed on the Tenant list part in your LCS project. On this part, you can find more details about the environment, including the name and the URL to sign in to each one. You can see which environments are running on which application version by selecting application version on the list.

Monitoring the Embed app

In the LCS portal, you get access to various logs for the activities that you do in the portal:

- Deployment: logs of deployment of the application versions
- Tenant provisioning: logs of creation of the new customer environments

- Tenant upgrades: logs of customer environments upgrades
- Application errors: errors that are displayed to the customers when they work with the Embed app functionality

To get to these logs, in the Rings section of your project, select **Maintain** menu and then select **Monitor** action.

On the **Ring monitoring** page, first select the **Category**, depending on the operation for which you want to retrieve the logs. Then set the **Period (min)** field to the number of minutes back in time, for which you want to see the logs. Don't use the **All** category.

> **NOTE**
> The logs typically appear in LCS with a delay of 15 minutes.

For easier troubleshooting, we recommend exporting the logs to a CSV file, using **Export to CSV** action, and then analyzing them in Microsoft Excel instead.

It's important to always review the logs before submitting any issues to Microsoft. When reporting issues, attach the relevant logs in TXT or CSV format, don't use screenshots.

See Also

Embed App Overview
Licensing in Dynamics 365 Business Central
Components and Capabilities
Microsoft Responsibilities for Apps on Business Central online
Preparing Demonstration Environments of Dynamics 365 Business Central
Preparing Test Environments of Dynamics 365 Business Central

Using Application Family in Embed App

2/17/2021 • 2 minutes to read • Edit Online

Business Central service is a part of rich ecosystem of surrounding applications and services. One of the prominent Embed App capabilities is to promote and use the ISV brand in various places of the overall service experience.

During onboarding, the ISV provides the application family name. The application family helps identify their solution among other Business Central apps and Embed App's of other ISVs.

The following table describes different aspects of using the application family when interacting with Business Central components and surrounding services.

AREA	HOW TO USE WITH THE EMBED APPLICATION FAMILY	ADDITIONAL DETAILS
Business Central Web client	Use the application family in the URL: `[application family].bc.dynamics.com`	Web Client URL
Business Central web services	Use the application family in the URL: `[application family].api.bc.dynamics.com`	Introduction to automation APIs
Business Central mobile app	Use the application family in the URL: `ms-businesscentral://[application family].bc.dynamics.com/`	Getting Business Central on Your Mobile Device
Business Central administration center	Application family is displayed for each environment	The Business Central Administration Center
In-code URL generation (GETURL)	GETURL returns the branded endpoint, including the application family.	GetUrl Method
Development endpoint for Visual Studio Code	To publish and debug apps in a Sandbox environment, use the "applicationFamily" property in the launch.json file	JSON Files
Microsoft AppSource	To install AppSource apps, use the branded URL with the following parameters: `https://[application family].bc.dynamics.com/?noSignUpCheck=1&filter=%27ID%27%20IS%20%27[APP ID]%27&page=2503`	Installing apps by selecting a direct app link in AppSource (for example, "Get it now" or "Free trial") isn't supported. This action redirects to the standard Business Central URL.
Microsoft Power BI	Supported for connecting to the environment data from Power BI and embedding Power BI pages in the Business Central Web client.	Enabling Your Business Data for Power BI

AREA	HOW TO USE WITH THE EMBED APPLICATION FAMILY	ADDITIONAL DETAILS
Outlook Add-in	Not supported	Using Business Central as your Business Inbox in Outlook
Excel Add-in	Supported	Setting up the Excel Add-In for Editing Business Central Data
CDS integration	Supported	Integrating with Microsoft Dataverse
Power Automate, Power Apps, Logic apps	Business Central standard connector currently can't be used (support is coming soon). The ISV needs to build a custom connector. The custom connector is required anyway, also for the Business Central environments, if you want to include data from other tables that aren't included with the standard Business Central connector.	Custom connector FAQ for Azure Logic Apps, Power Automate, and Power Apps
Cloud Migration Tool	Supported	Migrating On-Premises Data to Business Central Online

See Also

Embed App Overview
Licensing in Dynamics 365 Business Central
Components and Capabilities
Microsoft Responsibilities for Apps on Business Central online
Preparing Demonstration Environments of Dynamics 365 Business Central
Preparing Test Environments of Dynamics 365 Business Central

Application Access Management for the Embed App

2/17/2021 • 4 minutes to read • Edit Online

In this article, you'll learn how you can control which customers can create online environments based on your Embed App application family, by enabling your resellers (VARS) to use the Application Management API.

Overview

Both Business Central and Embed Apps are distributed by Cloud Solution Provider (CSP) resellers. Embed Apps use standard Business Central licenses—the same licenses used for Business Central. So any reseller who sells Business Central can also create online environments that run on your Embed App by default.

As an Embed App owner, you'll want to control which customers and which resellers can create online environments based on your Embed App application. You'd also like to allow these resellers to control whether customers have access to both Business Central and your Embed App or only to the Embed App.

To support these capabilities, two features are available, depending on your role:

ROLE	WHAT YOU CAN DO
Embed App ISV	Enable application access management for all environments in your LCS project. Register VARs to give them access to the Application Access Management API.
VAR	Once you've been registered by the Embed App ISV, you can allow or block access to the Embed App or Business Central app for specific customers.

Embed App ISV: Enable application access management and register VARs

Follow these steps to provide your VARs with access to the Application Access Management API. Once they have access, they can use the API to enable or disable the creation of online environments based on your Embed App application.

Preparation

- Prepare the VAR for the change

  Once you enable application access management in your LCS project, any existing customers already using your Embed App will be blocked from access to their Embed App environments by default. It also won't be possible to create new environments based on your Embed App application for any new or existing customers. The VAR will have to explicitly enable the access for each customer by using the API explained in the section below.Make sure you coordinate this change with your VARs.

- Get the VAR's CSP organizational ID (Microsoft ID). You'll need this ID for registration.

  The VAR can find their Microsoft ID in Partner Center, on the **Account Settings** > **Organizational profile** page.

Enable application access management and register VARs

1. Go to Lifecycle Services and open your project.
2. Select the **Manage VARs** tile.
3. On the **Manage VARs access** page, specify if VARs must be able to approve customers.

> **IMPORTANT**
> This step enables application access management. Once you have allowed VARs to approve customers, it won't take effect until you save the change. Be aware that once you save, any existing VAR customers already using your Embed App and any new customers will be blocked from access by default, until the VAR explicitly enables the app for them.

4. Now you start to register VARs. Select **+Add**.
5. On the **Register new VAR Tenant**, enter the VAR's Microsoft ID in the **Tenant ID** box.
6. Enter a description in the **Description** box to make it easier for you to identify the VAR.
7. Select **Add**.
8. Select **Save** when done.

Once a VAR is registered, they can start to enable and disable the Embed App for their customers using the Application Management API.

Disabling and enabling application access management

In your LCS project, you can specify that VARs can or cannot approve customers without having to remove VARs from the list. This capability is useful if you need to temporarily allow access to all VARs and all customers. For example, suppose a VAR is registered but they're not ready to use the API yet.

> **IMPORTANT**
> If you specify that VARs cannot approve customers, even if the VARs are still listed on the **Manage VARs access** page, the environment access settings made by VARs for their customers will be ignored. The existing access settings will be enforced again once you allow the VARs to approve customers again and save the changes.

VAR: Manage customer access to the Embed App environments

As a VAR, once you've been registered by your ISV, you must explicitly give every customer access to create Embed App environments. Access is given by using the application access management API. The required steps are outlined in the sections that follow.

Prerequisite: Register an application for in your Azure Active Directory tenant

You only have to do this step once. In your Azure Active Directory (Azure AD) tenant, register an application that has delegated permission to the Dynamics 365 Business Central API. You'll use this application to call theBusiness Central Administration Center APIfor each customerthat you want to onboard.

Apart from registering an application, you'll also need an access token. For more information, see Setting up Azure Active Directory (AAD) based authentication and Getting an access token.

Onboarding customers

With application access management enabled, the typical process for getting customers up and running on a Embed App environment is as follows:

1. Connect with the customer using Partner Center:

   a. Establish reseller relationship with the customer
   b. Add necessary Business Central subscriptions
   c. Assign Business Central licenses to the customer users.

   For more information, see Connect with customers.

2. Control access to Embed App and Business Central application using the application access management API:

   a. Enable the customer to create environments based on the Embed App
   b. Block the customer from creating environments based on the Business Central application.

   For more information, see Application Access Management API. The article explains the API calls that you can make to get the list oftheavailable Embed Apps and enable or disable a specific Embed Appfor the customer.

   This work is done using your AAD application and delegated admin access and credentials.

3. Create the first Embed App environment for the customer by signing in to it for the first time, using the branded Embed App URL:

   ```
   https://[application family].bc.dynamics.com/[Customer Azure Active Directory Tenant ID]/Production
   ```

See Also

Application Access Management API
Using Application Family Managing an Business Central Embed App in Microsoft Lifecycle Services

Application Access Management API

2/17/2021 • 2 minutes to read • Edit Online

As a **Delegated Tenant Admin**, you can manage access to application families available in the service. The application family is Business Central or Embed App applications that may be provisioned through the service.

You can get the list of applications that are available to the customer tenant. From this list you can determine, by setting the access property, for which applications an environment may be created on the tenant.

Get List Of Manageable Applications

Returns a list of manageable applications by family and country code.

```
GET /admin/v2.3/manageableapplications
```

Response

Returns a wrapped array of applications.

```
{
  "value": [
    {
      "applicationFamily":  string, // The name of the family for a given Business Central offered
application. (Typically this will just be "BusinessCentral")
      "access":  boolean, // Indicates if the tenant has access to the application family/country code
combination
      "countryCode": string, // The country code of the application.
    }
  ]
}
```

Example (PowerShell)

```
$url = "https://api.businesscentral.dynamics.com/admin/v2.3"
$result = Invoke-WebRequest -Uri "$($url)/manageableapplications" -Headers @{"Authorization" = "Bearer
$token"} | ConvertFrom-Json
Write-Host $($result.value | ConvertTo-Json)
```

Control the access to Applications

Pass the application family name in the URL and a boolean in the body.

- True - enables the access.
- False - disables the access.

```
Content-Type: application/json
PUT /admin/v2.3/manageableapplications/{applicationFamily}/countries/{countryCode}
```

Route Parameters

`applicationFamily` - The name of the family for a given Business Central offered application. (Typically this value will just be "BusinessCentral")

`countryCode` - Country code for the targeted application.

Body

```
{
  <boolean> // Desired access state
}
```

> **NOTE**
> It is not possible to disable the access to an application family for the AAD tenant which already has an environment created in that family.

Expected Error Codes

`invalidInput` - the targeted property in invalid in some way

- target: {targeted tenant's AAD Id} - attempt to remove access to an application but the targeted tenant already has an environment in that application

`resourceDoesNotExist` - couldn't find the targeted application

Example (PowerShell)

```
$url = "https://api.businesscentral.dynamics.com/admin/v2.3"
$country = 'DK'
$applicationFamily = 'frentals'
$access = "true"
Invoke-WebRequest -Uri "$($url)/manageableapplications/$($applicationFamily)/countries/$($country)"  -
Headers @{"Authorization" = "Bearer $token"} -Body $access -Method Put -ContentType "application/json"
```

See Also

Application Access Management
Using Application Family

App Management for ISVs

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Each Business Central environment is built as a collection of apps. These apps include Microsoft apps and apps from AppSource that reselling partners have installed for customers. The apps are working together to provide customers with a broad set of features to address their various business, market, and industry needs.

As an authorized ISV, you can deliver your functionality or your services as apps in AppSource. For more information, see Get Started with Building Apps. To help you manage your apps running in different customer Business Central environments, Business Central provides the App Management API.

About the App Management API

The App Management API is a REST-based API. It requires that you're an authorized ISV and your apps have been registered by Microsoft. Once registered, you access the API by using this global endpoint: https://apps.businesscentral.dynamics.com.

You can use the API for the following operations:

- Make major, minor, and hotfix app updates available to customers for installation from the Business Central administration center.

  You make the updates available by uploading them to the App Repository. The new app versions will then be available on Manage Apps page of the Business Central administration.

- Retrieve the list of the customers' environments that have your app installed.
- Schedule the automatic deployment of the app hotfixes for their customers' environments.

The App Management API lets you apply modern continuous integration (CI), continuous deployment (CD), and DevOps practices to your work, for example:

- Automate operations by using Microsoft Azure DevOps Services or other available process automation tools.
- Organize role-based access control.
- Manage your apps at scale, in multiple geo locations, supported by advanced and well-controlled build, test, and release flows.

For more information about the API, see App Management API.

> **NOTE**
>
> Currently, direct access to the API is only available for the ISVs working with the Embed apps; not Add-on and Connect apps. To manage versions of Add-on and Connect apps, you use Partner Center to upload the new app versions to **Business Central** offers. The apps will undergo a technical and marketing validation before becoming available on AppSource. After passing validation, the new versions are made available in Business Central administration center to the customers that have these apps installed.

Legal information

The apps that are stored in the App Repository are governed by the Microsoft Publisher Agreement. For more information, see Publishing your business application.

The App Management API is governed by Microsoft APIs Terms of Use. For more information, see Microsoft APIs Terms of Use

See Also

Manage Apps in the Business Central Administration Center

App Management API

2/17/2021 • 13 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Entities

App

Description

The `app` entity represents a Business Central App that has been registered with the service and so it can be managed via the API. The `app` entity contains some basic metadata about the app, such as:

- its ID
- the Azure Active Directory Tenant ID of the publisher's organization
- which Azure location the respective .app files and other metadata should be stored in.

> **NOTE**
> An app doesn't refer to any specific .app file or version of the app.

Properties

NAME	DESCRIPTION	SCHEMA
ID	The ID of the app. The ID must remain the same across all app versions.	string (uuid)
publisher	The publisher's name.	string
publisherAadTenantId	The ID of the Azure Active Directory tenant that represents the publisher's organization.	string (uuid)
publisherContactEmail	The publisher's contact e-mail address.	string
storageLocation	The Azure location in which data related to this app should be stored.	string
_etag	The entity tag that contains a value unique to the entity's current state.	string

Available Endpoints

List apps

Lists all apps that match the provided (optional) filter that the current `principal` can access.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps?$filter=<odata_filter>
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
query	$filter	Restricts the set of items returned.	OData filter

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps?$filter=storageLocation eq 'West Europe'
```

Example Response

```
{
    "value": [
        {
            "id": "36d697e1-1982-43a5-9927-7f8f2a3eb361",
            "publisher": "Contoso",
            "publisherAadTenantId": "49cf52d4-fc10-41af-91ab-2d7cd011db47",
            "publisherContactEmail": "publisher@contoso.com",
            "storageLocation": "West Europe",
            "_etag": "5c47bfb0-d80c-4780-99ed-c3a3d38ac539"
        }
    ]
}
```

Country

The `country` entity represents a country that an `app` is available in.
A country is represented by its ISO 3166-1 alpha-2 (2-letter) country code.
Specific versions of Business Central Apps can be made available for specific countries.

Properties

NAME	DESCRIPTION	SCHEMA
countryCode	The ISO 3166-1 alpha-2 (2-letter) code for the country.	string
_etag	The entity tag that contains a value unique to the entity's current state.	string

Available Endpoints

List countries

Lists all countries the specified `app` has been made available in.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app to list the countries for.	string (uuid)

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-f10f36a2e213/countries
```

Example Response

```
{
    "value": [
        {
            "countryCode": "US",
            "_etag": "f6039211-0dea-481a-80a5-38498dcea011"
        },
        {
            "countryCode": "DK",
            "_etag": "bb8f8166-a6cc-4562-b135-0b13257b032f"
        }
    ]
}
```

Get country

Gets the `country` with the specified country code in the specified `app`.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app that the country belongs to.	string (uuid)
path	countryCode	The ISO 3166-1 alpha-2 (2-letter) code of the country.	string

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-f10f36a2e213/countries/US
```

Example Response

```
{
    "countryCode": "US",
    "_etag": "f6039211-0dea-481a-80a5-38498dcea011"
}
```

Add or update country

Adds a new `country` to the specified `app` or updates an existing one.

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app that the country belongs to.	string (uuid)

TYPE	NAME	DESCRIPTION	SCHEMA
path	countryCode	The ISO 3166-1 alpha-2 (2-letter) code of the country to add.	string
body	country	The country properties that should be added or updated.	Country

Example Request

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-f10f36a2e213/countries/AT

{
    "countryCode": "AT"
}
```

Example Response

```
{
    "countryCode": "AT",
    "_etag": "584915f0-6319-4aab-a23c-743c0c4d9aeb"
}
```

Principal

Description

The `principal` entity represents an Azure Active Directory user or application that has some level of access to an `app`.
Principal can have different roles that determine which actions they're allowed to do. The currently supported roles are:

- `Owner` - Owners are allowed to do all available actions on all entities.
- `Contributor` - Contributors have similar permissions to owners, except that they aren't allowed to add/update principals.
- `Reader` - Readers can read information about the various entities, but can't add/update anything.

Properties

NAME	DESCRIPTION	SCHEMA
aadTenantId	The ID of the Azure Active Directory Tenant the principal belongs to.	string (uuid)
id	The ID of Azure Active Directory user/application.	string (uuid)
roles	The list of roles the principal has assigned.	string[]
type	The principal type.	enum (User, Application)

NAME	DESCRIPTION	SCHEMA
_etag	The entity tag that contains a value unique to the entity's current state.	string

Available Endpoints

List principals

Lists all `principal`s that match the provided filter within the specified `app`.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/principals?$filter=<odata_filter>
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app to list the principals for.	string (uuid)
query	$filter	Restricts the set of items returned.	OData filter

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-f10f36a2e213/principals?
$filter=type eq 'User' and 'Owner' in roles
```

Example Response

```
{
    "value": [
        {
            "id": "c07b7af3-8c9a-4bb1-9a0b-03692ba98d6d",
            "type": "User",
            "aadTenantId": "0c1cbce0-b823-4acf-be2f-1ac3a4e81c21",
            "roles": [
                "Owner"
            ],
            "_etag": "c99541ab-a0d2-4807-86f8-e1eb1853eb4f"
        }
    ]
}
```

Get principal by ID

Gets the `principal` with the specified ID in the specified `app`.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/principals/{id}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app that the principal belongs to.	string (uuid)
path	id	The ID of the principal.	string (uuid)

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/principals/c07b7af3-8c9a-4bb1-9a0b-03692ba98d6d
```

Example Response

```
{
    "id": "c07b7af3-8c9a-4bb1-9a0b-03692ba98d6d",
    "type": "User",
    "aadTenantId": "0c1cbce0-b823-4acf-be2f-1ac3a4e81c21",
    "roles": [
        "Owner"
    ],
    "_etag": "c99541ab-a0d2-4807-86f8-e1eb1853eb4f"
}
```

Remove principal

Removes the `principal` with the specified ID in the specified `app`.
Note: Only principals with the `Owner` role are allowed to remove principals.

```
DELETE https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/principals/{id}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app that the principal belongs to.	string (uuid)
path	id	The ID of the principal.	string (uuid)

Example Request

```
DELETE https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/principals/c07b7af3-8c9a-4bb1-9a0b-03692ba98d6d
```

Add or update principal

Adds or updates the specified `principal` that belongs to the specified `app`. **Note:** Only principals with the `Owner` role are allowed to add or update principals. **Note:** The `aadTenantId` field should only be specified in the request body when the principal being added is of type `User`.

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/principals/{id}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app that the principal belongs to.	string (uuid)
path	id	The ID of the principal to add or update.	string (uuid)

TYPE	NAME	DESCRIPTION	SCHEMA
body	principal	The properties that should be added or updated.	Principal

Example Request

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/principals/c07b7af3-8c9a-4bb1-9a0b-03692ba98d6d

{
    "aadTenantId":  "20ba9ed9-d37b-4db0-ade4-f64322ad7c02",
    "Type":  "User",
    "roles":  [
                  "Reader"
              ]
}
```

When using `Type:"Application"`, the `aadTenantId` must not be used.

Example Response

```
{
    "id": "c07b7af3-8c9a-4bb1-9a0b-03692ba98d6d",
    "type": "User",
    "aadTenantId": "0c1cbce0-b823-4acf-be2f-1ac3a4e81c21",
    "roles": [
        "Reader"
    ],
    "_etag": "a98992bf-d7f6-460b-83bb-1498459a6a75"
}
```

Version

Description

A `version` represents an .app file that has been uploaded for a specific `country` in an `app`.
If a version of an app should be available in multiple countries, then the .app file needs to be uploaded to each country separately.

Properties

Version

NAME	DESCRIPTION	SCHEMA
appId	The ID of the app.	string (uuid)
availability	The version's availability.	enum (Preview, Available, Deprecated)
buildVersion	The build number of the version.	integer (int32)
countryCode	The country that the version was uploaded for.	string
dependencies	The dependencies of the uploaded package.	Dependency[]
majorVersion	The major number of the version.	integer (int32)

NAME	DESCRIPTION	SCHEMA
minorVersion	The minor number of the version.	integer (int32)
name	The name of the app.	string
packageId	The ID of the uploaded package.	string (uuid)
publisher	The publisher of the app.	string
revisionVersion	The revision number of the version.	integer (int32)
status	The version's status.	enum (Uploading, Uploaded, UploadFailed)
uploadedOn	The UTC date and time the version was uploaded on.	string (date-time)
version	The version of the app.	string
_etag	The entity tag that contains a value unique to the entity's current state.	string

Dependency

NAME	DESCRIPTION	SCHEMA
appId	The dependency app ID.	string (uuid)
name	The dependency app name.	string
publisher	The dependency app publisher.	string
version	The minimum version of the dependency.	string
incompatibleFromVersion	The initial app version of the dependency that is no longer compatible with the dependent app. If this value is set then it means that versions greater or equal to it are considered incompatible.	string

List versions

Lists all `versions` that match the provided filter.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/versions?$filter=
<odata_filter>
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app to list the versions for.	string (uuid)
path	countryCode	The code of the country to list the versions for.	string
query	$filter	Restricts the set of items returned.	OData filter

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/versions?$filter=MajorVersion eq 16 and MinorVersion eq 0
```

Example Response

```
{
    "value": [
        {
            "version": "16.0.1.2",
            "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
            "countryCode": "US",
            "packageId": "01d39741-8ab1-4fcc-b353-d9208ba475e3",
            "publisher": "Example Publisher",
            "name": "ExampleApp",
            "uploadedOn": "2019-07-09T11:54:45.5414576Z",
            "availability": "Available",
            "status": "Uploaded",
            "dependencies": [
                {
                    "appId": "1914cafc-93b9-46fd-b825-707f1743caee",
                    "name": "Example Dependency",
                    "publisher": "Other Publisher",
                    "version": "1.0.0.0"
                }
            ],
            "majorVersion": 16,
            "minorVersion": 0,
            "buildVersion": 1,
            "revisionVersion": 2,
            "_etag": "50fec40a-99b6-4743-bbe7-42e95cde2cfa"
        }
    ]
}
```

Upload version

Uploads an .app file into the specified `app` and `country` based on the provided package contents.

```
POST https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/versions
```

> **IMPORTANT**
> Make sure that you registered your app with the service and added a country to it, before you attempt to upload the app version.

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
body	requestData	The request data.	Upload Version Request

Upload version request

NAME	DESCRIPTION	SCHEMA
initialAvailability	The initial availability that the uploaded app version should have.	enum (Preview, Available, Deprecated)
packageContents	The base64-encoded contents of the package (.app file) to upload.	string (byte)

Example Request

```
POST https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/versions

{
    "initialAvailability": "Preview",
    "packageContents": <contents>
}
```

Example Response

```
{
    "version": "16.0.1.2",
    "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
    "countryCode": "US",
    "packageId": "01d39741-8ab1-4fcc-b353-d9208ba475e3",
    "publisher": "Example Publisher",
    "name": "ExampleApp",
    "uploadedOn": "2019-07-09T11:54:45.5414576Z",
    "availability": "Preview",
    "status": "Uploaded",
    "dependencies": [
        {
            "appId": "1914cafc-93b9-46fd-b825-707f1743caee",
            "name": "Example Dependency",
            "publisher": "Other Publisher",
            "version": "1.0.0.0"
        }
    ],
    "majorVersion": 16,
    "minorVersion": 0,
    "buildVersion": 1,
    "revisionVersion": 2,
    "_etag": "50fec40a-99b6-4743-bbe7-42e95cde2cfa"
}
```

Get version

Gets the `version` in the specified `app` and `country`.

```
GET
https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/versions/{versionNumber}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
path	versionNumber	The version number.	string

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/versions/16.0.1.2
```

Example Response

```
{
    "version": "16.0.1.2",
    "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
    "countryCode": "US",
    "packageId": "01d39741-8ab1-4fcc-b353-d9208ba475e3",
    "publisher": "Example Publisher",
    "name": "ExampleApp",
    "uploadedOn": "2019-07-09T11:54:45.5414576Z",
    "availability": "Preview",
    "status": "Uploaded",
    "dependencies": [
        {
            "appId": "1914cafc-93b9-46fd-b825-707f1743caee",
            "name": "Example Dependency",
            "publisher": "Other Publisher",
            "version": "1.0.0.0"
        }
    ],
    "majorVersion": 16,
    "minorVersion": 0,
    "buildVersion": 1,
    "revisionVersion": 2,
    "_etag": "50fec40a-99b6-4743-bbe7-42e95cde2cfa"
}
```

Update version

Updates the `version` in the specified `app` and `country` with the provided updated data.

Note: only some properties can be updated.

```
PATCH
https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/versions/{versionNumber}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app that the version belongs to.	string (uuid)

TYPE	NAME	DESCRIPTION	SCHEMA
path	countryCode	The country code.	string
path	versionNumber	The version number of the version to update.	string
body	version	The properties that should be updated.	Version

Example Request

Marking an app version as deprecated.

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/versions/16.0.1.2

{
    "availability": "Deprecated"
}
```

Example Request

Marks older versions of your app as incompatible with a dependency app, starting with a specific version. In such cases, make sure you upload another version of your app that is compatible with the new version of the dependency app.

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/versions/16.0.1.2

{
    "dependencies": [
        {
            "appId": "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
            "incompatibleFromVersion": "16.0.0.0"
        }
    ]
}
```

Example Response

```
{
    "version": "16.0.1.2",
    "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
    "countryCode": "US",
    "packageId": "01d39741-8ab1-4fcc-b353-d9208ba475e3",
    "publisher": "Example Publisher",
    "name": "ExampleApp",
    "uploadedOn": "2019-07-09T11:54:45.5414576Z",
    "availability": "Deprecated",
    "status": "Uploaded",
    "dependencies": [
        {
            "appId": "1914cafc-93b9-46fd-b825-707f1743caee",
            "name": "Example Dependency",
            "publisher": "Other Publisher",
            "version": "1.0.0.0"
        }
    ],
    "majorVersion": 16,
    "minorVersion": 0,
    "buildVersion": 1,
    "revisionVersion": 2,
    "_etag": "3c0f1fd2-266e-491b-82e3-74f09a975267"
}
```


Environment

Description

Represents a customer's `environment` that has a specific `version` of an `app` installed.

Properties

NAME	DESCRIPTION	SCHEMA
aadTenantId	The ID of the customer's Azure Active Directory tenant.	string (uuid)
appId	The ID of the installed app.	string (uuid)
applicationFamily	The environment's application family.	string
countryCode	The country code the environment belongs to.	string
locationName	The Azure location the environment is in.	string
name	The environment's name.	string
type	The environment's type.	enum (Production, Sandbox)
version	The version of the installed app.	string

List environments

Lists the `environment`s in the specified `app` and `country` that have the app installed.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/environments
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
query	$filter	Restricts the set of items returned.	OData filter

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/environments?$filter=version eq '16.0.1.2'
```

Example Response

```
{
    "value": [
        {
            "aadTenantId": "539a833c-4e6a-42d8-a081-27fa9d810424",
            "version": "16.0.1.2",
            "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
            "countryCode": "US",
            "applicationFamily": "Business Central",
            "locationName": "West Europe",
            "name": "MyCustomer",
            "type": "Production"
        }
    ]
}
```

Environment Hotfix

Description

An `environment hotfix` represents the action of pushing out an update (new `version`) of an `app` to a customer's `environment` as part of fixing a critical issue.
Only `version`s where the major and minor components haven't changed can be pushed as hotfixes.

Properties

NAME	DESCRIPTION	SCHEMA
appId	The ID of the app.	string (uuid)
completedOn	Date and time (UTC) when the hotfix was applied.	string (date-time)
countryCode	The country code.	string
createdOn	Date and time (UTC) when the hotfix request was created.	string (date-time)

NAME	DESCRIPTION	SCHEMA
environmentAadTenantId	The ID of the customer's Azure Active Directory tenant.	string (uuid)
environmentApplicationFamily	The environment's application family.	string
environmentName	The environment's name.	string
errorMessage	The error message provided when the hotfix was unable to be applied.	string
id	The ID hotfix request.	string (uuid)
ignoreUpgradeWindow	Determines whether the environment's upgrade window should be ignored when applying the hotfix.	boolean
runAfter	Date and time (UTC) after which hotfix should start to be applied.	string (date-time)
sourceAppVersion	The version of the app that was installed before the hotfix was applied.	string
startedOn	Date and time (UTC) when the hotfix started to be applied.	string (date-time)
status	Status.	enum (Scheduled, Running, Succeeded, Failed, Canceled, Skipped)
targetAppVersion	The version of the app containing the fixes that should be applied. The value is the version that the app will be updated to.	string

List environment hotfixes

Lists all the hotfix operations that were requested for a specific `app` and `country`.

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/environmentHotfixes?
$filter=<odata_filter>
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
query	$filter	Restricts the set of items returned.	OData filter

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/environmentHotfixes?$filter=targetAppVersion eq '16.0.1.2'
```

Example Response


```
{
    "value": [
        {
            "id": "db311b6a-062e-4756-b17e-73aeb89c45cc",
            "environmentAadTenantId": "539a833c-4e6a-42d8-a081-27fa9d810424",
            "sourceAppVersion": "16.0.0.5",
            "targetAppVersion": "16.0.1.2",
            "status": "Scheduled",
            "createdOn": "2020-03-05T15:41:20.6468128Z",
            "runAfter": "2020-03-05T17:30:00.0000000Z",
            "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
            "countryCode": "US",
            "environmentApplicationFamily": "Business Central",
            "environmentName": "MyCustomer",
            "environmentType": "Production"
        }
    ]
}
```


Get environment hotfix

Gets an `environment hotfix` operation by its ID.

This endpoint can be used to track the status/outcome of a hotfix request.

```
GET
https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/environmentHotfixes/{id}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
path	id	The ID of the hotfix request.	string (uuid)

Example Request

```
GET https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/environmentHotfixes/db311b6a-062e-4756-b17e-73aeb89c45cc
```

Example Response

```
{
    "id": "db311b6a-062e-4756-b17e-73aeb89c45cc",
    "environmentAadTenantId": "539a833c-4e6a-42d8-a081-27fa9d810424",
    "sourceAppVersion": "16.0.0.5",
    "targetAppVersion": "16.0.1.2",
    "status": "Scheduled",
    "createdOn": "2020-03-05T15:41:20.6468128Z",
    "runAfter": "2020-03-05T17:30:00.0000000Z",
    "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
    "countryCode": "US",
    "environmentApplicationFamily": "Business Central",
    "environmentName": "MyCustomer",
    "environmentType": "Production"
}
```

Schedule environment hotfix

Schedules a hotfix for a specific `environment` to the specified `version`.
The hotfix operation ID that is returned can be used to track the status/outcome of the operation (see: Get environment hotfix).

```
POST https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/environmentHotfixes
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
body	environmentHotfix	Environment hotfix to schedule.	Environment Hotfix

Example Request

```
POST https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/environmentHotfixes

{
    "environmentAadTenantId": "539a833c-4e6a-42d8-a081-27fa9d810424",
    "targetAppVersion": "16.0.1.2",
    "runAfter": "2020-03-05T17:30:00.0000000Z",
    "environmentApplicationFamily": "Business Central",
    "environmentName": "MyCustomer",
    "environmentType": "Production"
}
```

Example Response

```
{
    "id": "db311b6a-062e-4756-b17e-73aeb89c45cc",
    "environmentAadTenantId": "539a833c-4e6a-42d8-a081-27fa9d810424",
    "sourceAppVersion": "16.0.0.5",
    "targetAppVersion": "16.0.1.2",
    "status": "Scheduled",
    "createdOn": "2020-03-05T15:41:20.6468128Z",
    "runAfter": "2020-03-05T17:30:00.0000000Z",
    "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
    "countryCode": "US",
    "environmentApplicationFamily": "Business Central",
    "environmentName": "MyCustomer",
    "environmentType": "Production"
}
```

Update environment hotfix

Updates an existing `environment hotfix` operation.
It can currently only be used to cancel a requested hotfix that hasn't yet started running.

```
PATCH
https://apps.businesscentral.dynamics.com/v1.0/apps/{appId}/countries/{countryCode}/environmentHotfixes/{id}
```

Parameters

TYPE	NAME	DESCRIPTION	SCHEMA
path	appId	The ID of the app.	string (uuid)
path	countryCode	The country code.	string
path	id	The ID of the hotfix request.	string (uuid)
body	environmentHotfix	Environment hotfix properties to update.	Environment Hotfix

Example Request

```
PATCH https://apps.businesscentral.dynamics.com/v1.0/apps/41a68924-7fcf-4fd0-9200-
f10f36a2e213/countries/US/environmentHotfixes/db311b6a-062e-4756-b17e-73aeb89c45cc

{
    "status": "Canceled"
}
```

Example Response

```
{
    "id": "db311b6a-062e-4756-b17e-73aeb89c45cc",
    "environmentAadTenantId": "539a833c-4e6a-42d8-a081-27fa9d810424",
    "sourceAppVersion": "16.0.0.5",
    "targetAppVersion": "16.0.1.2",
    "status": "Canceled",
    "createdOn": "2020-03-05T15:41:20.6468128Z",
    "runAfter": "2020-03-05T17:30:00.0000000Z",
    "appId": "41a68924-7fcf-4fd0-9200-f10f36a2e213",
    "countryCode": "US",
    "environmentApplicationFamily": "Business Central",
    "environmentName": "MyCustomer",
    "environmentType": "Production"
}
```


Getting detailed error messages

To get detailed error messages from t API calls, wrap the calls in a try-catch block, as shown in the following PowerShell example:

```
try {
    Invoke-WebRequest ....
}
catch [System.Net.WebException]
{
    $Exception = $_.Exception
    $respStream = $Exception.Response.GetResponseStream()
    $reader = New-Object System.IO.StreamReader($respStream)
    $respBody = $reader.ReadToEnd() | ConvertFrom-Json | ConvertTo-Json -Depth 100
    $reader.Close();
    Write-Error $Exception.Response.Headers["ms-correlation-x"]
    Write-Error $Exception.Message
    Write-Error $respBody -ErrorAction Stop
}
```

Please make sure you provide full output in textual format when reporting the issues to Microsoft.

See Also

App Management for ISVs

Get Started as a Reseller of Business Central Online

2/17/2021 • 9 minutes to read • Edit Online

If you want to build your business on Dynamics 365 Business Central online, you must get set up as a reseller in the Microsoft Partner Center. In this article, we take you through the first four steps in your journey.

Step 1: Become a partner

Becoming a Microsoft partner gives you access to the Microsoft resources needed to sell solutions. The steps in this section below are required to gain access to the programs that enable are intended to help you get set up to sell Business Central.

In order to be able to service Business Central licenses and provide help and support to your customers, your company must have a Microsoft Partner ID (MPN ID), and you must enroll in the Cloud Solution Provider (CSP) program.

Join the Microsoft Partner Network

Microsoft Partner Network (MPN) membership unlocks our best resources to differentiate your business, take your product to market, and sell your solutions. To become a partner, you must join the Microsoft Partner Network (MPN), at which time you will be assigned an MPN ID. MPN membership is free to all partners; you can enroll in the MPN here.

Once signed up, you will get an MPN ID – your gateway to access all the membership resources and benefits for your partnership with Microsoft. There is no cost to obtain a MPN ID as a Network member, and with options to upgrade to an Action Pack subscription or work toward a competency, you can access even more benefits.

Set up your Partner Center account

Once you have joined the Microsoft Partner Network (MPN), you can set up your Partner Center (PC) account. The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Your Partner Center account provides you with access to pricing information, tools and services, and enables you to manage admin credentials for your company's work account. Partner Center is also where you can purchase or renew subscriptions to Microsoft Action Packs, create a business profile to receive and manage sales leads from Microsoft, and see if you qualify for co-selling opportunities.

Enroll in the CSP program

The Cloud Solution Provider (CSP) program helps your company to be more involved in your customers' businesses, beyond reselling licenses. In CSP, you can choose to enroll as an *indirect reseller* or a *direct bill partner*.

In most cases, you will enroll as an *indirect reseller* and then work with an *indirect provider*, also referred to as a *distributor*, who then manages all interaction with Microsoft in terms of licensing and technology, so that you can focus on sales and support. If you decide to enroll as a *direct bill partner* in order to fully own the end-to-end relationship with both customers and Microsoft, make sure that you meet the eligibility requirements. For more information, see Enroll in the Cloud Solution Provider program in the Microsoft Partner Center content.

The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Some indirect providers (distributors) provide their resellers with a custom portal that optimizes and enhances the experience beyond the Partner Center. They can also provide indirect resellers with an API to automate some of the customer onboarding steps. Contact your indirect provider to find out more.

Both indirect resellers and direct bill partners can access and support their customers' Business Central by setting up a reseller relationship with them.

To service customers in a specific country, your partner company's Azure Active Directory (Azure AD) tenant and CSP account must be registered in the regional CSP market that covers that country. For more information, see Cloud Solution Provider program regional markets and currencies.

> **NOTE**
>
> When you buy Business Central offers on behalf of your CSP customers, the CSP offer must be available in both your own tenant's country and in your customer's tenant's country. For example, if your tenant is located in Slovakia and the customer's tenant is in Germany, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.
>
> Similarly, if your tenant is located in Germany and the customer's tenant is in Slovakia, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.

In the Microsoft Partner Center documentation, you can learn how to request a reseller relationship with customers, assign licenses to users, and create new subscriptions. Business Central is one of the subscriptions that you can create, and there are Business Central-specific license types that you can assign to users.

You must also assign certain roles to users in your own organization so that they can support your customers.

Add users from your own organization

In most cases, your partner company includes employees with different responsibilities, and you can assign different roles depending on people's responsibility. This way, you can be very explicit about who from your staff can access your customers' data, for example.

For each user, you can assign permissions for 2 categories of tasks:

- Managing your organization's account

  This controls access to the functionality of the Partner Center portal

- Assisting your customers

  This controls access to your customers' environments

When you establish a reseller relationship with a customer in Partner Center, you must specify which people from your organization will assist that customer's tenant as either *Admin agent* or *Helpdesk agent*. These users can manage implementation, support, and troubleshooting tasks for your customers. Once the reseller relationship with a customer is established, they will be able to login into the Business Central environments of the customer without a license. The number of partner users that can access customer environments is not restricted.

Both roles will provide your users with exactly the same level of access to the customer's Business Central environment. However, they have very different capabilities to manage the customer's Active Directory and other subscriptions that the customer might have. For more information, see Admin roles.

This way of accessing customer resources is called *delegated administration*, and the partner users are therefore called *delegated administrators* or *delegated admins* in daily shorthand. For more information, see Delegated Administrator Access to Business Central Online.

> **NOTE**
>
> These users cannot provide accounting services for the customers. For this purpose, the customers must use the **External Accountant** license, which is also available via CSP.

Caution

In the Microsoft 365 admin center and Microsoft Azure Management portal, both customers and partners can invite external users (guests) into their Active Directory. When a partner user is added as a guest to the customer's Azure AD, they can no longer log in as a delegated admin into the customer's Business Central. In order to log in, the local user (guests or native) must have a valid Business Central license assigned to them.

Step 2: Go to market

When you become a Microsoft Partner Network member, you gain access to membership benefits that can help you build and grow your business. For more information, see Explore your Go-To-Market with Microsoft offers in the Partner Center docs.

As a Dynamics 365 reseller, you benefit from Microsoft's investments in an always up-to-date modern platform, you can bundle recognized apps from the Microsoft commercial marketplace into an offering that fits the needs of your customers, reach more customers by using Microsoft's commercial marketplace to promote your packaged consulting service offerings or customization services, and streamline your own processes and build tools with Power BI, Power Automate, and Power Apps connected to Business Central.

Specifically for going to market with Business Central, Microsoft provides resources and guidance on the Business Applications for small and medium-sized businesses (SMBs) site. For more information, see Business Central Go-To-Market resources.

Step 3: Give powerful demos

You can create a trial environment based on content in cdx.transform.microsoft.com.

For more information, see Preparing Demonstration Environments of Dynamics 365 Business Central.

Step 4: Help your customers get started

When you onboard customers to Business Central, you have access to their account as a delegated administrator and can help them get things set up. For more information, see Administration of Business Central Online.

Connect with customers

As a CSP partner, you can manage a customer's subscriptions and services on their behalf in the Partner Center by establishing a reseller relationship with your customers. If they already have an account, such as if they currently use Microsoft 365, other Dynamics 365 apps, or PowerApps, for example, you can send them an invitation straight from the Partner Center. For more information, see Connect with customers in Partner Center (indirect providers/distributors) and Connect with customers (indirect resellers).

When the customer's internal administrator receives the invitation link and navigates to it, they must acknowledge that they have read the Microsoft Customer Agreement and that they can authorize you as their reseller on behalf of their organization. For more information, see Confirm customer acceptance of the Microsoft Customer Agreement.

When a customer accept this reseller relationship, delegated administration privileges are granted to the partner. For more information, see Delegated Administrator Access to Business Central Online.

If you are working as an indirect reseller, your indirect provider (distributor) must associate your customers with

you in their Partner Center.

The customer can decide to remove their partner's delegated admin privileges from their tenant, but still retain the relationship with their partner for subscription and license renewal purposes. Removing delegated admin privileges will block the partner's access to the customer's Business Central. The access can be restored if the partner sends the reseller relationship request to the customer again.

If the customer wants to cancel their relationship with their partner, the partner must remove the relationship in Partner Center.

Get Business Central right for the customer

The default version of Business Central is just that - a default version. In many cases, you'll enhance the default version with apps from the Microsoft commercial marketplace. But you can also customize pages for a profile and change which UI elements are visible. For more information, see Customize Business Central in the business functionality content.

If your customer wants more tweaks, you can create customizations of profiles and pages in code. For more information, see Customizing the User Interface for User Roles in the developer content.

Step 5: Configure the support experience

As a Business Central reselling partner, you are an administrator of your customers' Business Central tenants, and you are the first line of support. This means that you will get requests for support from your customers that you must triage, investigate, and either resolve or escalate to Microsoft. For more information, see Technical Support for Business Central.

Step 6: Maintain your customers' Business Central

As a Business Central reselling partner, you are the administrator of the Business Central tenants of your customers. You are expected to help your customers maintain their solution, including setting the upgrade window, monitoring telemetry, updating customizations, and managing apps.

For more information, see Administration of Business Central Online and The Business Central Administration Center.

See also

Administration of Business Central Online
Deployment of Dynamics 365 Business Central on-premises
Trials and Sign-ups for Business Central Online
Licensing in Dynamics 365 Business Central
Learn how to partner with indirect providers in the Cloud Solution Provider program

FAQ for Developing in AL

2/17/2021 • 2 minutes to read • Edit Online

This topic contains a number of frequently asked questions and answers to these questions.

How do I get started?

For an overview of developing apps for Dynamics 365 Business Central, see aka.ms/GetStartedWithApps

Next, follow the Getting Started with AL to set up the tools.

Which version of the AL Language extension should I use?

For Dynamics 365 Business Central cloud sandboxes you must use the AL Language extension available in the Visual Studio Code Marketplace.

For the latest Developer Preview releases you must use the AL Language extension that is available in the *next major* artifact through the "Ready! for Dynamics 365 Business Central" program on Microsoft Collaborate.

How do I enable the debugger?

To read about enabling debugging in AL, see here Debugging. To read about snapshot debugging, see Snapshot Debugging.

Can I create something similar to Menusuites?

In the AL Language extension, the concept of Menusuites is not supported. The two primary purposes of Menusuites are:

- Making pages searchable
- Making pages accessible through a navigation structure

The first purpose can be achieved in Extensions by using the new properties added to Pages and Reports. For more information, see Adding Pages and Reports to Search.

The second purpose can be achieved by extending the Navigation Pane page and/or by adding Actions to other existing pages that can serve as a navigation starting point. For more information, see Adding Menus to the Navigation Pane.

How do I upgrade Extensions V1 to Extensions V2?

For information on upgrading, see the following topics: Upgrading Extensions v2 and Converting from Extensions v1 to Extensions v2.

File APIs are not available in Extensions V2. What do I do?

Code that relies on temporary files must be rewritten to rely on `InStream` and `OutStream` types. Code that relies on permanent files must be rewritten to use another form of permanent storage.

DotNet types are not available in Extensions V2. What now?

For cloud solutions .NET interop is not available due to safety issues in running arbitrary .NET code on cloud

servers.

With the AL Language extension, you can find AL types that replace the most typical usages of .NET like HTTP, JSON, XML, StringBuilder, Dictionaries, and Lists. Many .NET usages can be replaced directly by the AL types resulting in much cleaner code. For more information, see HTTP, JSON, TextBuilder, and XML API Overview.

For things that are not possible to achieve in AL code, the recommendation is to use Azure Functions to host the DLL or C# code previously embedded and call that service from AL.

Extensions published from Visual Studio Code or created using Designer have disappeared from a sandbox environment. Why?

Extensions that have been published to a sandbox environment from Visual Studio Code or created using Designer are removed when the sandbox environment is updated or relocated within our service. However, the data of an app is not removed, so you only have to re-publish and install the app to make it available.

For more information, see Sandbox Environments.

See Also

Getting Started with AL
Keyboard Shortcuts
AL Development Environment

FAQ about Library and Dependency Apps in Business Central

2/17/2021 • 2 minutes to read • Edit Online

This section contains answers to frequently asked questions about library apps and dependency apps in Business Central.

What is a library app?

A library app contains common code that other apps depend on. If you are going to have multiple AppSource apps that all share common code, such as licensing, registration,and so on, you can put that common code into a library app and have the AppSource apps depend on the library app.

How does the library app get installed to a tenant?

A library app does not appear on AppSource. It only lives in Business Central. Our service is built to install the library apps behind the scenes. Here is how it works. Customers finds an app they want to install on AppSource. That customer does not know the app even has library app(s). When they click to install the app, our service looks into that apps json/manifest to see if it first needs to install any library/dependency apps it may depend on. If so, it installs them first before installing the main AppSource app.

What is a dependency app?

A dependency app isn't much different from a library app. A dependency app does have its own offer on AppSource. For example, you might have AppSource "App A" that serves a purpose on its own and is listed on AppSource. You then also have "App B" as its own offer on AppSource but it does depend on code within "App A" and needs "App A" to be installed before it can be installed. This is really the only difference though. Behind the scenes, library and dependency apps behave the same when it comes to walking that dependency chain and auto installing any app the top-level app depends on.

Can I have multiple library apps for my AppSource app?

Yes. Just make sure you list all library apps as dependencies within the AppSource app json file/manifest.

When I get the latest updated version of my app, why don't I get the updated library/dependency apps that my AppSource app depends on?

Currently, updating of apps does not handle the library/dependency chain. This work is planned for a future release. For now, you need to uninstall and reinstall the library/dependency apps as well.

What are the validation requirements for library apps

Library apps only get validated technically, but don't go through any type of marketing validation. Also, the library apps are expected to be covered by the user scenarios and tests that you submit for the core app. The technical requirements are described here.

See also

FAQ about Managing and Submitting your Business Central Offer
FAQ about Updating your Business Central App
Update Lifecycle for AppSource Apps FAQ
The Lifecycle of Apps and Extensions for Business Central

FAQ about Testing your Business Central App

2/17/2021 • 4 minutes to read • Edit Online

This section contains answers to frequently asked questions about testing your app when you submit an app for Business Central.

Since my app goes through validation, do I really need to thoroughly test my app?

Yes. It is your app and you are the expert, so you should have a vested interest in high quality. The main goal of validation is to ensure app code is meeting requirements and policy. The testing done during validation is only to ensure good user experience on the most common app scenarios. We do not test every scenario of an app. Using customers to find bugs is not the proper approach.

Is it ok to submit my app for validation before we complete our own testing?

No. Please do not submit your app for validation until it has been 100% tested. This will lead to many delays in the validation process otherwise. Also, it wastes time and resources.

Is it ok to test my AppSource app in an on-premises environment?

No. You must test your app using Business Central online. Although Business Central online and on-premises are very similar, they are not the exact same. If you only test on-premises, invariably issues will be found in our validation testing.

Does it really matter what version of Business Central I test on before submitting my app for validation?

Yes. It is critical that you always test on the latest version at the time when you are ready to submit for validation. Testing on the wrong version usually leads to validation failure. For example, let's say the latest version of Business Central is 15.4 at the time when you submit for validation You tested on 15.0. The product has changed between those versions, and deprecated features or other changes could result in your app behavior changing.

We recommend that you consider using Docker containers. Use the `Get-BcArtifactUrl` function in the BCContainerHelper PowerShell module to give you the artifact URL for the latest sandbox build for the specified country. You don't have to figure out what product version is active at time of submission. That function does it for you.

How thorough should our testing be?

You should always test with 100% coverage, or at least as close to 100% as you can get. The testing should be a combination of automated and manual tests. The apps with good testing infrastructure behind them are the most successful.

Do I have to test on every country that I intend to support with my app?

Yes. If you support multiple countries, test your app on every country. Each country's code base is slightly

different from both the base application and other countries. It is critical to make sure that the app publishes, syncs, and installs on every country you support. Because an app installs fine on one country doesn't mean it will publish/install fine on another. We have seen many times where it passes on one but fails on another during our validation.

Do you have recommendations on maintenance testing of our apps?

Yes. You should be testing your apps against our various build branches. Through Docker, you have access to our latest public sandbox builds and through the "Ready! for Dynamics 365 Business Central" program on Microsoft Collaborate you can also get access to *Next Minor* and *Next Major* builds. Test often, especially against the *Next Minor* build. This allows you to catch any bugs that may arise from core changes in the product.

I only made minor code changes in my updated app. Can I test just these changes?

No. You should always test 100% coverage no matter what. Testing only what you changed is not the correct approach. Even minor changes can lead to breaking changes where you least expect it.

Do I need to do upgrade testing?

Yes, this is a must. If an app fails to upgrade, customers are unable to get your updated app. This is one of the common failures we see today in validations. You should want your app upgrade to work optimally.

Any recommendations on upgrade testing?

Test the upgrade with extensive app data included. Many of the upgrade failures for customers are data related. The upgrade fails because specific data scenarios were not considered for testing. Or worse, the upgrade succeeds, and data is lost.

Do I only need to do upgrade testing from previous version to current?

No. It is very important you test from various previous versions of your app. This is because we do not automatically upgrade apps for minor releases. You could have a tenant back on version 1.0.0.0 of your app and have to jump all the way to version 1.0.0.5. We don't guarantee direct upgrades of apps from their most previous version.

See also

FAQ about Updating your Business Central App
FAQ about Library & Dependency Apps in Business Central
Update Lifecycle for AppSource Apps FAQ
The Lifecycle of Apps and Extensions for Business Central

FAQ about Updating your Business Central App

2/17/2021 • 5 minutes to read • Edit Online

This section contains answers to frequently asked questions about updating your app for Business Central.

Is it the same process for an app update as the first version?

Yes. You upload the updated app file into Partner Center and submit as normal. It goes through a more scaled-back validation process than your original version, but it does get validated. If it fails validation, it comes back to you for fixing. If it passes validation, it gets checked into Business Central.

What are some "need to know" considerations with an updated app?

When you submit an updated version of your app, you must increase the version number in the app's json/manifest files. Business Central doesn't allow overwrites. So we need the version number increased for us to check the updated app into our service upon it passing validation.

Never change the app's App ID in the json/manifest files. This is must stay the same across versions for various reasons, not least for upgrade reasons.

When is my updated app available for tenants to install?

As soon as your updated app passes validation and is checked into our service, it then becomes the active version (for whatever the current Business Central version is at that time). Even though your offer might still show as in progress in Partner Center, the updated app is active and ready to install. Also, even though your version number in AppSource might show as older, tenants will still get this latest updated version. For example, your updated version might be version 1.0.0.5. And in AppSource it might still show 1.0.0.1. Tenants will get the version 1.0.0.5.

What version of Business Central is my updated app compatible with?

When you submit your app for validation, we validate it against the latest version at that point in time. Once the app passes validation, it is then checked into our service and configured for that version of Business Central, unless a newer version has been released in the meanwhile. For example, if you submit your app and the latest version of Business Central at that time is 16.1, your app will then be compatible and configured for 16.1. When versions 16.2, 16.3, and so on roll out, your app is automatically configured for the latest version. Tenants that are on earlier versions of Business Central will not get that updated version. They still however can install the previous versions of your app. They will get the version of your app that is configured for the version of Business Central that their tenant is on.

When tenants have their Business Central version upgraded, do apps ever get automatically updated?

For minor releases, we do not auto-update apps. This is because of customer feedback and them not wanting their apps auto updated. The only exception to this is if an app will be broken in a minor release due to changes in the base product. In this case, we configure that app in our service as required. When tenants then get upgraded to that minor release, if they have that app, it then gets updated automatically. This is to avoid the app being broken for the customer.

For major releases, we do auto-update every app on every tenant to the app's latest available version. We

consider major releases to be our "refresh" releases.

How do tenants that already have an existing version of my app get my latest updated version?

By uninstalling and reinstalling the app. When doing the reinstall, it calls into our service and finds the latest active version of an app. There is a new **Manage Apps** page in the Business Central administration center that will soon allow updating of AppSource apps without having to do uninstall and reinstall.

Do I have to submit an updated version of my app for the major releases?

No. The only reason you would need to submit an updated version of your app for a major release is if your app is going to be broken in that release. If your app will not be broken, then your latest available version of the app will be rolled over to that major release.

How frequent should I submit updated versions of my app?

We recommend that you bundle more bug fixes and features so that your app doesn't have to be updated frequently. This has been voiced by our Business Central customers. They do not want to be constantly updating their apps in their tenants. We recommend a minimum 1-month app update cadence.

What if I have a critical hotfix?

We treat critical hotfixes with the utmost importance. We do have a process around this. Additional information on this hotfix process can be found here.

If I make changes to the library app, must I also submit an update for the AppSource app?

You would upload the updated library app to Partner Center, leave the main app as is, and submit for validation. We then see that the main app has not changed and only validate the library app.

But wouldn’t I need to change the dependency in my main app’s json to reference the updated library app file?

No. The version number in the dependency listing in the json file to an app is a minimum version. The main app is essentially saying, "I need version 1.0.0.1 or greater" of the library app. For example, the AppSource app lists version 1.0.0.1 for the library app, and that means that it can also use version 1.0.0.2.

Why don't I see the updated version of my app in my sandbox tenant?

Your tenant is on an older version of Business Central and has not yet been upgraded to the latest version of Business Central. The latest updated version of your app is only compatible with the latest Business Central version and later. If you upgrade your tenant to the latest version, you can then update your app.

See also

FAQ about Managing and Submitting your Business Central Offer
FAQ about Library & Dependency Apps in Business Central
Update Lifecycle for AppSource Apps FAQ

FAQ about Managing and Submitting your Business Central Offer

2/17/2021 • 2 minutes to read • Edit Online

This section contains answers to frequently asked questions about managing the offer in Partner Center when you submit an app for Business Central.

If I click the Go Live button, does that mean my app will go live to AppSource?

No. This button is deceiving, and I wish it were worded differently. When you click this button, it triggers our validation process and puts your app into our validation queue.

What should I know about the Review and publish button?

This is the button you should click when you want to submit your app for validation each time. Once you click the button, it takes you to a Review and publish page. Make sure all checkboxes are marked on that page. This pulls in all changed data into our validation queuing system properly.

I only made marketing changes. Will my app have to go through technical validation?

No. If you do not change your app in a submission, we skip technical validation.

See also

FAQ about Updating your Business Central App
FAQ about Library & Dependency Apps in Business Central
Update Lifecycle for AppSource Apps FAQ
The Lifecycle of Apps and Extensions for Business Central

Marketing Validation FAQ

2/17/2021 • 2 minutes to read • Edit Online

Where do I state the countries, editions and languages that my offer supports?

You are required to state the countries, editions and languages that your offer supports in the very bottom of your offer's description text. You can use the following format:

Supported Editions:

The app supports the Essentials and Premium Editions of Microsoft Dynamics 365 Business Central.

Supported Countries:

Canada, Mexico and United States

Supported Languages: This app is available in English (United States) and Spanish (Mexico).

Do you have any tips and tricks for what I should write in the description text?

Yes. We have detailed guidelines and good advice about that here.

How do I refer correctly to the product?

When you mention the product, both throughout your description text, as well as in your marketing material, you need to refer to the product, in the following way:

First mentions: Microsoft Dynamics 365 Business Central

Secondary mentions: Dynamics 365 Business Central

Subsequent mentions: Business Central

Therefore, you cannot use any abbreviations such as "MS Dyn 365 BC" or "Microsoft Dynamics NAV".

What are the requirements for my offer's help and support page?

You are required to submit two distinct pages for support and help i.e. they cannot be the same. You can see what you have to include on both pages in the table below.

HELP	SUPPORT
Learning material such as FAQs, step by step guides, video tutorials, webinars etc.	At least two contact options (e.g. e-mail, phone, chat) and a defined SLA for how much time it takes before you answer to support inquires.

Can I use the Microsoft Dynamics 365 Business Central logo?

No, you cannot use the Business Central logo, as it's a Microsoft trademarked logo. However, you can use the "Get it from Microsoft AppSource" badge, which you can find here.

See Also

Marketing Validation Checklist

Update Lifecycle for AppSource Apps FAQ

2/17/2021 • 5 minutes to read • Edit Online

Please see the following sections for frequently asked questions regarding updating apps on AppSource.

I want to submit an updated version of my app. What is the process for that?

For any updated version of your app (small or large changes), you follow the same submission as your original version. It must go through the same validation process. The following are the steps that you must follow:

- Increase the version number of your app within it's json/manifest file.
- Do not change the app's AppID within it's json/manifest file. That needs to remain the same for life of the app
- Include an upgrade codeunit and ensure that it works.
- Upload the app file to your existing Partner Center offer.
- In Partner Center, edit the version field to match what is now in your updated app file.
- Make any other edits in Partner Center as needed.
- Submit for validation.
- Validation takes place as normal (against current production version of Business Central at time of submission):
  - Code review is needed for avoiding violations and ensuring requirements are met.
  - Test validation is scaled back for updated versions.

What happens when my updated app passes validation?

The app is checked into our service and becomes the active version for that current production release of Business Central. If a tenant is already upgraded to the current release, they can install this updated version of your app.

Does Microsoft now update my app (to this latest updated version) on all tenants that already have a previous version?

Microsoft will only force update apps during the 2 major releases (release wave 1 and 2). For these releases, each tenant will have their AppSource apps force updated to the latest available versions in our service.

How do I update the apps on my tenant for minor releases then?

If a tenant wants to update the apps on their tenant during the minor releases, the tenant admin needs to handle this. Here are the steps you follow to update your apps:

- Login to your Business Central Web client instance.
- Navigate to the **Extension Management** page.
- Find the app and uninstall it.
- Reinstall the app.

That gives you the latest available version in our service.

How often should I submit updates of my app?

Our recommendation is to pack more bug fixes and features into less frequent updates. Try to avoid frequent submissions containing very few changes. Being on a more frequent cadence than Business Central (monthly) is not advised. This leads to lower churn to production tenants.

What if a customer reports a critical bug in my app and needs an immediate hotfix version of my app?

We do have a fast track validation process for situations like this. These types of situations become top priority in our queue. A very quick validation takes place with the goal of having the hotfix in our service that same day. The following is the hotfix process:

- Fully test the hotfix version of your app to ensure it fixes the problem and to make sure no other issues are introduced.
- Submit the app via Partner Center per the normal process.
- Email rweigel@microsoft.com to notify him of the hotfix situation.
- Provide justification as to why this is critical. Some definitions of critical being:
  - App is causing the tenant to be unresponsive or unusable with no workaround
  - App is serving a critical business process and that process cannot be executed without a fix
- If justification proves to be critical, hotfix process can proceed.
- Quick Validation takes place ASAP and app is uploaded to our service.
- Tenants can now install this version of your app.

> **NOTE**
> Please ensure that you are only using this process for critical situations. Do not try to use it for minor bug fixes.

Do you have any tips for us when submitting updates of our app?

Yes, we have some valuable tips we would like to share. These are tips that can save you time in the validation process. They will help lead to fewer (and possibly zero) failures during validation. Most importantly; they will lead to fewer issues being found in production by customers.

- Follow the checklist, for more information see Technical Validation Checklist. The checklist is ever evolving and requirements might change or be added. You might miss something from the checklist, leading to validation failure and delaying the passing of your updated app.
- Use AppSourceCop, for more information see Using the Code Analysis Tool. This helps to catch any missing prefix/suffix and DataClassification. Too often we see these fail the updated versions of apps.
- Sign your app. This fails many app validations. We try to publish the app during validation and it is not properly code-signed leading to failure to publish.
- Publish and install your app. This is another big validation failure we see too often.
- Test your app's functionality with 100% coverage. You are the expert on the app and know it best. If you are only testing a small percentage of your app, customers will most likely find issues resulting in you having to update your app more often. And if customers are the ones finding your app issues, they may decide to uninstall it. You should have a vested interest in providing a quality app.
- Test the upgrade of your app. upgrade from the previous version to this latest. Your updated app will not pass validation until the upgrade works. If it fails in our validation, we will return it to you, leading to a delay in it going to our service.

NOTE

The tips that we provide above are for your benefit to pass validation the first time through each submission. And to emphasize these points, think of the delays that can arise when you do not follow these tips. If you submit, it can take a couple days before we validate the app. Once we validate it, if all is good, it should pass quickly. If we find issues that lead us to fail the validation, we send the app back to you as you have to make fixes. It could then take you days to properly fix. Next time you submit, your app does not go to the front of the queue. It begins at the bottom again which means it could be another couple days before we validate it again. By following our tips above, you can avoid those delays. Spend a bit more time up front finding those issues yourself, leading to a quicker path to our service.

See Also

Retaining table data after publishing
Checklist for Submitting Your App
Upgrading Extensions
Using the Code Analysis Tool

FAQ About the Windows Client and Business Central

2/17/2021 • 5 minutes to read • Edit Online

The first releases of Business Central on premises included an installed client derived from Dynamics NAV. Starting with 2019 release wave 2, this legacy component, referred to as "the Windows client", will no longer be available for Business Central. Find answers for some of the most common questions here.

I have heard "modern clients only". What is this about?

Businesses and users want to be reassured that only the newest, most advanced, and up-to-date tools are being used to access their data. With Business Central 2019 release wave 2, released October 2019, users switch to the **modern experience** in the browser ("the web client"), the Android or iOS mobile apps, or the Windows 10 desktop app, which are available through the respective stores.

Connecting the Windows client to Business Central is not supported in Business Central 2019 release wave 2 and onwards.

Why is Microsoft discontinuing the Windows client?

Our customers must feel comfortable that the tools they use are fit for new hardware, operating systems, and changing environments. We have accelerated our investment in speed and productivity features for the modern clients, thereby achieving a major milestone in its transformation into a world-class desktop experience for both new and expert users.

While the Windows client was inherently bound to the Windows operating system, the modern clients allow us to reach more customers and more users within an organization, no matter their platform or device of choice. The latest technologies allow us to innovate at a rapid pace and respond to accelerating compliance requirements, the changing technology landscape, and requests from the community. In addition, the lower installation footprint on client devices makes it easier for IT departments and hosters to maintain and support their user base.

When is the Windows client discontinued in Business Central?

From **October 2019**, with Business Central 2019 release wave 2.

It was first announced in 2018 at various conferences and then with a detailed timeline earlier in 2019. For more information, see Business Central April 2019 Update and the road ahead.

Will the Windows client still be supported in older releases of Business Central and Dynamics NAV?

Yes. You can safely continue to use the Windows client on premises and receive support as long as you follow the lifecycle policy for your on-premises installations of Business Central. For more information, see Lifecycle FAQ - Dynamics.

The Windows client **remains supported** for the Business Central April 2019 release and all earlier releases of Business Central on premises and Dynamics NAV, in accordance with the support lifecycle process.

Does this impact me if I use Business Central online?

No. This change only impacts on-premises installations because the Windows client was only available on premises.

Does this impact me if I use Business Central on premises?

Yes. When you **choose to upgrade** to Business Central 2019 release wave 2 or later, you must switch to access Business Central using one of the modern clients. The most popular choice on desktop computers is the web-browser client where your browser is pointing to an on-premises web server using a URL, such as this example (not active): `https://myserver.mydomain.com/BC170`

What if I really want to have an installable component or at least an icon on my desktop?

You can always add a browser shortcut on your desktop or pin the web page with Business Central to your Windows task bar. Alternatively, the Business Central Windows 10 desktop app, which is available from Microsoft Store, is a great way to access Business Central both online and on-premises. To get the app, go to Microsoft Dynamics 365 Business Central in the store.

How does this impact mobile?

There is **no impact** on mobile apps for Business Central as they are already part of the modern-client family. For more information about the mobile apps, seeGetting Business Central on Your Mobile Device.

Can I still work with Business Central data in Excel?

Yes. There are multiple ways to work with Business Central and Excel, including the following:

- The Open in Excel feature that downloads any list as an Excel file for your processing or reporting
- The Edit in Excel feature that allows you to edit almost any list-based data in Excel and publish it back to Business Central

For more information, see Viewing and Editing in Excel From Business Central. For instructions on how to configure it for on-premises, see Setting up the Excel Add-In for Editing Business Central Data.

Note that the legacy, COM-based Excel plugin that used to be included on the installation media is no longer supported.

Can I still use the same Outlook add-in?

Yes. This change does not impact the Outlook add-in. In fact, the modern Outlook add-in is based on the same familiar web experience. For more information, see Using Business Central as your Business Inbox in Outlook.

What happened to C/SIDE, the legacy development environment?

In line with the retirement of the Windows client, Business Central 2019 release wave 2 marks a milestone as the first release without the legacy development environment (also known as C/SIDE). The modern developer experience, which is based on Visual Studio Code and the new AL language, supports developing large apps, such as the base application from Microsoft.

Therefore, C/SIDE is discontinued for Business Central going forward. Partners enjoy tremendous productivity and performance gains after moving to the newest tools. For more information, see Development in AL.

Which features are available in the modern clients and where can I find the roadmap?

Business Central is a highly adaptable modern business management solution. It is rich in features and options and is continuously being enhanced. The roadmap is best represented by the release plans, which are updated every six months. For more information, see Overview of Dynamics 365 Business Central 2019 release wave 2.

See Also

FAQ for Developing in AL
Features not implemented in on-premises deployments of Dynamics 365 Business Central
Business Central Component and System Topology, Additional Components
Software Lifecycle Policy and Dynamics 365 Business Central On-Premises Updates
Dynamics 365 Business Central Compliance
FAQ for Dynamics 365 Update Policies
Dynamics 365 Resources
Welcome to Dynamics 365 Business Central

FAQ about Connecting to Business Central Online from On-Premises Solutions

2/17/2021 • 7 minutes to read • Edit Online

This section contains answers to frequently asked questions about connecting on-premises solutions to Business Central online.

> **TIP**
> Check the tips in this article if your organization is not yet ready to migrate to Business Central online but still wants to enjoy some of the benefits of the cloud. The same tips apply to when you are migrating to the cloud and are running the migration tool. For more information, see Running the Cloud Migration Tool.

Which products and versions are supported for this connection?

The current version of Business Central can connect the following products in order to provide intelligent insights:

- Dynamics GP (supported major versions)
- Business Central on-premises

  Currently, you can migrate to Business Central online from versions 14, 15, 16, and 17.

If you are currently on a version of Dynamics NAV, you must upgrade to Business Central on-premises, and then switch to Business Central online. For more information, see Upgrading from Dynamics NAV to Business Central online.

System requirements

To connect to the cloud through Business Central, the on-premises solution must use SQL Server 2016 or a later version, and the database must have compatibility level 130 or higher. The on-premises solution must also be one of the supported versions.

How is my on-premises data replicated to my Business Central online tenant?

Data is replicated using an Azure service called Azure Data Factory (ADF). ADF is a service that is always running within the Business Central online service manager. When you have connected to the intelligent cloud, a data pipeline is created in the ADF service so that data can flow from your on-premises solution to your Business Central online tenant. If your data source is a local SQL Server instance, you will also be asked to configure a self-hosted integration runtime (SHIR). The runtime is installed locally and manages the communication between the cloud services and your on-premises data without opening any ports or firewalls.

Are there any limits on the amount or type of data will replicate?

There are no restrictions on the type of data that can be replicated. In the current version of Business Central, the migration tool is by default limited to migrate databases up to 80 GB. If your database is larger than 80 GB, we recommend that you reduce the number of companies that you are migrating data for. You can specify which companies to include in the migration in the assisted setup wizard.

If you want to add more companies after the first selection of companies, you can add additional companies in the **Cloud Migration Management** page in Business Central online. For more information, see Adding a tenant to an existing runtime service, or updating companies.

If you are looking at migrating databases larger than 80 GB, we recommend that you contact the support team and work with them to make sure that the migration is successful.

Is my SQL connection string required to set up the connection?

Yes. The SQL connection string is passed to Azure Data Factory, where it is encrypted and delivered to your Self-Hosted Integration Runtime. The connection string is used to communicate with your SQL Server instance during the data replication process. For more information, see How do I find my SQL connection string?.

How do I find my SQL connection string?

Find the connection string to your SQL database in SQL Management Studio or Visual Studio. The user name and password defined in the connection requires a SQL Authenticated user name/password. Your connection string will look something like this:

Server=tcp:{ServerName},1433;Initial Catalog={DatabaseName};Persist Security Info=False; User ID={UserName};Password={Password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=True;Connection Timeout=30;

How do I find the Integration Runtime name?

Find the Integration Runtime name in the Microsoft Integration Runtime Manager, which you can find in your Windows system tray or by searching for the program. You must type the name. You will not be able to copy and paste the name.

I am a hosting partner - do I need to configure the Self-Hosted Runtime Service for each tenant?

No, there is no limit on the number of tenants that can be added to your Self-Hosted Integration Runtime. Each added tenant will have a dedicated pipeline created.

Will data from tables with code customizations replicate?

No, only tables that are available in both your on-premises solution and your Business Central online tenant will replicate. Any customization must be made into an extension and installed on both your on-premises solution and your Business Central online tenant to replicate.

Why are my permissions restricted in the Business Central online tenant?

When you connect your on-premises solution to Business Central online for intelligent insights, all existing users are automatically added to the *Intelligent Cloud* user group, unless they have the SUPER permission set. In this configuration, your on-premises solution is the master where all business transactions take place. The Business Central online environment is read-only, and the data is used to generate intelligent business insights based on your on-premises data for you. We restrict permissions to prevent users from accidentally entering transactions or updating master records only to have that information overwritten and lost when data replication takes place.

Can I 'turn off' my intelligent cloud?

You can switch off your connection to the Business Central online environment at any point. Once you disable

your intelligent cloud configuration, your on-premises solution and the Business Central online tenant will become independent of one another. If you switch off the connection, and you want to use your Business Central online environment as your primary solution to run and manage your business, you must reassign permissions to provide read/write access to the relevant users.

For more information, see Managing Users and Permissions.

Will my on-premises users and permissions replicate?

No. Since you are not required to configure your on-premises solution with Azure Active Directory (Azure AD), we cannot guarantee a mapping between on-premises users and users in your Business Central online tenant. Business Central online requires Azure AD accounts, and users must be manually added. All permissions must be granted in the Business Central tenant, independent from your on-premises permissions.

For more information, see Managing Users and Permissions.

Can I view insights from cloud services in my on-premises solution?

Yes, the **Intelligent Cloud Insights** page can be hosted within your on-premises solution if that is one of the currently supported solutions. Each user will need to have a Business Central license to view the data.

Can you export to Excel, modify the contents, and import the data back in?

You can export the list to Excel from the Business Central online tenant, but since the data is read-only you cannot make changes and import it again.

Is the data replication only one way?

Yes, data is only replicated from the on-premises solution to your Business Central online tenant.

Is there a cost to connect to the intelligent cloud?

Currently, the only costs associated with the intelligent cloud are your named user license costs. For more information, see the Business Central Licensing Guide (download).

Why did my Role Center change after configuring the intelligent cloud?

To keep the Role Center experience as clean as possible and avoid permission errors, we automatically hide actions that would generate a permission error for the user.

Should I uninstall all my Business Central extensions?

Not necessarily. Most extensions will run without issues in the online environment. You may want to consider uninstalling extensions that send data to an external service to avoid potential duplicated calls to that service. It is a best practice to test any extension in a sandbox tenant configured for the Business Central online environment that you are connecting to.

How do I build an extension that enables data replication?

The extension must be created in the same manner as any other extension. For data to replicate, you must add a **ReplicateData** property to your table and set the value to *True*. If your extension connects with an external service and you want to restrict any service calls from your Business Central online tenant, a good practice

would be to store the connection information in a separate table and set the **ReplicateData** property to *False*. This would enable you to keep the extension installed but prevent it from making any type of service calls from the read-only Business Central tenant. Once the extension is installed in Business Central online and on-premises, the data will begin to replicate.

See also

Running the Cloud Migration Tool
Migrating On-Premises Data to Business Central Online

Resources for Help and Support for Dynamics 365 Business Central

2/17/2021 • 8 minutes to read • Edit Online

As a Business Central partner, you have access to resources that can help you support your Business Central customers, and you have access to resources that can help you be more productive as a partner.

This page outlines the resources available to you.

Product Help

The functionality in the default version of Business Central is described on the Docs.microsoft.com site as described in the following table.

NAME	LOCATION	DESCRIPTION
Business functionality docs	/dynamics365/business-central	Use this library to learn about business functionality.
Development and administration docs	/dynamics365/business-central/dev-itpro/	Use this content to learn how to extend, customize, and administrate Business Central.

AL developer documentation

In the Development in AL section, you find descriptions of processes such as compilation and debugging, and conceptual information about object types such as tables and events.

The reference documentation of the AL language publishes under the AL Programming umbrella. This content is partly generated automatically from code, but currently most of the reference content is maintained by hand. Use the following landing pages to quickly find the reference content that you need:

- Methods
  - Data Types and Methods in AL
- Properties
  - Properties Overview
- Triggers
  - Triggers Overview
- Objects
  - Table object
  - Table extension object
  - Page object
  - Page extension object
  - Page customization object
  - Report object
  - XMLport object
  - Query object
  - Codeunit object
  - Profile object

- Control add-in object

Product versions and Help versions

In general, the Business Central content on the docs.microsoft.com site reflects the latest version of Business Central online with limited support for earlier versions.

If you support Business Central on-premises, your solution might be one or two versions older than the latest version. This means that the content in the business functionality documentation might describe functionality that your users do not have access to. For a better experience, we recommend that you take a copy of our content at the time when that reflected the version that your on-premises solution is based on and deploy that to your own website. For more information, see the On-premises deployments section in the Configure the Help Experience article.

Customize and extend the user assistance

If you customize or extend Business Central, you are expected to also customize the user assistance so that users will have access to content that can help them get started, get unblocked, and learn more. For more information, see User Assistance Model and Configure the Help Experience.

Support

As a Business Central reselling partner, you are an administrator of your customers' Business Central tenants, and you are the first line of support. You can customize the support experience, and you have access to information that can help you troubleshoot any issues that your customers report.

For more information, see Technical Support.

Training

You and your customers can find free eLearning content on the Microsoft Learn landing page for Business Central with suggested collections for getting started, managing financials, and extending Business Central.

The Learn site also includes role-specific learning paths for developers, administrators, and functional consultants.

Additionally, partners in the Business Central community offer training and books.

Resources

As a partner, you can keep on top of current and upcoming capabilities, and you can share Microsoft's roadmap with your prospects, for example. This section provides links to places to keep track of for people who are new to Business Central as well as for people who have been working with the product for years.

This article provides information about the following types of resources:

- How to get started as a partner
- Where to learn about current and upcoming capabilities
- How to make a feature request
- Where to find blog posts
- Where to find communities
- Where to file bugs and issues
- Where to contribute to code or docs
- Where to ask non-product related questions

Get set up as a partner

If you are not already a Microsoft partner, your company must get set up, and so must you as an employee. For

more information, see Get Started as a Reseller of Business Central Online and Get Started with Building Apps.

Learn about current or upcoming capabilities

You can learn about current and coming capabilities through a number of different resources as outlined in the following table.

NAME	LOCATION	DESCRIPTION
Release plans	https://docs.microsoft.com/dynamics365/release-plans/	Get an overview of upcoming and recently released capabilities in Business Central and other Dynamics 365 apps.
Business functionality	https://docs.microsoft.com/dynamics365/business-central/across-business-functionality	Use this content to learn about business functionality in the default version of Business Central.
Development in the AL language	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/developer	Use this content to learn how to extend and customize Business Central using the native AL language.
Administrative tasks	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/administration/	Use this content to learn how to administer Business Central online.
Development in the AL language	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/developer	Use this content to learn how to extend and customize Business Central.
Security in Business Central	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/security/	Use this content to help you understand and improve the security of Business Central.
Compliance	https://docs.microsoft.com/dynamics365/business-central/compliance/compliance-overview	Use this content to learn about compliance in relation to Business Central.
Performance	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/performance	Use this content to learn how to extend and customize Business Central.
Integrating with Business Central using web services	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/webservices/	Use this content to learn how to integrate Business Central with other products by using web services.
Migrate to Business Central online	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/upgrade/upgrading-to-business-central-online	Use this content to learn how you can move an on-premises solution to Business Central online.
Features not implemented in on-premises deployments	https://docs.microsoft.com/dynamics365/business-central/dev-itpro/features-not-implemented-on-premises	Use this content to learn which capabilities in Business Central online are not available in Business Central on-premises deployments.

NAME	LOCATION	DESCRIPTION
The Business Central learning catalog	https://docs.microsoft.com/dynamics365/business-central/readiness/readiness-learning-catalog	Get an overview of role-specific training material from Microsoft.
Free eLearning	https://docs.microsoft.com/learn/dynamics365/business-central	Free Business Central collections and learning paths on Microsoft Learn

Share an idea about a new feature you'd like to have

On the Dynamics 365 Ideas site, you can provide suggestions for new feature and capabilities. Your input goes directly to Business Central's engineering backlog for investigation and prioritization.

Make sure to search through the list of submitted suggestions, as chances are that someone already submitted something similar and might have already received votes. Vote if an idea already has been submitted to get it prioritized on the team's backlog.

Business Central blog posts

The Dynamics 365 blog is where Microsoft publishes announcements, updates, and tips and tricks, including for Business Central.

NAME	LOCATION	DESCRIPTION
Business Central on the Dynamics 365 blog	https://cloudblogs.microsoft.com/dynamics365/it/product/business-central/	Use this blog to learn about opportunities, processes, and tools for the Business Central partner community. Some older blog posts are still available on the Community site for partners or the Community site for business users.

Business Central communities

On the Business Central Community site, you have access to the Business Central Forum. You can use this forum to submit a question and learn from other Business Central community members. MVPs, partners, and Microsoft employees participate in the conversations.

You can also join the BCUG/NAVUG User Group for Dynamics 365 Business Central and Dynamics NAV, which is a user-led, user-driven community of more than 26,000 users, partners, and MVPs.

Summary of where to file bugs and issues

As a partner, you have different support channels depending on what type of issue you want support for. The following list outlines the various channels.

ISSUE TYPE	SITE
Submit support request on behalf of your Business Central online customers	Start at the Business Central administration center where you can easily submit a support request in the Power Platform admin center
Report bug in a preview or beta version	The MS Collaborate site
Collaboration on the AL language and developer experience	The AL Developer Preview GitHub repo

ISSUE TYPE	SITE
As an ISV, report an issue in production code, such as a problem with Microsoft's application, upgrade, or telemetry	The Partner Center - choose the **Support** section, start a partner request, search for *Business Central*, and then choose the relevant category for *Product Support* > *Business Central Development* and submit your support request.
Report bug in supported in-market versions of Business Central on-premises	The Support for business site

> **IMPORTANT**
> To submit support requests on behalf of your customer, you must be a delegated admin on the customer tenant. If your company is a CSP direct bill partner, you must have *Advanced* or *Premier* support plan.

For more information, see Technical Support for Dynamics 365 Business Central.

Engage with us on GitHub

GitHub brings together communities of developers and other contributors to discover, share, and build software. Here are some useful repositories for Business Central:

- Microsoft AL

  For collaboration on the AL development environment in Visual Studio Code, use the GitHub repo here: https://github.com/microsoft/al

- Microsoft AL application add-ons

  There are a couple of ways to engage with us in the https://github.com/microsoft/ALAppExtensions repo:

  - You can grab the code and contribute to the published apps.
  - If you're building your own app and need something specific from us, such as an event, you can help improve the general extensibility of the business logic.

- NAV Docker

  Use this repo to collaborate around the source code and the scripts of the generic docker image for Business Central: https://github.com/Microsoft/nav-docker

- BcContainerHelper/NavContainerHelper

  Use this repo to collaborate around the source code and the scripts of BcContainerHelper PowerShell module for Business Central: https://github.com/Microsoft/NavContainerHelper

- Documentation

  The product Help and developer and administration content is based on GitHub as well, and you can collaborate and extend the docs. The GitHub repos are:

  - Product Help (English US): https://github.com/MicrosoftDocs/dynamics365smb-docs
  - Developer and administration content: https://github.com/MicrosoftDocs/dynamics365smb-devitpro-pb

  For more information, see Extend, Customize, and Collaborate on the Help.

Non-product related questions

On occasion, as a partner, you will run into questions that are not directly related to the product. The following list outlines how to get answers to those questions.

FOR QUESTIONS RELATED TO	CONTACT
Problems with listing or publishing apps to AppSource due to Commercial Marketplace issues	The Partner Center - choose the **Support** section, start a partner request, search for *Business Central*, and then choose the relevant category for *Commercial Marketplace* and submit your support request.
Licensing or PSBC agreements	Email MBS Orders or MBS Agreements, respectively
Microsoft Partner Network, Partner Center, Cloud Solution Provider program	The Partner Center Support site
Payments, credit terms, checks, wire, or similar	Email MBS Accounting
Technical issues with PSBC, PartnerSource, or Order Central	Email IT MBS Support
Incentives	Email CSA Team
Cloud Solution Provider incentives	Email Online Channel Incentives Support
CSA/Ocina escalations	Email NAOC Channel Incentives Escalations
Volume licensing	The Call Logging Tool site or email Online Licensing

Trials

Giving prospects access to a pre-configured trial of Business Central is an elegant way to introduce them to Business Central. You can use the standard trial provided by Microsoft, or you can prepare your own including relevant extensions.

For more information, see Preparing Demonstration Environments.

See also

Technical Support
Configuring the Help Experience
Migrate Legacy Help

The Business Central Administration Center

Legal Landing Page for Microsoft Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This page provides links to legal information for Business Central.

Microsoft Online Service Terms (OST)

- Online Service Terms(OST)

Microsoft AL Language

- AL Language - Terms of Use
- AL Extension Third Party Notice

Envestnet Yodlee - Bank Feeds

- Envestnet Yodlee - Bank Feeds Terms and Conditions

Trial Services terms

- Trial Services terms

See Also

Privacy FAQ
Legal Resources for Business Central On-Premises
Developer and Administration Help for Microsoft Dynamics 365 Business Central

Technical Support for Dynamics 365 Business Central

2/17/2021 • 4 minutes to read • Edit Online

Each customer of Business Central has a partner who assists with technical support when requested by the internal administrator. As a Business Central reselling partner, you are an administrator of your customers' Business Central tenants, and you are the first line of support. This means that you will get requests for support from your customers that you must triage, investigate, and either resolve or escalate to Microsoft.

In this section, you can learn about the tools that are available to you to help you troubleshoot your customers' Business Central.

Configuring the support experience

Because you are the first line of support for your customers, you must make it easy for them to contact you. To that end, there is a section in the Help and Support page in your customers' Business Central tenants where they can find this information.

> **IMPORTANT**
>
> You must have set up users in your own tenant in Partner Center as either *Admin agent* or *Helpdesk agent*, and they must have *delegated administration* privileges in your customer's Business Central to support the customer. For more information, see Delegated Administrator Access to Business Central Online.

To supply your support contact information in the administration center

1. In the Business Central administration center, choose the environment that you want to specify your contact details for, such as *Production*, and then, in the **Support** menu, choose **Manage Support Contact**.
2. Fill in the **Name**, **Email address**, and the **Website** fields, so that your users know how to contact you for technical support.
3. Optionally, choose the **Apply to all environments** checkbox if you want to add the same details to all related environments for this tenant.

Your customer can now contact you if they experience problems that they cannot resolve themselves. If you also cannot resolve a reported issue, you can escalate the issue to Microsoft. For more information, see Managing Technical Support.

On-premises deployments

In on-premises deployments of Business Central, the **Help and Support** page does not contain the section for contacting technical support. Instead, you can enter an agreement with your customer's administrator about how and when to contact you.

There are two other links in the **Help and Support** page that you can customize:

- Blog

  Specifies a link to where your customers can access a blog about their solution

- Coming soon

  Specifies a link to where your customers can access a roadmap for future changes

If you choose to not modify these settings, then the links go to Microsoft's blog and release notes.

For more information, see Configuring Business Central Web Server Instances.

Getting support for extension issues

As a partner, you must identify if the issue is caused by application logic or platform behavior:

- If the issue is caused by application logic, you must identify the publisher of the extension.
- If the extension is a per-tenant extension, as a partner, you must fix the issue.
- If the extension is an AppSource extension, you must contact the AppSource partner that developed the extension.
- If the extension is published by Microsoft, you must contact Microsoft support.

Troubleshooting and support

The Business Central administration center is your primary tool to support your customers. However, you can also log in to the customer's Business Central as the delegated admin for troubleshooting.

For more information, see Managing Technical Support.

Summary of where to file bugs and issues

As a partner, you have different support channels depending on what type of issue you want support for. The following list outlines the various channels.

ISSUE TYPE	SITE
Submit support request on behalf of your Business Central online customers	Start at the Business Central administration center where you can easily submit a support request in the Power Platform admin center
Report bug in a preview or beta version	The MS Collaborate site
Collaboration on the AL language and developer experience	The AL Developer Preview GitHub repo
As an ISV, report an issue in production code, such as a problem with Microsoft's application, upgrade, or telemetry	The Partner Center - choose the **Support** section, start a partner request, search for *Business Central*, and then choose the relevant category for *Product Support > Business Central Development* and submit your support request.
Report bug in supported in-market versions of Business Central on-premises	The Support for business site

IMPORTANT

To submit support requests on behalf of your customer, you must be a delegated admin on the customer tenant. If your company is a CSP direct bill partner, you must have *Advanced* or *Premier* support plan.

TIP

When you submit your first support ticket as a partner, you must specify details about your company's support plan. If you or your colleagues do not know these details, contact your Microsoft rep.

Non-product related questions

On occasion, as a partner, you will run into questions that are not directly related to the product. The following list outlines how to get answers to those questions.

FOR QUESTIONS RELATED TO	CONTACT
Problems with listing or publishing apps to AppSource due to Commercial Marketplace issues	The Partner Center - choose the **Support** section, start a partner request, search for *Business Central*, and then choose the relevant category for *Commercial Marketplace* and submit your support request.
Licensing or PSBC agreements	Email MBS Orders or MBS Agreements, respectively
Microsoft Partner Network, Partner Center, Cloud Solution Provider program	The Partner Center Support site
Payments, credit terms, checks, wire, or similar	Email MBS Accounting
Technical issues with PSBC, PartnerSource, or Order Central	Email IT MBS Support
Incentives	Email CSA Team
Cloud Solution Provider incentives	Email Online Channel Incentives Support
CSA/Ocina escalations	Email NAOC Channel Incentives Escalations
Volume licensing	The Call Logging Tool site or email Online Licensing

See Also

Inspecting and Troubleshooting Pages
The Business Central Administration Center
Managing Technical Support
Deployment Overview
Administration of Business Central Online
Administration of Business Central On-Premises
Provide technical support (Microsoft Partner Center)
Providing support to your customers (Microsoft Partner Center)

Dynamics 365 Business Central User Assistance Model

2/17/2021 • 7 minutes to read • Edit Online

The Business Central user assistance model is based on the following principles:

- Get started

  Default values and setup wizards makes it easy to start using Business Central with your own data, in-product videos give new users a quick introduction to how the product works, and Home pages give easy access to key tasks so users can easily get started with work every day.

- Get unblocked

  Embedded user assistance implemented as tooltips answers most immediate questions about what fields and actions do.

- Learn more

  The Help menu and the tooltips provide context-sensitive links to Help articles with more information.

Apps, extensions, and customizations are expected to follow the same model by applying tooltips to controls on page objects, and by providing links to Help for their functionality. For more information about customizing and extending the user assistance, see Extend, Customize, and Collaborate on the Help and Configure the Help Experience.

In this article, we'll talk about the user assistance model itself and what it does.

Help users get started

The user assistance concept of *Get Started* is not just about getting started with Business Central on the first day. It's also about getting started all the other days, and about getting started with infrequent and unfamiliar tasks.

Assistance in the shape of wizards is helpful for set up or filling in data for a complicated report, for example. Home pages that are designed for a particular role or job help users get started with their daily work – they can easily get to their most important tasks, and that means that Business Central helps them get their work done more efficiently.

Help users get unblocked

Even the best designed user interface can still be confusing to some. It can be difficult to predict what users will find confusing, and that is why the base application includes descriptions for all controls and actions that can be accessed when you choose the caption of the control or action. In combination with descriptive captions and instructional text, these tooltips, or callouts, are our current implementation of *embedded user assistance*, which is an important principle in today's world of software design.

The tooltips help users unblock themselves by providing an answer to the most likely questions the users might have, such as "What data can I input here?" or "What is the data used for?". Keep tooltips in mind when you develop the user interface of your solution.

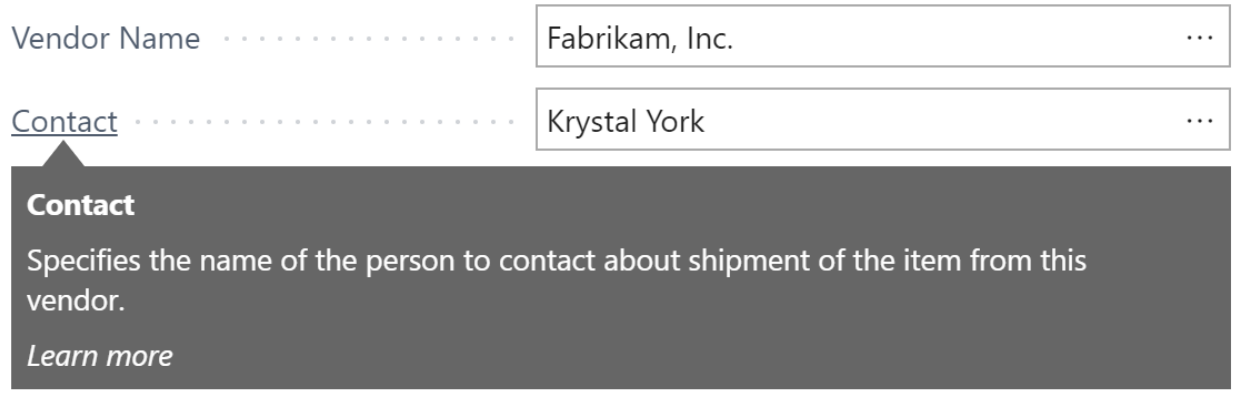
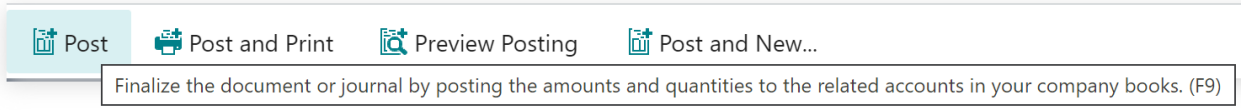


Table fields can be read-only in one page and editable in another so the tooltips describe the difference. In Business Central, this type of "What is this field?"-content is embedded in the page objects, and resource files can be used for translated user interfaces. For more information, see Working with Translation Files.

Most tooltips end with an automatically generated link to *learn more* as described in the Help users learn more section; however, tooltips for actions do not have *Learn more* links:



Users can always use the Ctrl+F1 keyboard shortcut to access the *learn more* content that is configured for the currently-selected item in the user interface.

The tooltips in Business Central are conceptually similar to *field descriptions* in Dynamics 365 Finance and related apps, and *flyouts* in the Fluent Design guidelines. Business Central does not have the equivalent of teaching tips that you can add to your solution, though our platform includes a few flyouts of the same type.

Guidelines for tooltip text

The Microsoft user assistance model requires a tooltip for all controls of type Action and Field that exist on page objects. Follow these guidelines:

- If the control is a field, begin with the verb *Specifies*.
- If the control is an action, begin with a verb in the imperative form , such as *Calculate* or *View*.
- Include the most valuable information that users need to perform the task(s) that the field or action supports.

  For example, for the **Post** action, do not write *Post the document*. Write, for example, *Update ledgers with the amounts and quantities on the document or journal lines*.
- Describe complex options in tooltips for option fields.

  Use a colon to call out the option name and its description. See example 3 below.
- Try to not exceed 200 characters including spaces.

  This makes the tooltip easier to scan so the user can get unblocked quickly. However, the UI will render longer tooltip text if you want to provide more detailed user assistance.
- Do not use line breaks in the tooltip text.

  The tooltip cannot render formatting or line breaks.

Examples:

CONTROL NAME	TOOLTIP
Password field	Specifies your company's password to the service that converts bank data. The password that you enter in this field must be the same as on the service provider's sign-on page. (175 characters including spaces)
Entries action	View the history of transactions that have been posted for the customer. (72 characters including spaces)
Account Type field	Specifies the purpose of the account. Total: Used to total a series of balances on accounts from many different account groupings. To use Total, leave this field blank. Begin-Total: A marker for the beginning of a series of accounts to be totaled that ends with an End-Total account. End-Total: A total of a series of accounts that starts with the preceding Begin-Total account. The total is defined in the Totaling field. (522 characters including spaces)

Help users learn more

The content that Microsoft publishes under the user assistance concept of *Learn more* is intended to answer those questions that the user interface (including the tooltips) cannot answer, such as where that page fits into the bigger workflow, or what comes next, or what would be the alternative. Users can access this content either through the *Learn more* link in tooltips, or by using the *Ctrl+F1* keyboard shortcut. For more information, see Configure Context-Sensitive Help.

The base version of Business Central uses content that is published to an online library (Docs.microsoft.com/dynamics365/business-central) so that it can also serve as onboarding material and as feature overviews that you can share with prospects. The content is written in MarkDown, and our source files are available in a public GitHub repo that you can extend and customize for your customers.

There are repos in GitHub for the source content and each of the languages that Microsoft translates to. For more information, see Extend, Customize, and Collaborate on the Help.

For the base version of Business Central, free online learning is also available on Microsoft Learn. For more information, see the Business Central Learning Catalog.

Feedback and contributions

On docs.microsoft.com, each article has two buttons at the end of the article. The *Product feedback* button sends you to the Ideas site, and the *Sign in to give documentation feedback* button lets you submit feedback about the content through GitHub. In both cases, you must create an account if you do not already have one. For *product feedback*, you must sign in with your work or organizational email account. For *access to GitHub*, you can use any email address when you create an account.

We welcome your contributions, both as pull requests with suggestions or corrections to the content, and as GitHub Issues with bugs or questions. However, we can only accept feedback and contributions to the content in the *dynamics365smb-docs* repo and we cannot address issues or questions about the product.

> **IMPORTANT**
> Microsoft only accepts pull requests to the *dynamics365smb-docs* repo, not the language-specific repos. If you have feedback about translations, you can report a GitHub issue in the relevant repo.

Microsoft also accepts contributions and feedback about the development and administration content through

the dynamics365smb-devitpro-pb. This repo does not have translation repos associated with it, but other than that, the same rules apply as for the *dynamics365smb-docs* repo.

For more information, see Extend, Customize, and Collaborate on the Help.

Working in MarkDown

If you fork one of our repos, you will be authoring in MarkDown. We recommend that you learn the basics by referring to the Docs Markdown reference section in the Docs Contributor Guide.

The Docs Authoring Pack for VS Code can aid with MarkDown authoring and validation in Visual Studio Code. However, you can also use other text editors to edit MarkDown.

For other tips and tricks, see Extend, Customize, and Collaborate on the Help and Blog post: Collaborate on content for Business Central.

Translate the Help

If you want to deliver a localization app, or if you want to deliver your functionality in more than one country, you will want to translate the Help. We suggest that you take a look at the Microsoft Dynamics 365 Translation Service, which is available as preview in Microsoft Dynamics Life Cycle Services. For more information, see Translate documentation files.

The user interface text, including the tooltips, is translated as part of the application. For more information, see Working with Translation Files.

Style

At Microsoft, we are in process of simplifying and unifying our style guides. To get to know the Microsoft style, use the Microsoft Writing Style Guide as a good starting point. The Business Central follows most of the guidelines in the Microsoft Writing Style Guide with exceptions for industry terminology and other product-specific issues.

See Also

Configure the Help Experience
Adding Help Links from Pages, Reports, and XMLports
ToolTip Property
InstructionalText Property
Development of a Localization Solution
Translate documentation files
Resources for Help and Support
Blog post: Extending and customizing the Help
Blog post: Collaborate on content for Business Central
Docs Contributor Guide
Docs Authoring Pack for Visual Studio Code
Style Guide for Microsoft Dynamics NAV (requires login)
Microsoft Cloud Style Guide (requires login)

Configuring the Help Experience for Dynamics 365 Business Central

2/17/2021 • 6 minutes to read • Edit Online

The default version of Business Central comes with conceptual overviews and other articles that publish to the https://docs.microsoft.com/dynamics365/business-central/ site. This location is accessible from the Help menu and through the Learn More links in all tooltips. Each extension that you add will include its own tooltips and links to Help.

But what if you want to deploy Business Central locally? Or if you have a vertical solution so that you want to refer your customers to your own website for Help? Or if you have a legacy Help collection based on the Dynamics NAV Help Server? These and other scenarios are also supported in Business Central.

Apps for online tenants

When you build an app for Business Central using the AL developer experience, you are expected to comply with the Business Central user assistance model. The user assistance model requires the use of tooltips and context-sensitive links to Help content that is hosted on a website. For more information, see the Deploy content to your website section and the Configure Context-Sensitive Help article.

> **TIP**
> The website does not have to be publicly accessible, but it must be accessible to all users of the solution that it supports.

You can add Microsoft's content to your website, or you can deploy just your own content. The choice is yours and depends on the requirements of your users, the size of your app, and the amount of customization you want to make. The custom Help toolkit includes tools that can help you prepare and deploy content. For more information, see Custom Help Toolkit.

On-premises deployments

For deploying Business Central on-premises, you can choose between using the legacy Dynamics NAV Help Server and an online website, and you can configure different Help experience for each Business Central Web Server instance. Help Server is a simple website that requires your Help to be in a specific format (HTML files), while the online website can host any content that you want to make available. Your choice depends on the needs of your solution and your users. If you add configuration for an online library, you must remove any settings for Help Server.

> **IMPORTANT**
> The legacy Dynamics NAV Help Server component will be deprecated. We recommend that you invest in a different type of website. For more information, see the 2020 release wave 2 release plan and Custom Help Toolkit.

TIP

The content on the https://docs.microsoft.com/dynamics365/business-central/ site and in the various GitHub repos reflects the latest version of Business Central, unless otherwise specified.

We recommend that you get your version of Microsoft's content close to the time the subsequent major version of Business Central becomes available. For example, if you are deploying version 16.4, you should have taken a snapshot of the content in GitHub before version 17.0 became available.

Online library

To display content from an website that hosts your user assistance content, specify the URL in the settings for the Business Central Web Server. The navsettings.json file must contain the following setting in the `ApplicationIdSettings` element:

```
{
  "NAVWebSettings": {
    // [...more keys]
  },
  "ApplicationIdSettings": {
    //BaseHelpUrl:  The location of Help for this application.,
    "BaseHelpUrl": "https://mysite.com/{0}/documentation/",
     // [...more keys]
  }
}
```

NOTE

Replace the value of the BaseHelpUrl key with the URL for your own website. The parameter, {0}, represents the locale of the browser that the user is using, such as en-us or da-dk, and is set automatically at runtime.

For more information, see Configuring Business Central Web Server Instances.

TIP

The website does not have to be publicly accessible, but it must be accessible to all users of the solution that it supports.

Help Server

If you want to use Help Server, then you must specify the server and port in the installation options. The Help Server website can also serve as a starting point for adding a library to your existing website, for example.

The navsettings.json file must contain the following settings in the `NAVWebSettings` element:

```
{
  "NAVWebSettings": {
    //HelpServer: Name of the Dynamics NAV Help Server to connect to.,
    "HelpServer": "https://myserver.com",
    //HelpServerPort:  The listening TCP port for the Dynamics NAV Help Server. Valid range: 1-65535,
    "HelpServerPort": "49000",
    // [...more keys]
  },
  "ApplicationIdSettings": {
    // [...more keys]
  }
}
```

In the example, `https://myserver.com` represents the URL to the Help Server instance. For more information, see Configuring Microsoft Dynamics NAV Help Server in the developer and adminstration content for Dynamics NAV.

> **IMPORTANT**
>
> If you use Help Server, the UI-to-Help mapping functionality that is described in Configure Context-Sensitive Help does not work. Neither does the original Help lookup mechanism that was based on filenames that reflected the object IDs, such as N_123.htm for the page object with the ID 123. For more information, see Blog post: Reusing classic object-based Help on your Dynamics 365 Business Central Help Server.

> **TIP**
>
> If you are upgrading from Microsoft Dynamics NAV 2017, you can reuse your existing Help Server content by simply replacing the product name and make any other changes that apply to your Business Central environment.

You can also still download the files that were made available for Microsoft Dynamics NAV 2017. The download consists of 45 CAB files with the content from the Dynamics NAV 2016 DVD rebranded to Microsoft Dynamics NAV 2017. There are CAB files with the W1 application Help translated into each of the supported languages plus the local functionality for the country/region where that language is spoken. There are also CAB files with local functionality in English. The files were published as a single download so each administrator could choose exactly the files that they needed at the time. For more information, see Microsoft Dynamics NAV 2017 Classic Help Download.

> **IMPORTANT**
>
> Specifically for the preview of Business Central in India, the installation of Help Server fails due to missing files on the installation media. The solution is to install Help Server without the HTML files for local functionality and instead pick up the content from GitHub. For more information, see Get updates from Microsoft.

Deploy content to your website

Business Central has no firm requirements for the website that hosts your online library for your Business Central online or on-premises. You can deploy your content using any tool and process, such as Azure Static Web Apps, Azure App Service, a website that can render MarkDown files using a customization of the DocFx Flavored MarkDown engine, or third-party services such as MkDocs.

You can see an example of how to deploy content to an Azure web app in the article Deploy custom help to Azure, which supports the custom Help toolkit for Dynamics 365 Finance, Dynamics 365 Supply Chain Management, and Dynamics 365 Commerce. That article also describes how you can build a search service for your website. This is currently not relevant for Business Central, but you might find the guidance helpful anyway.

Use the HtmlFromRepoGenerator tool tool in the custom Help toolkit to clone a repo and generate the corresponding HTML files automatically. If, instead, you want to create your own tooling and processes around DocFx, which is an open-source tool for converting markdown files, you can see examples in the Build HTML files section of the contributor guide.

Fork the Microsoft repos, and customize or extend the content

If you want to customize or extend the Microsoft Help, you can fork our public repo for either the source repo in English (US) at https://github.com/MicrosoftDocs/dynamics365smb-docs, or one of the repos that contain translations. For more information, see Extend, Customize, and Collaborate on the Help and Custom Help Toolkit.

See Also

User Assistance Model
Adding Help Links from Pages, Reports, and XMLports
Working with Dynamics NAV Help Server
Configuring Microsoft Dynamics NAV Help Server
Migrate Legacy Help to the Business Central Format
Building Your First Sample Extension With Extension Objects, Install Code, and Upgrade Code
Building an Advanced Sample Extension
Development of a Localization Solution
System Requirements
Resources for Help and Support
Blog post: Extending and customizing the Help
Blog post: Collaborate on content for Business Central
Blog post: Reusing classic object-based Help on your Dynamics 365 Business Central Help Server
Docs Contributor Guide
Docs Authoring Pack for Visual Studio Code

Configure Context-Sensitive Help

2/17/2021 • 6 minutes to read • Edit Online

A key pillar of helping users help themselves is to give them access to Help for the particular part of Business Central that they are working in.

App-level configuration

Specify where the Help for your functionality is published in the *contextSensitiveHelpUrl* property in the app.json file. For example, if you publish your content to `https://mysite.com/documentation`:

```
"contextSensitiveHelpUrl": "https://mysite.com/documentation/",
```

In this example, when the user is using your app's functionality, the *contextSensitiveHelpUrl* property specifies that the links to Help will go to the *mysite.com* site. When the user is using functionality from the base application, then the Help calls will go to the *docs.microsoft.com* site.

If your app only supports a limited number of locales, you can specify that as well as shown in the following example:

```
"contextSensitiveHelpUrl": "https://mysite.com/{0}/documentation/",
"supportedLocales": [
  "en-GB", "en-IE
],
```

The *contextSensitiveHelpUrl* and *supportedLocales* properties specify that the links to the Help for page objects in this app must go to the *mysite.com* site, but that the site only supports those two languages. All other Help calls from objects in this app will go to the default locale on the specified webserver, in this case the equivalent of `https://mysite.com/en-GB/documentation/my-feature`.

Help calls for Microsoft objects will continue to go to the *docs.microsoft.com* site.

Localization apps

Specifically for localization apps that translate Business Central into languages that are not offered by Microsoft, the app.json file must be set to specify the destination of links to Help as shown in the following example:

```
"helpBaseUrl": "https://mysite.com/{0}/documentation/",
"supportedLocales": [
  "ca-es"
],
```

The *helpBaseUrl* and *supportedLocales* properties specify that the links to the Help must go to the *mysite.com* site when the user is using the product in Catalan. If the user switches the application language to English (US), then the Help calls will go to the *docs.microsoft.com* site.

Page-level configuration

Your target website is expected to have a default page that will display if nothing else is specified. But for each page or page extension in your app, you can specify the Help page that describes this page or the fields that the

page extension adds to the page. You can do that using the *ContextSensitiveHelpPage* property as shown in the following example:

```
page 50101 "Reward Card"
{
    PageType = Card;
    SourceTable = Reward;
    ContextSensitiveHelpPage = 'sales-rewards';

}
```

In this example, the app contains a page object that is mapped to the *sales-rewards* Help file on the website that the app.json specifies. As a result, the *Learn more* link in the tooltips for this page will go to the equivalent of `https://mysite.com/documentation/sales-rewards`.

Similarly, the following code example shows a page extension object that sets the *ContextSensitiveHelpPage* property so that the *Learn more* link in tooltips for the fields that this page extension adds to the Customer Card will go to the `https://mysite.com/documentation/sales-rewards` rather than the default location at docs.microsoft.com:

```
pageextension 50104 "Customer Card Ext" extends "Customer Card"
{
    ContextSensitiveHelpPage = 'sales-rewards';
    layout
    {...}
}
```

You can use the ContextSensitiveHelpPage property to direct Help calls from multiple page objects or actions to the same article. For example, Microsoft has chosen to group the context-sensitive links depending on the granularity of the Help for specific area in the base application. If the Help for a specific area is made more granular, then the context-sensitive Help mapping is updated accordingly.

Your target website is expected to have a default page that will display if no other page is appropriate. For every page where *ContextSensitiveHelpPage* is not set, this default Help page will be shown.

For page extensions, the value of the *ContextSensitiveHelpPage* property will apply only to the controls that the page extension adds to the extended page objects. For example, if your page extension adds two new controls to the base application's Customer Card page, then the *Learn more* links in the tooltips for those two controls will go to the Help page that you have specified. The *Learn more* links in the rest of the controls will go to the default Help that is specified in the base application. This way, multiple apps can extend the same page object and each apply their own content-sensitive Help link without overwriting the context-sensitive links for other apps.

> **NOTE**
>
> The app.json file also contains a *help* property that is used by AppSource to specify the link that describes the app or solution.

How it works for the base application

In the current version of Business Central, the context-sensitive links to Help for the base application is based on a different UI-to-Help mapping that is stored in table 2000000198 **Page Documentation**. In this table, all page objects in the default version of Business Central are listed, and have a target Help article associated with each of them. Multiple page objects can be associated with the same Help article, such as when a specific workflow involves multiple pages.

The base URL to the location of the target articles that are listed in table 2000000198 **Page Documentation** is

specified at the app level as https://docs.microsoft.com/dynamics365/business-central/. In an extension, you can overrule this URL so that all calls for Help go to your site instead. This is especially important for localization apps where all context-sensitive Help calls for that app's language must go to the provider's website. For more information, see Configuring the Help Experience.

Adding page-level UI-to-Help mapping to the system table

You can run a script that populates the **Page Documentation** table with a mapping for Microsoft's page objects and your own page objects. This is useful if you want to reuse legacy Dynamics NAV Help for your Business Central on-premises deployment.

Caution

While it is possible to reuse the Dynamics NAV legacy Help with the legacy Dynamics NAV Help Server, and to populate the system table, **Page Documentation**, we recommend that you convert any existing content to the Business Central format, and that you fork our GitHub repos. For more information, see Extend, Customize, and Collaborate on the Help for Dynamics 365 Business Central and Migrate Legacy Help to the Dynamics 365 Business Central Format.

In the following example, you have chosen not to apply context-sensitive Help links to your page objects and instead you want to overwrite the UI-to-Help mapping that Microsoft has made in the system table.

Let's assume that the current mapping is:

PAGE ID	PAGE NAME	REGION/COUNTRY	RELATIVE PATH
4	Payment Terms	W1	sales-manage-sales
11300	Financial Journal	BE	how-to-create-financial-journals

You want to replace the values of the fields in the **Relative Path** column with classic page-level Help files:

PAGE ID	PAGE NAME	REGION/COUNTRY	RELATIVE PATH
4	Payment Terms	W1	N_4
11300	Financial Journal	BE	N_11300

Once you have done this mapping, you can apply it to the **Page Documentation** table by using a script that updates the table in the SQL Server database, for example.

You can find a couple of suggestions for how to go about this in our blog post, Blog post: Reusing classic object-based Help on your Dynamics 365 Business Central Help Server.

See also

User Assistance Model
Resources for Help and Support for Dynamics 365 Business Central
Adding Help Links from Pages, Reports, and XMLports
Migrate Legacy Help to the Business Central Format
Building Your First Sample Extension With Extension Objects, Install Code, and Upgrade Code
Building an Advanced Sample Extension
Development of a Localization Solution
JSON Files
Blog post: Extending and customizing the Help
Blog post: Collaborate on content for Business Central

Blog post: Reusing classic object-based Help on your Dynamics 365 Business Central Help Server
Docs Contributor Guide
Docs Authoring Pack for Visual Studio Code

Migrate Legacy Help to the Dynamics 365 Business Central Format

2/17/2021 • 5 minutes to read • Edit Online

Business Central implements a user assistance model where tooltips explain all fields and actions, and articles with conceptual descriptions of functionality are published to a website. If you are building an app, you are expected to comply with this model. However, there are many ways in which you can migrate and reuse your existing Help within this model.

Reusing existing web content

When you move to Business Central, you can reuse your existing product Help solution in most situations, especially if the content is already published to an *online library*, which is an internal or external website. In that case, all you have to do is to add that website to the configuration of Business Central online or on-premises. For more information, see Configuring the Help Experience.

More specifically, if you have content that you created for Dynamics NAV, then you can choose to reuse that content for your Business Central solution.

For example, you have a Dynamics NAV Help Server website with HTML files that describe your solution according to the Microsoft Dynamics NAV 2013 R2 documentation model and format. You can reuse the Help Server website and rebrand the website and the content. You then connect your Business Central solution with the Help Server website. For more information, see Configuring the Help Experience.

Business Central does not support the field-based approach to context-sensitive Help that Dynamics NAV 2017 and earlier versions use. Instead, you must use the Business Central page-based approach to context-sensitive Help. You do not have to convert your existing Help, but you do need to make the content available. For more information, see the Converting existing content section.

> **NOTE**
> For apps for Business Central online, you must apply tooltips to controls and actions in both page objects and page extensions, and you must supply context-sensitive links. For more information, see Configuring the Help Experience.

Converting existing content

If your existing content is in a different format, such as PDF files, Word documents, or printed manuals, you must decide if you want to keep the content as-is, or if you want to convert it to a format that can be accessed from inside Business Central. There are third-party tools available that can help you migrate your content, depending on the current format and the target format.

For example, if you are migrating your solution from Dynamics GP, you might have content in PDF files. You can choose to convert the content to MarkDown as described in the Moving to MarkDown section, and then publish to a new online library on your current website.

Migrating from Dynamics NAV

If you are migrating your solution from Microsoft Dynamics NAV 2013 R2 or later versions of Dynamics NAV, then you most likely have been using the Dynamics NAV Help Server, and your Help content is in HTML format. That means that you can reuse your existing content as-is, you can use the Custom Help Toolkit to get new HTML files to supplement your existing content, or you can use publicly available third-party solutions to

convert some or all of your HTML files to MarkDown if you want to follow similar processes to the ones the Microsoft team follows. For more information, see the Moving to MarkDown section.

If you are migrating from an earlier version of Dynamics NAV, then you can choose to first migrate to the Microsoft Dynamics NAV 2013 R2 format, and then migrate again to MarkDown. For more information, see Upgrading Your Existing Help Content in the legacy docs for Microsoft Dynamics NAV 2013 R2.

Converting legacy Dynamics NAV field Help to tooltips

Tooltips play an important role as part of the Business Central user assistance model, and we encourage you to apply tooltips to your controls and actions as well.

If you would like to reuse text from your Dynamics NAV Help for table fields, the FieldTopicTextExtractor tool can extract the first paragraph from your Dynamics NAV Help topics for table fields. The resulting spreadsheet will help you copy and paste the text from your Dynamics NAV Help topics into the ToolTip property of your controls.

Mapping the table fields to page controls is not always straightforward because the same table is often used by two or more pages. As a result, the page ID can be many numbers away from the table ID. It requires knowledge of the actual solution to determine which tooltips are relevant for which pages. The **FieldTopicTextExtractor** tool generates an Excel file that you can use for this purpose, since you can sort and filter, and then copy content in Excel.

In the base application, tooltips are in the page objects, so, at Microsoft, we edit code to edit tooltips. You do not have to do the same. You can choose to work with tooltips in the translation files or straight in the .AL files. Different solutions require different processes, so pick the process that is more efficient for you. We chose to associate the "What is this field?"-content with the user interface, meaning the controls on page objects. In the Dynamics NAV Help model, we chose a different approach, focusing on the database structure. Both approaches have their advantages and disadvantages, but the Business Central user assistance model currently focuses on the user interface with tooltips on page objects. For more information, see User Assistance Model.

For more information, see Working with Application Objects as Text Files in the docs for Microsoft Dynamics NAV 2016, How to Add Translated Strings By Importing and Exporting Multilanguage Files in the docs for Microsoft Dynamics NAV 2018, and Working with Translation Files for Business Central.

Moving to MarkDown

Converting your existing content to MarkDown can be done using third-party tools, including but not limited to PanDoc or the Writage plugin for Word.

Once you have converted your content to MarkDown, we recommend storing your content in a repository, such as a Git repo in Azure DevOps, a private or public repo in GitHub, or a project in MkDocs. You can use open-source tools such as DocFx to generate content for your website. In general, working in MarkDown means that you have access to a world of open-source tools and do not have a hard dependency on Microsoft providing you with tools. The Custom Help Toolkit can help you prepare content for publishing. For more information, see Custom Help Toolkit.

If you do not yet have a website that you publish content to, then there are several ways in which you can create such a site. The MkDocs project generates a website for you, but you can also work with a web designer to build a site to host your content. We recommend deploying to an Azure web app.

See Also

Configuring the Help Experience
Extend, Customize, and Collaborate on the Help
User Assistance Model
Development of a Localization Solution

Custom Help Toolkit

2/17/2021 • 2 minutes to read • Edit Online

Microsoft has published a GitHub repository with scripts and tools that can help you prepare customized Help for your Business Central solution. This Help content can then be accessed from the user interface through the *Learn more* links as described in Configure Context-Sensitive Help. You are welcome to use any tools or processes to build and deploy content; this toolkit is intended to help you in some of the steps that are required.

The GitHub repository includes source code for the tools, and we welcome contributions and feedback to collaborate on improving the toolkit.

Tools in the toolkit

The toolkit is available at https://github.com/microsoft/dynamics365smb-custom-help. The repo contains the following tools as well as the source code for the tools:

- HtmlFromRepoGenerator tool

  For more information, see Custom Help Toolkit: The HtmlFromRepoGenerator tool

- Dynamics NAV field Help conversion scripts

  For more information, see Custom Help Toolkit: The FieldTopicTextExtractor tool

- HtmlLocaleChanger tool

  For more information, see Custom Help Toolkit: The HtmlLocaleChanger tool

Custom Help

Depending on your solution, you are expected to deploy Help to a website that can be accessed by users of your solution. For more information, see User Assistance Model and Configuring the Help Experience for Dynamics 365 Business Central.

See also

Configure Context-Sensitive Help
User Assistance Model
Migrate Legacy Help to the Dynamics 365 Business Central Format
Extend, Customize, and Collaborate on the Help for Dynamics 365 Business Central

Custom Help Toolkit: The HtmlFromRepoGenerator tool

2/17/2021 • 4 minutes to read • Edit Online

The custom help toolkit includes the **HtmlFromRepoGenerator** tool that gets Microsoft's content in MarkDown files and converts it to HTML files. You can then deploy the HTML files to a website.

The HtmlFromRepoGenerator tool is a wrapper around the DocFx component that Microsoft uses to generate HTML files for the Docs.microsoft.com site. You can write your own scripts around this component, and maybe you already have that in place. If you don't, then the HtmlFromRepoGenerator tool can help you get started. Run the tool from a command prompt, use the examples as inspiration but remember to update the paths to suit your configuration.

Use the HtmlFromRepoGenerator tool to get MarkDown files and generate HTML files

HtmlFromRepoGenerator.exe provides functionality that supports the creation of custom Help based on source files from Microsoft. You can use HtmlFromRepoGenerator.exe to:

- Clone a Microsoft documentation repository (repo)
- Update links to files that are no longer in the clone
- Update the **ms.locale** value to match the language options that are supported by Business Central
- Generate HTML files that can be used for publishing to your own website

  The HTML files will be generated in the **business-central\d365businesscentral** subfolder, such as *D:\BC\de-DE\business-central\d365businesscentral*. This name of the subfolder is set in the docfx.json file in the en-US source repo. The tool applies the `<meta name="robots" content="NOINDEX, NOFOLLOW">` tag to the HTML files, so if you use the tool to process your own content, you should remove these tags from your files but leave the tags in your version of Microsoft's content.

  The files are generated based on stylesheets and templates that are part of the tool.

- Compare a localized Microsoft repo to the en-US repo to identify differences and update links accordingly

  The en-US repo includes files that are required for building the various localized versions, including the very important docfx.json file that sets the output location and certain metadata settings, some of which are important for the Docs.microsoft.com site only. For more information, see Get content from the GitHub repos.

Syntax

Here is the syntax for HtmlFromRepoGenerator.exe:

```
HtmlFromRepoGenerator.exe --Out <path> [--DoNotClone <true|false>] [--Repo <URL>] [--RemoveGitFolder
<true|false>] [--LogsDir <.\logs>] [--EnRepo <URL>] [--EnOut <path>] [--Lng <language code>] [--Rtl] [--?[--
]]
```

The following table provides an explanation of the parameters:

PARAMETER	DESCRIPTION
Out	Specifies the folder where your existing clone is, or the folder to clone the repo to, such as D:\BC. If you run HtmlFromRepoGenerator to clone a repo, this folder must not already exist.
DoNotClone	Set this parameter when you run the tool against a previously cloned repo.
Repo	Specifies the repo URL. Optional if you run the tool based on a previously cloned repo. Examples of Microsoft documentation repo URLs include `https://github.com/MicrosoftDocs/dynamics365smb-docs` for English (US) and `https://github.com/MicrosoftDocs/dynamics365smb-docs-pr.de-de` for German (Germany).
RemoveGitFolder	Specifies whether to remove the `.git` folder.
LogsDir	Specifies the folder to save logs files to.

The following additional parameters are used when the tool is run against the localized Microsoft documentation repos:

PARAMETER	DESCRIPTION
EnRepo	Specifies the URL of the en-US repo. Optional if you run the tool on a previously cloned repo. The Microsoft documentation repo URL for English (US) is https://github.com/MicrosoftDocs/dynamics365smb-docs.
EnOut	Specifies the folder where the en-US repo exists, or the folder to clone it to. This folder must not already exist if you run the tool based on a previously cloned repo.
Lng	Specifies the language value to use for `ms.locale` metadata in the generated HTML files. If this parameter is not set, the tool uses en-US.
Rtl	Set this parameter if the language uses right-to-left (RTL) formatting. Examples of RTL languages include Arabic and Hebrew.

Examples

> **NOTE**
> The Microsoft repos contain many files, so the process takes several minutes. If you run the tool against multiple localization repos, the process takes longer.

The following example clones the en-US repo and generates HTML files.

```
HtmlFromRepoGenerator.exe --out "D:\BC\en-US" --repo "https://github.com/MicrosoftDocs/dynamics365smb-docs"
--LogsDir D:\BC\logs\en-US
```

The following example uses a previously cloned en-US repo and generates HTML files.

```
HtmlFromRepoGenerator.exe --out "D:\BC\en-US" --DoNotClone --LogsDir D:\BC\logs\en-US
```

The following example clones both the de-DE and en-US repos, and generates HTML files for de-DE.

```
HtmlFromRepoGenerator.exe --out "D:\BC\de-DE" --repo "https://github.com/MicrosoftDocs/dynamics365smb-docs-
pr.de-de" --EnRepo "https://github.com/MicrosoftDocs/dynamics365smb-docs" --EnOut "D:\BC\en-US" --lng "de-
DE" --LogsDir D:\BC\logs\de-DE
```

The following example uses previously cloned de-DE and en-US repos to generate HTML files for de-DE. Make sure that the de-DE repo is up to date if you use an existing cloned repo.

```
HtmlFromRepoGenerator.exe --out "D:\BC\de-DE" --DoNotClone --enOut "D:\BC\en-US" --lng "de-DE" --LogsDir
D:\BC\logs\de-DE
```

IMPORTANT

Do not run HtmlFromRepoGenerator.exe repeatedly on a previously-cloned repo. HtmlFromRepoGenerator modifies the links during processing, so running HtmlFromRepoGenerator more than once on the same content will result in incorrect links. If you want to rerun HtmlFromRepoGenerator, either use HtmlFromRepoGenerator to create a new clone of the repo, or revert all local changes to your existing clone.

Modifying the styling of the generated HTML files

The tool generates the HTML files based a set of predefined templates. In most cases, you can modify the stylesheets in the `styles` folder to modify the appearance of your content.

For advanced scenarios, you can modify the templates used by the HtmlFromRepoGenerator tool. The source files are included in the *SourceCode* folder in the GitHub repo. The templates are in the *SourceCode\HtmlFromRepoGenerator\HtmlFromRepoGenerator\HtmlFromRepoGenerator\Resources* subfolder.

NOTE

If you modify the templates, you must rebuild HtmlFromRepoGenerator.exe using Visual Studio or similar.

For more information, see Introduction to DocFX Template System.

See also

Custom Help Toolkit

Custom Help Toolkit: The HtmlLocaleChanger tool

2/17/2021 • 2 minutes to read • Edit Online

The custom help toolkit includes the **HtmlLocaleChanger** tool that can process HTML files containing Help content for Business Central to change the metadata for language.

HTML files published by Microsoft for documentation contain the ms.locale metadata which describes the language of the content. Depending on the website that you deploy your content to, the value can be used to switch between different translations of the content. For example, on the docs.microsoft.com site, the value specifies if the version of the article that is published under the URL `https://docs.microsoft.com/de-de/dynamics365/business-central` is in fact in German (Germany) or in another language, such as English. The same value is also used in the translation process.

Use the HtmlLocaleChanger tool to align locales

The **HtmlLocaleChanger** tool can update your HTML files with a new value for the *ms.locale* entry in the metadata for the specified files. For example, if you have HTML files for German (Germany) and you want to make the same content available in German (Austria), then copy the files to a new folder, and then run the tool to change the setting from *ms.locale: de-de* to *ms.locale:de-at*.

Here is the syntax for HtmlLocaleChanger.exe:

```
HtmlLocaleChanger.exe --h <path> --l <locale> --v <true|false>
```

Here is an explanation of the parameters:

PARAMETER	DESCRIPTION
h	Specifies the path to the HTML files that you want to process.
l	Specifies the new locale for the HTML files.
v	**true** to enable verbose logging; otherwise **false**.

Example

The following example changes the original locale to *de-at* with verbose logging:

```
HtmlLocaleChanger.exe --h D:\BC\de --l de-at --v
```

In the example, the `D:\BC\de` folder contains HTML files with the locale *de-de*, and the command changes that locale setting to *de-at*. You can now publish the relevant HTML files to your website in both locales.

See also

Custom Help Toolkit
Custom Help Toolkit: The HtmlFromRepoGenerator tool

Custom Help Toolkit: The FieldTopicTextExtractor tool

2/17/2021 • 2 minutes to read • Edit Online

If you are migrating a solution from Dynamics NAV to Business Central, you can use this tool to export the text contained in the first paragraph of field-level Help topics for Microsoft Dynamics NAV 2018 or earlier versions. You can then use these paragraphs as tooltip text in your Business Central solution.

> **NOTE**
> The tool does not import the tooltip text directly into your Business Central page objects. Instead it creates a spreadsheet that will make it easier to reuse the text from field-level Help topics in your Business Central solution.

Use the FieldTopicTextExtractor tool to extract content Excel

Use the **FieldTopicTextExtractor** tool to populate an Excel workbook with the field IDs and the corresponding first paragraph from a set of HTML topics from the Dynamics NAV field-level Help. The Dynamics NAV field topics have a name in the format `T_10_20.html` for field 20 on table 10. The name of the resulting Excel spreadsheet will be in the format Field_topic_summaries*.xlsx.

Here is the syntax for FieldTopicTextExtractor.exe:

```
FieldTopicTextExtractor.exe --n --o
```

Here is an explanation of the parameters:

PARAMETER	DESCRIPTION
n	Specifies the path to the HTML files with Dynamics NAV field topics.
o	Specifies the path where the Excel spreadsheet must be created.

Examples

The following example extracts the introductory paragraph from T_*.htm files in D:\NAV to an Excel spreadsheet with the name Field_topic_summaries*.xlsx in D:\BC:

```
NavFieldsTooltips.exe --n D:\NAV --o D:\BC
```

After extracting the Dynamics NAV field topic text

Excel makes it easy to bulk-apply and bulk-edit strings because you can sort and filter data. If you work with dedicated technical writers, then they are probably more efficient if they can work with tooltip text in Excel prior to adding tooltip text to your Business Central solution. You can also take the opportunity to revise the text. For more information, see Guidelines for tooltip text.

Before you import the tooltips into your solution, map the original field IDs to page control IDs in your Business Central solution. It requires knowledge of the actual solution to determine which tooltips are relevant for which pages, and Excel is very handy to help you in this process.

See also

Custom Help Toolkit
Custom Help Toolkit: The HtmlFromRepoGenerator tool
Migrate Legacy Help to the Business Central Format

Extend, Customize, and Collaborate on the Help for Dynamics 365 Business Central

2/17/2021 • 18 minutes to read • Edit Online

The source files for the Help for the base application are available in public GitHub repos so that you can easily extend and customize the content for your customers. In this section, you can learn about working with the GitHub repos and MarkDown files. You can also find guidance in the Docs Contributor Guide.

> **TIP**
> If you want to get Microsoft's content and deploy it to your own website with or without customizations, see Custom Help Toolkit.

Get content from the GitHub repos

There are repos in GitHub for the source content and each of the languages that Microsoft translates to. The dynamics365smb-docs repo contains the source content in English (US). If you want access to the content in other languages, navigate to the relevant repo - the names follow this pattern: `dynamics365smb-docs-pr.<language>-<country>`, such as dynamics365smb-docs-pr.da-DK for the Danish version.

You can use the HtmlFromRepoGenerator tool to get the latest version of Microsoft's content and generate HTML files that you can then customize. The tool handles the GitHub work for you, but you will still have to understand the basics of the Microsoft GitHub repos.

When Microsoft publishes an update to the content, the *live* branch in the corresponding GitHub repo is updated. The source repo is updated at least weekly; however, the related language-specific repos are updated less frequently, based on when new translations are made available. You can use the Custom Help Toolkit to get the current version of Microsoft's content and prepare HTML files for customization. Alternatively, if you customize the Microsoft content based on MarkDown, you can use scripts to get the current version. The GitHub platform and tooling will help you manage any potential merge conflicts if you have made changes to the same files as Microsoft has. For more information, see Set up Git repository locally for documentation in the Docs Authoring Guide and Fork a repo in the Help for GitHub.

> **TIP**
> You do not have to get acquainted with GitHub if you just want to get the Microsoft content in HTML format to deploy to a website, for example. You do not even have to get a GitHub account, as shown in the Getting by without GitHub section. However, in many scenarios, you might want to join us in GitHub for closer collaboration and easy of extensibility.

If you fork one of our repos, you can choose to update your fork with regular updates from the Microsoft repo.

For guidance about what the Microsoft-provided content for Business Central is all about, see User Assistance Model.

The remaining sections of this article are intended for people who do **not** use the Custom Help Toolkit - and for the curious. See the following table to find what you want to learn more about.

TO LEARN MORE ABOUT THIS SUBJECT	READ THIS SECTION
Files and subfolders in the GitHub repos	What the GitHub repos contain
How to interact with the GitHub repos without using the HtmlFromRepoGenerator tool	Get updates from Microsoft
The mechanics of working in GitHub based on our internal contributor guide	Get started with GitHub
How you can contribute to Microsoft's content	Contributing
Forking a repo without using the HtmlFromRepoGenerator tool	Get the content without a GitHub account
Generating content for your website without using the HtmlFromRepoGenerator tool	Build HTML files
Potential problems you might see when you customize Microsoft's content	Known issues with Microsoft's content
Using the Dynamics 365 Translation Service to manage translations	Translate the content

What the GitHub repos contain

Microsoft's GitHub *dynamics365smb-docs* repos for Business Central Help contain the following folders:

- archive

  Contains files that are not published but kept for backwards compatibility use internally at Microsoft. You can ignore this folder.

- business-central

  Contains files that are relevant for Business Central

- media-source

  Contains source files for some of the pictures that are used in the Business Central content

- Templates

  Contains a template that you can use if you build HTML files for the legacy Dynamics NAV Help Server website

The repos also contain files in the root of the repos that are used internally by Microsoft for managing the content on the docs.microsoft.com site and on GitHub. They are not relevant for the purpose of extending or customizing the content.

> **TIP**
> The Business Central installation media still contain CAB files for Help Server; however, the latest content is always available in the GitHub repos. If you find that the CAB files are outdated, or if they do not contain the files that you expect, get the latest files from GitHub. For more information, see the Get updates from Microsoft and Get the content without a GitHub account sections, respectively.

If you want to contribute to the developer and administration content, clone or fork the

`https://github.com/MicrosoftDocs/dynamics365smb-devitpro-pb` repo

Get updates from Microsoft

Microsoft makes frequent changes to the Business Central content, and those changes show up in the public GitHub repos. The base repo, MicrosoftDocs/dynamics365smb-docs, is updated weekly, and the translations are updated monthly. When you decide it is time to get the latest version of the content from Microsoft, you can do that using GitBash or GitHub Desktop. In the Help for GitHub, you can see an example of how this works in GitBash. In GitHub Desktop, just use the *Merge into current branch* menu item to pull changes from the origin into your fork.

However, if your solution is available in more than one country, then you are likely to want to make content available in multiple languages. Microsoft has a GitHub repo for each supported language, but the configuration files are only available in the English (US) source repo, MicrosoftDocs/dynamics365smb-docs.

The following script was developed by a Danish partner in order to get the Microsoft source for a number of languages, copy media files to the localization repos, and then build HTML files. The script is provided in agreement with the partner without further support.

```
$languages = $("da-dk","de-ch","de-de")
$git = "C:\Program Files\Git\cmd\git.exe"
$docfx = "C:\GitHub\DocFx\docfx.exe"
$365docs = "C:\GitHub\MSFT\dynamics365smb-docs"
$langDir = "c:\Working\help\dynamics365smb-docs-pr."

Start-Process -FilePath $git -ArgumentList "clone --single-branch --branch live
https://github.com/MicrosoftDocs/dynamics365smb-docs.git" -WorkingDirectory "C:\working\help" -Wait
foreach ($language in $languages)
{
    $arguments = $("clone --single-branch --branch live https://github.com/MicrosoftDocs/dynamics365smb-
docs-pr." + $language + ".git")
    Start-Process -FilePath $git -ArgumentList $arguments -WorkingDirectory "C:\working\help" -Wait
    Copy-Item $($365docs + "\business-central\docfx.json") $($langDir + $language + "\business-central")
    Copy-Item $($365docs + "\business-central\media") $($langDir + $language + "\business-central") -Recurse
-Force
    Copy-Item $($365docs + "\business-central\LocalFunctionality") $($langDir + $language + "\business-
central") -Recurse -Force
    Copy-Item $($365docs + "\Templates") $($langDir + $language) -Recurse -Force
    Set-Content -Path $($langDir + $language + "\business-central\docfx.json") -Value (get-content -Path
$($365docs + "\business-central\docfx.json"))
    Start-Process -FilePath $docfx -ArgumentList $("C:\working\help\dynamics365smb-docs-pr." + $language +
"\business-central\docfx.json" + " --output c:\working\output\" + $language)
}
```

For more information, see the Build HTML files section.

Because the Microsoft repos are public, you do not need a valid GitHub account in order to get the content. However, we recommend that your organization has a system account with access to GitHub at a minimum.

Get started with GitHub

To join Microsoft in the world of GitHub and MarkDown, there are new terminology and tools to get used to. The following list outlines the main steps, but you can find additional content, tools, and ideas in the GitHub documentation and other forums.

1. Fork the right repo

   You cannot work directly in the Business Central repos in the MicrosoftDocs GitHub org, such as the dynamics365smb-docs repo. The first thing you need to do is create a fork of the repo under your GitHub account. A fork is a copy of this repo that lets you work freely on the content without affecting the MicrosoftDocs/dynamics365smb-docs repo.

Alternatively, you can *clone* the Microsoft repo. This is useful if you don't intend to customize Microsoft's content, for example. But in many cases, *forking* the repo is more preferable.

For more information, see Set up your GitHub account and Set up Git repository locally for documentation in the Docs Authoring Guide.

> **TIP**
> You are not required to make your GitHub repos public. When you fork a public repo, you can specify in the settings for the new repo if the repo is public, private, or available only to specific GitHub accounts.

2. Install GitHub Desktop (optional) and clone your forked repo.

   GitHub Desktop makes is easy to work and collaborate with repos locally from your own desktop. For more information, see GitHub Desktop.

3. Get hold of your favorite MarkDown editor, and start making changes.

   The help content is stored in the *business-central* folder of the repo. Articles use a syntax for formatting text called Markdig Flavored Markdown, which is CommonMark compliant. To learn more about working with markdown, see Getting started with writing and formatting on GitHub.

   If you want to work locally, you can edit using any text editor. Just save the file as a .md type. Here are two good tools that provide you with some nice features, including a preview of how the content will be rendered in HTML:

   - Visual Studio Code

     Add the Docs Authoring Pack for Visual Studio Code, which gives you spell checker, MarkDown validation, and many other productivity features

   - Atom

     Atom has spell check and is good for managing many files

Internally at Microsoft, some authors use Code, others use Atom, and for light-weight work, we tend to just edit the content in the browser. You can find more guidance for how to get started with MarkDown in the Docs Contributor Guide. This guide is published by the team that built the Docs.microsoft.com site where the Business Central team publishes their docs.

Contributing

A benefit of GitHub is the ability for you to contribute to the core content that the Microsoft team provides in the dynamics365smb-docs repo. For example, you might have a new article that you think would be beneficial or you might have a correction to an existing article. If you would like to contribute to the MicrosoftDocs/dynamics365smb-docs repo, you create a *pull request* from your repo to the MicrosoftDocs/dynamics365smb-docs repo. The Microsoft team will then review the request and include the changes as appropriate.

> **NOTE**
> Microsoft accepts pull requests to the *dynamics365smb-docs* repo only, not the language-specific repos. If you have feedback about translations, you can report a GitHub issue in the relevant repo.

To create a pull request to the MicrosoftDocs/dynamics365smb-docs repo by using GitHub Desktop, do the following:

1. Commit the changes to your repo that you want to include in the pull request.

2. Choose **Sync** to push the changes up to your repo on GitHub.
3. When the sync is completed, choose **Pull Request**, make sure that the pull request points at the *live* branch, and then choose **Pull Request**.

Get the content without a GitHub account

If you do not want to collaborate with Microsoft on the content, you can get the latest version of the content from GitHub without a GitHub account. For example, if you want content that is newer than the content on the Business Central installation media, you can get the latest by simply downloading the content of the relevant GitHub repo, which you can do without a GitHub account - the Microsoft repos are public so that anyone can always get to them. Use the HtmlFromRepoGenerator tool, create your own scripts, or follow this process to fork a repo manually.

To get files without a GitHub account

1. Go to the relevant GitHub repo, such as this one for German: https://github.com/MicrosoftDocs/dynamics365smb-docs-pr.de-de/.

   You can see in the browser when the content was last updated.

2. Choose the green **Clone or download** button, and then choose **Download ZIP**.

3. Open the downloaded *dynamics365smb-docs-pr.de-de-live.zip* file and extract to a relevant location.

   Now you have a copy of Microsoft's content. Next, you can generate HTML files for use on your website as described in the Build HTML files section.

Build HTML files

For publishing to your own website, you can use the HtmlFromRepoGenerator tool that is part of the custom Help toolkit for Business Central to clone a repo and generate the corresponding HTML files.

Alternatively, you can create your own tooling and processes around DocFx, which is an open-source tool for converting markdown files. This section provides some guidance on how you can use DocFx to publish HTML files from your fork of one of the Microsoft repos. You can find additional tips in the Custom Help Toolkit article.

> **TIP**
> You can also use DocFx to generate content for the legacy Dynamics NAV Help Server. In that case, use the NAV docfx.json file from dynamics365smb-docs.

1. Install DocFx on your computer.

   DocFx is a command line tool, but you can also run it from a PowerShell script.

   You must provide a .JSON file that defines certain build settings, including the output folder in which to store the generated HTML files. We suggest that you use the docfx.json configuration file from the dynamics365smb-docs repo. For more information, see Getting Started with DocFX.

2. To change settings in the docfx.json file, open the docfx.json file from the folder containing your local clone in your preferred editor.

   The following table describes key parameters for you to customize.

PROPERTY	DESCRIPTION

PROPERTY	DESCRIPTION
dest	Specifies the output folder of the generated HTML files, such as `c:\Working\output\`.
template	Specifies the templates that the HTML files will be generated after. The default for Microsoft is blank, but the value can be a string or an array.
globalMetadata	Contains metadata that will be applied to every file, in key-value pair format. We encourage you to use this property to apply the `ROBOTS: NOINDEX, NOFOLLOW` metadata to each HTML file. The intent is that search engines will find Microsoft's original content on the docs.microsoft.com site rather than any customizations that you and hundreds of other may have published. For an example, see the NAVdocfx.json file. If you use the NAVdocfx.json file to build HTML files for non-Microsoft functionality, then change the value of the `ROBOTS` property. You can also add other global metadata, or metadata that applies to specific subfolders.
fileMetadata	Contains metadata that will be applied to specific files, based on the specified parameters, in key-value pair format. The default is currently blank.
markdownEngineName	Specifies the "flavor" of MarkDown to use to build the HTML files. The default is `markdig`.

For more information, see the Properties for build section in the DocFx user manual.

The docfx.json files in the Microsoft repos have additional settings for the docs.microsoft.com site. If you build the HTML files based on the docfx.json in the Microsoft repos, make sure that you have configured it for your needs.

3. If you have cloned a localization repo such as dynamics365smb-docs-pr.da-dk, you must also clone the dynamics365smb-docs repo and copy the content of the \business-central\media\ folder.

   The localization repos only contain the files that are translated into the relevant languages. Microsoft does not translate all illustrations; therefore, the localization repos do not contain the many untranslated images, screenshots, and other illustrations. If you build a localization repo as-is, then the HTML files will have broken links to the missing illustrations.

   In the sample script described above, the following command copies the media folder:

   ```
   Copy-Item $($365docs + "\business-central\media") $($langDir + $language + "\business-central") -
   Recurse -Force
   ```

4. Go to your desktop and open a command prompt.
5. Go to the docfx installation folder.
6. Run the equivalent of the following command:

   ```
   docfx "c:\GitHub\MSFT\dynamics365smb-docs\business-central\docfx.json"
   ```

The files are generated as .html files and stored in the output location that is specified in the docfx.json file.

> **IMPORTANT**
>
> Depending on the website that the HTML files will be deployed to, you might not be able to use the table of contents file (TOC.html) that is generated in this process. That file is structured based on the configuration of the https://docs.microsoft.com site. If you use the legacy Dynamics NAV Help Server, then you must use the ToC.xml file instead.

The table of contents on the docs.microsoft.com site is currently a MarkDown file, TOC.md, but we are planning to convert it to a YAML file in order to be more compliant with the docs.microsoft.com site. Once we have converted the TOC.md file to TOC.yml, you will still be able to use DocFx.exe to build HTML files, but you will have to port your customizations of the TOC.md file to the new YAML format.

The root of the MicrosoftDocs repos contain files that are related to internal Microsoft processes, such as `.openpublishing.build.ps1`. These scripts are used to validate and preview content, but they rely on internal Microsoft resources that are not publicly available. The `.openpublishing.redirection.json` file lists files that were published to the docs.microsoft.com site but have been deprecated later. As part of standard website practices, the docs.microsoft.com site uses redirection to avoid broken links when a page is deleted, and the `.openpublishing.redirection.json` file provides the mapping for redirection.

For inspiration for how to build your own help website, see How-to: Customize DFM Engine in the DocFx user manual and the Azure App Service documentation.

For tips and tricks about writing in MarkDown, see the Authoring Guide.

Links to anchors across languages

If your website supports two or more locales, you can use DocFx to generate HTML files for the relevant languages. However, you may experience problems with links to anchors, also known as bookmarks.

For example, if your content has a link from article1.md to a specific section in article2.md, that link would be formatted like this: `[My translated subheading](article2.md#my-translated-subheading)`. Then, when you run DocFx to generate HTML files in Danish, DocFx will convert that link to `[Min oversatte overskrift](article2.md#min-oversatte-overskrift)`. That is not a problem because the link will work in both English and Danish.

But if you then want to use that link elsewhere, the link only works for one of the languages because the anchor changed its name in the Danish translation. If you link to that subheading in article2 from your marketing site or support site, or if you use it as the value of the ContextSensitiveHelpPage property, then it only works in English.

To work around this problem, we recommend that you create explicit anchors by tagging your subheading to give it a *fixed anchor*. The following example illustrates how that would look in MarkDown:

```
### <a name="subheading"></a>My translated subheading
```

You would then be able to use the same link across all locales:
`[My translated subheading](article2.md#subheading)`, which would render in HTML as `myurl.com/docs/article2#subheading` across all languages.

For more information, see Using hashtag in cross reference in the GitHub documentation.

> **NOTE**
>
> Adding fixed anchors is only relevant if you want to generate a link to a subheading that works consistently across languages.

Alternatively, you can add a post-processing step to the script that you use to run DocFx to change the equivalent of `<h3 id=da-DK-anchor-name>` with `<h3 id=en-US-anchor-name>`. In this example, the step would change `<h3 id=min-oversatte-overskrift'>` to `<h3 id=my-translated-subheading'>`.

Known issues with Microsoft's content

Microsoft's content in the various GitHub repos is optimized for the docs.microsoft.com site and the tools that are used for this site. If you reuse Microsoft's content, you may experience a number of known issues, depending on how you publish your content. This section describes recommended steps to work around these issues.

Docs are not available for a specific version

Microsoft's public GitHub repos reflect the latest version of Business Central. If you want to deploy help for an earlier version of Business Central on-premises, then you can use the HTML files on the installation media. If you find that that particular version is missing content, then please check the following sections for suggested workarounds.

Broken links

If you deploy Microsoft's content to a website, your tools or your users will report that some links do not work. The links result in a 404 error or similar. These errors are caused by Microsoft having deleted the target files due to rework of the content. On the docs.microsoft.com site, we have tools that automatically handle links to deleted files through redirection. But if you deploy Microsoft's content to your own website, you will not have the same redirection.

We run periodic tests to catch these errors, but if you do see an error that is caused by a file not existing anymore, check the `.openpublishing.redirection.json` file in the root of the source repo. This file is used by the docs.microsoft.com site to manage redirection when a file is deprecated. For example, if you get an error that *"finance-how-to-set-up-sepa-direct-debit.md does not exist"*, then you can see in the `.openpublishing.redirection.json` file that the article has been deprecated and replaced by *finance-collect-payments-with-sepa-direct-debit.md*. You can replace the link in the file that is looking for *finance-how-to-set-up-sepa-direct-debit.md* to link to *finance-collect-payments-with-sepa-direct-debit.md* instead.

> **TIP**
> Use the HtmlFromRepoGenerator tool to manage this for you.

ToC.xml for Help Server is different from the TOC.md file

Microsoft does not currently maintain the ToC.xml file and does not add new features to it. While the Help Server component is still supported, it will be deprecated in 2021 release wave 1. As a result, it contains links that are broken as described in the previous section.

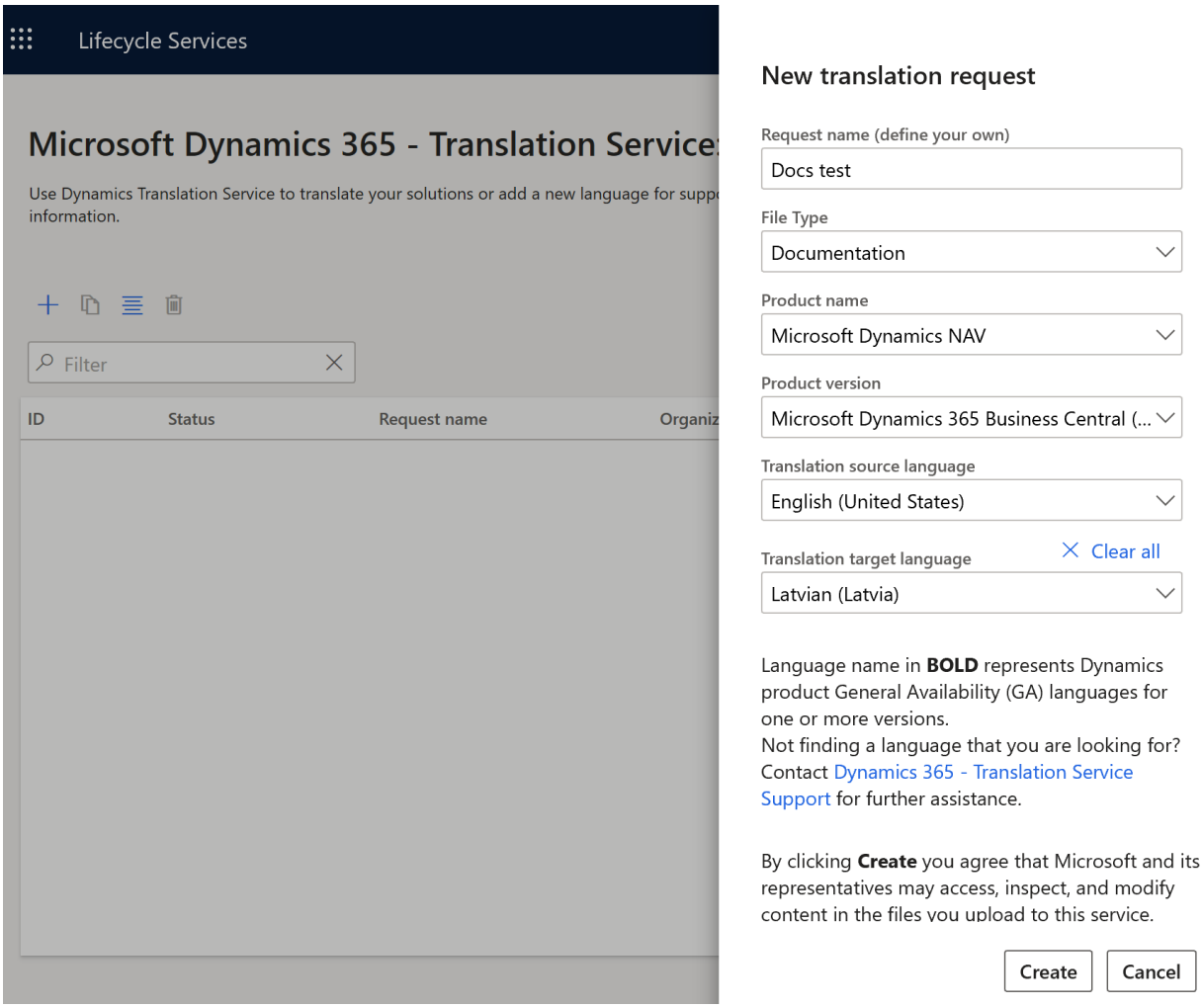
Translated content is not available

Microsoft creates content in English (US) that then gets translated into the Microsoft-provided target languages. The translations are available in the relevant localization repos within a few weeks.

Translate the content

You can use the Dynamics 365 Translation Service (DTS) to translate your own or the Microsoft-provided content into other languages. The service is hosted in Lifecycle Services and currently supports translation of content in Word documents and HTML files. For more information, see Translate documentation files.

To translate content for either Business Central or Dynamics NAV, choose Dynamics NAV as the product as shown in the following illustration:



See also

Business Central User Assistance Model
Configuring the Help Experience
Custom Help Toolkit
Custom Help Toolkit: The HtmlFromRepoGenerator tool
Custom Help Toolkit: The FieldTopicTextExtractor tool
Custom Help Toolkit: The HtmlLocaleChanger tool
Authoring Guide
Docs Contributor Guide
Docs Authoring Pack for Visual Studio Code
Getting started with writing and formatting on GitHub
Visual Studio Code
Atom
DocFx
Blog post: Extending and customizing the Help
Blog post: Collaborate on content for Business Central

Authoring Guide for Dynamics 365 Business Central

2/17/2021 • 9 minutes to read • Edit Online

If you are contributing to the Business Central Help, or if you are customizing and extending the Microsoft content for your own solution, you will probably want to use the same tools, processes, and style guide that Microsoft uses.

Resources

- The Microsoft Writing Style Guide is published online

  The content on the Docs.microsoft.com site generally follows the Microsoft Writing Style Guide. The content for Business Central varies in certain ways, partly with product-specific terminology, and a generally more conservative approach to contractions, for example.

- Extend, Customize, and Collaborate on the Help shows you the basics of collaborating on content for Business Central
- The Docs Contributor Guide has many tips and tricks for authoring in MarkDown
- The Docs Authoring Pack for Visual Studio Code adds productivity tools to Visual Studio Code

Write for accessibility

At Microsoft, we write for accessibility, which also means that the same content applies to interactions with the software across devices, regardless of input method, for example. For more information, see Describing interactions with UI in the Microsoft Writing Style Guide.

The accessibility requirements also impacts metadata for illustrations, such as the following:

```
:::image type="content" source="media/illustration.png" alt-text="Text used by screen readers":::
```

Most of Microsoft's articles use a different MarkDown formatting for illustrations, such as the following:

```
![Lightbulb that opens the Tell Me feature](media/ui-search/search_small.png "Tell me what you want to do")
```

Both formats are valid MarkDown, and both formats are supported by DocFx.exe. For more information, see Images in the Docs Contributor Guide.

Authoring in MarkDown

The Business Central content is styled using a MarkDown syntax as described below. Extended guidance is available in the MarkDown Reference section in the Docs Contributor Guide.

Headings

Use `#` for headings. For more information, see Headings in the Docs Contributor Guide.

In the source files for the Business Central content, which publishes as English (US) on the Docs.microsoft.com site, the title of an article is expected to use Title Case (capitalize each word, except prepositions) whereas subsequent headings use Sentence case (capitalize the first word, only). The Microsoft Writing Style Guide recommends a different approach.

Metadata

The content on the Docs.microsoft.com site includes metadata that is used by the site and feeds our internal telemetry dashboards. The metadata is required for Microsoft's content but not necessarily for content that extends or customizes Microsoft's content.

However, some metadata is recommended as a best practice as outlined in the following table.

METADATA TAG	DESCRIPTION
title	Used by search engines. Recommended length of 60-70 characters, including spaces. We recommend that the title metadata is the same as the first heading in the article.
description	Used by search engines. Recommended length of 100-160 characters, including spaces.
author	GitHub account of the main contributor to the article.
ROBOTS	Please apply this tag if you customize one of Microsoft's articles and deploy that to your website. By applying the metadata `ROBOTS: NOINDEX, NOFOLLOW`, you help make sure that search engines will find the original article rather than your customized version. Users of your Help will still be able to find the customized article through context-sensitive Help links and other links.

Bulleted lists

Use `-` or `*` to create bullets as shown in the following example:

```
The following options are available:

- first option
- second option
- third option
```

> **TIP**
> It doesn't matter if you use `-` or `*`, but the current version of the Docs Authoring Pack requires each article to use one or the other and not mix them up.

As a best practice, the MarkDown validation in the current version of the Docs Authoring Pack recommends a blank line between options in a bulleted list for better readability in MarkDown.

Ordered lists

Use numbers for ordered lists. Do not add spaces between the lines.

```
1. Choose the ![Lightbulb that opens the Tell Me feature](media/ui-search/search_small.png "Tell me what you
want to do") icon, enter **Payment Journal**, and then choose the related link.
2. In the **Payment Journal** window, on the first journal line, enter the relevant information about the
payment entry.
3. To apply a single vendor ledger entry:
4. In the **Applies-to Doc. No.** field, choose the field to open the **Apply Vendor Entries** window.
```

Bold and italics syntax

Use `**bold**` and `*italics*`

Tables

For tables in the body, use the markdown syntax. The Docs Authoring Pack for Visual Studio Code has a shortcut for adding a table, and you can also use Tables Generator.

```
| To   | See                       |
|------|---------------------------|
|<text>|<link>|
| | |
| | |
```

MarkDown syntax for nested tables is limited, so we recommend using HTML-syntax for nested tables in ordered and unordered lists use HTML-syntax.

Placeholders

Rather than repeating text in two or more articles, use *includes*. For more information, see Included Markdown files.

For Business Central, we use includes for boilerplate text, for content that is repeated in more than one article, and for the product name. That way, we can make changes in just one location - and so can you.

> **TIP**
> In the dynamics365smb-docs repo, the includes are in the `business-central\includes` subfolder.
>
> In December 2020, the two placeholders for the product name, prodshort.md and prodlong.md, were renamed to prod_short.md and prod_long.md. The change solved a problem internally at Microsoft, because we have a tool that helps identify spelling error, and that tool generated warnings for prodshort.md and prodlong.md.

Comment syntax

Useful for sections that are not ready and will not pass build checks.

```
<!-- Comments -->
```

Examples:

```
<!-- [Managing Payables](payables-manage-payables.md)-->
<!-- This is a paragraph that spans more lines and I can just put the comment tag
at the beginning and end of it -->
```

Links

Ordinary link to a different topic in the same folder

These links have the format `[link text](filename.md)`.

Example: `[Managing Payables](payables-manage-payables.md)`

Link to a topic in a subfolder of the source topic

These links have the format `[link text](subfolder/filename.md)`.

For example, you want to link to payables-manage-payables.md from ui-work-general-journals.md, where the folder structure is as follows:

- articles

    - ui-work-general-journals.md
- ManagePayables
    - payables-manage-payables.md

Here is the link: `[Manage Payables](ManagePayables/payables-manage-payables.md)`

Link to a topic in a different folder than source topic

These links have the format `[link text](../folder/filename.md)`.

For example, you want to link to payables-manage-payables.md from receivables-manage-receivables.md, where the folder structure is as follows:

- articles
    - ManageReceivables
        - receivables-manage-receivables.md
        - ui-work-general-journals.md
        - ManagePayables
            - payables-manage-payables.md

Here is the link: `[Manage Payables](../ManagePayables/payables-manage-payables.md)`

Link to a place in the same article

From within an article, you can create a link to a specific heading in the same article. You can create the link like other links except with the following format:

`[link text](#target-heading)`

target-heading is the text of the heading that you want to link to, except it is all lowercase and spaces between words are replaced with hyphens. For example, here is the link:
`[How Autoscaling Works](#how-autoscaling-works)`

To the heading: `## How Autoscaling Works`

Link to a place in a different article

From an article, you can create a link to a specific heading in another article. You can create the link like other links except with the following format:

`[link text](targetarticlename#target-heading)`

targetarticlename is the file name of the article, including the .md file type. target-heading is the text of the heading that you want to link to, except it is all lowercase and spaces between words are replaced with hyphens.

For example, to link to the heading "How autoscaling works" in the article Autoscaling.md, add the following code: `[How autoscaling works](Autoscaling.md#how-autoscaling-works)`

Links to anchors across languages

If your website supports two or more locales, you can use DocFx to generate HTML files for the relevant languages. However, you may experience problems with links to anchors, also known as bookmarks.

For example, if your content has a link from article1.md to a specific section in article2.md, that link would be formatted like this: `[My translated subheading](article2.md#my-translated-subheading)`. Then, when you run DocFx to generate HTML files in Danish, DocFx will convert that link to `[Min oversatte overskrift](article2.md#min-oversatte-overskrift)`. That is not a problem because the link will work in both English and Danish.

But if you then want to use that link elsewhere, the link only works for one of the languages because the anchor changed its name in the Danish translation. If you link to that subheading in article2 from your marketing site or

support site, or if you use it as the value of the ContextSensitiveHelpPage property, then it only works in English.

To work around this problem, we recommend that you create explicit anchors by tagging your subheading to give it a *fixed anchor*. The following example illustrates how that would look in MarkDown:

```
### <a name="subheading"></a>My translated subheading
```

You would then be able to use the same link across all locales:
`[My translated subheading](article2.md#subheading)`, which would render in HTML as
`myurl.com/docs/article2#subheading` across all languages.

For more information, see Using hashtag in cross reference in the GitHub documentation.

> **NOTE**
> Adding fixed anchors is only relevant if you want to generate a link to a subheading that works consistently across languages.

Line breaks (soft return)

In the editor, add two blank spaces at the end of the sentence and hit return. This is used in the See Also list. (See Also must be heading 2.)

Continue steps after a non-step para

Enter four spaces in front of the non-step para. Otherwise, the non-step para will restart the step sequence.

TOC

The TOC structure of the TOC.md file is as follows:

```
#[Overview](overview.md)
 ##[Topic 1](topic-1.md)
 ##[Topic 2](topic-2.md)
 ##[Topic 3](topic-3.md)
 ##[Topic 4](topic-4.md)
```

Standard phrases

All fields in Business Central have tooltips; therefore, do not document fields in Help. To refer readers to the tooltips, use this standard phrase where relevant:

> "Choose a field to read a short description of the field or link to more information."

For more information, see Business Central User Assistance Model.

Recycle content

Rather than copy-pasting content that you want to surface in two or more places, use *includes*. For more information, see Included Markdown files in the Docs Contributor Guide.

Topic titles

- Use imperative verb form for step-based topics ("Pay vendors").
- Use gerund verb form for conceptual, non-step topics. ("Paying Vendors")
- Use nouns for highest-level topics. ("Sales")

File naming

Rules

- No spaces or punctuation characters. Use hyphens to separate the words in the file name.
- Use all lowercase letters
- No more than 80 characters - this is a publishing system limit
- Use action verbs that are specific such as develop, buy, build, troubleshoot. No -ing words.
- No small words - don't include a, and, the, in, or, etc.
- All files must be in markdown and use the .md file extension.

Examples

TOPIC TITLE	NAMING
Select a Company	ui-how-select-company.md
Enter Criteria in Filters	ui-enter-criteria-filters.md
Troubleshooting: Record Locked by Another User	ui-troubleshoot-record-locked-another-user.md
Changing Basic Settings	ui-change-basic-settings.md
Sales	sales-manage-sales.md
Set Up Currencies	finance-setup-currencies.md
Set Up Purchasers	purchases-how-setup-purchasers.md
Understanding Session Timeouts	admin-understand-session-timeouts.md
Manage Data Encryption	admin-manage-data-encryption.md

Country-specific content

To simplify content localization and translation, country-specific articles live in country-specific folders. The TOC entries live under the "Local Functionality" parent node.

See also

Business Central User Assistance Model
Extend, Customize, and Collaborate on the Help
Configuring the Help Experience
Docs Contributor Guide
Docs Authoring Pack for Visual Studio Code
Getting started with writing and formatting on GitHub
Visual Studio Code
Atom
DocFx
Blog post: Extending and customizing the Help
Blog post: Collaborate on content for Business Central

Administration of Business Central Online

2/17/2021 • 13 minutes to read • Edit Online

As an administrator, you can manage Business Central online tenants both as an internal administrator, who is typically an employee of the company that bought the Business Central subscription, and as an administrator from the reselling partner company. Some of the tools are the same, and some tools are available to partners only. Here you can learn which tools are available to you as an administrator.

Administration as an internal administrator

Internal administrators are the system administrators, IT professionals, or superusers of the customer's company, who are assigned the **Global admin** role in the Microsoft 365 admin center. For more information, see About admin roles in the Microsoft 365 admin content.

Administration in Business Central

As the internal administrator, you can add users and grant permissions. For more information, see Create Users According to Licenses and Assign Permissions to Users and Groups in the business functionality content for Business Central.

As an internal administrator you have **read-only** access to all areas of Business Central. You will have this access even if your organization decides not to continue with Business Central and cancels the subscription. That way, you, as the internal administrator, can export relevant data, while no one else in your organization has a license to sign in and use Business Central.

> **IMPORTANT**
>
> You must have a Business Central license in order to set up integration to other products, or perform any other tasks in Business Central, except the two mentioned above. For information about licensing, see Microsoft Dynamics 365 Business Central Licencing Guide.

For other tasks, you can access the Business Central administration center, where you can manage upgrade schedules and other tasks. For more information, see The Business Central Administration Center.

You can also use telemetry to track usage and monitor user sessions, for example. For more information, see Monitoring and Analyzing Telemetry and Managing Sessions.

Administration of a trial

If your organization has signed up for a Business Central trial, you can extend the free trial, and you can start the process of finding a reselling partner to help you get a subscription. For more information, see Dynamics 365 Business Central Trials and Subscriptions.

Administration in the Microsoft 365 admin center

The **Global admin** role makes you an administrator of your organization's Microsoft 365 tenant. This means that you can manage the subscription, add or remove users, and assign or remove licenses in the Microsoft 365 admin center. For more information, see Microsoft 365 Admin help center.

Collaboration with reselling partners

When your organization subscribes to Business Central, you have a relationship with an authorized partner of Microsoft. The partner company assists with licensing, configuration, and other tasks. They can also help you get telemetry about your Business Central environment.

The partner will have access to your tenant as a *delegated administrator*. You can configure their access to your data. For more information, see Managing delegated permissions as an internal administrator.

If your organization decides to switch to another partner, you must take the following steps:

1. Ask your current partner to remove the reseller relationship with you in the Partner Center
2. Remove their delegated administration privileges
    a. In the Microsoft 365 admin center, under **Settings**, choose **Partner relationships**, and then select the partner of interest
    b. In the details pane, choose **Remove delegated admin**
    c. In the confirmation pane, choose **Remove**

    You must also disable their user accounts from your Business Central.
3. Remove any settings in the Business Central administration center if the partner did not already clear their settings.

    For more information, see Internal administrators.
4. Add your new partner to your subscription, and work with them to get them set up

Unsubscribing from Business Central

If the organization decides not to continue with Business Central, you can then cancel the subscription.

In the Microsoft 365 admin portal, you can remove licenses from users. As the administrator, you can remove a trial subscription from your company's account. But to cancel a paid subscription, you must contact your reselling partner, and they can cancel the relevant subscription in the Partner Center.

For more information, see Unsubscribe or Remove Business Central in the business functionality content for Business Central.

Administration as a partner

As a Business Central reselling partner, you are the administrator of the Business Central tenants of your customers. You have access to the administration tools of their Microsoft 365 account and their Business Central administration center where you can specify update windows. You can also log into their Business Central as a *delegated administrator* if you want to reproduce errors.

Join the Microsoft Partner Network

Microsoft Partner Network (MPN) membership unlocks our best resources to differentiate your business, take your product to market, and sell your solutions. To become a partner, you must join the Microsoft Partner Network (MPN), at which time you will be assigned an MPN ID. MPN membership is free to all partners; you can enroll in the MPN here.

Once signed up, you will get an MPN ID – your gateway to access all the membership resources and benefits for your partnership with Microsoft. There is no cost to obtain a MPN ID as a Network member, and with options to upgrade to an Action Pack subscription or work toward a competency, you can access even more benefits.

Set up your Partner Center account

Once you have joined the Microsoft Partner Network (MPN), you can set up your Partner Center (PC) account. The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Your Partner Center account provides you with access to pricing information, tools and services, and enables you to manage admin credentials for your company's work account. Partner Center is also where you can

purchase or renew subscriptions to Microsoft Action Packs, create a business profile to receive and manage sales leads from Microsoft, and see if you qualify for co-selling opportunities.

Enroll in the CSP program

The Cloud Solution Provider (CSP) program helps your company to be more involved in your customers' businesses, beyond reselling licenses. In CSP, you can choose to enroll as an *indirect reseller* or a *direct bill partner*.

In most cases, you will enroll as an *indirect reseller* and then work with an *indirect provider*, also referred to as a *distributor*, who then manages all interaction with Microsoft in terms of licensing and technology, so that you can focus on sales and support. If you decide to enroll as a *direct bill partner* in order to fully own the end-to-end relationship with both customers and Microsoft, make sure that you meet the eligibility requirements. For more information, see Enroll in the Cloud Solution Provider program in the Microsoft Partner Center content.

The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Some indirect providers (distributors) provide their resellers with a custom portal that optimizes and enhances the experience beyond the Partner Center. They can also provide indirect resellers with an API to automate some of the customer onboarding steps. Contact your indirect provider to find out more.

Both indirect resellers and direct bill partners can access and support their customers' Business Central by setting up a reseller relationship with them.

To service customers in a specific country, your partner company's Azure Active Directory (Azure AD) tenant and CSP account must be registered in the regional CSP market that covers that country. For more information, see Cloud Solution Provider program regional markets and currencies.

> **NOTE**
>
> When you buy Business Central offers on behalf of your CSP customers, the CSP offer must be available in both your own tenant's country and in your customer's tenant's country. For example, if your tenant is located in Slovakia and the customer's tenant is in Germany, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.
>
> Similarly, if your tenant is located in Germany and the customer's tenant is in Slovakia, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.

In the Microsoft Partner Center documentation, you can learn how to request a reseller relationship with customers, assign licenses to users, and create new subscriptions. Business Central is one of the subscriptions that you can create, and there are Business Central-specific license types that you can assign to users.

You must also assign certain roles to users in your own organization so that they can support your customers.

Add users from your own organization

In most cases, your partner company includes employees with different responsibilities, and you can assign different roles depending on people's responsibility. This way, you can be very explicit about who from your staff can access your customers' data, for example.

For each user, you can assign permissions for 2 categories of tasks:

- Managing your organization's account

  This controls access to the functionality of the Partner Center portal

- Assisting your customers

This controls access to your customers' environments

When you establish a reseller relationship with a customer in Partner Center, you must specify which people from your organization will assist that customer's tenant as either *Admin agent* or *Helpdesk agent*. These users can manage implementation, support, and troubleshooting tasks for your customers. Once the reseller relationship with a customer is established, they will be able to login into the Business Central environments of the customer without a license. The number of partner users that can access customer environments is not restricted.

Both roles will provide your users with exactly the same level of access to the customer's Business Central environment. However, they have very different capabilities to manage the customer's Active Directory and other subscriptions that the customer might have. For more information, see Admin roles.

This way of accessing customer resources is called *delegated administration*, and the partner users are therefore called *delegated administrators* or *delegated admins* in daily shorthand. For more information, see Delegated Administrator Access to Business Central Online.

> **NOTE**
> These users cannot provide accounting services for the customers. For this purpose, the customers must use the **External Accountant** license, which is also available via CSP.

Connect with customers

As a CSP partner, you can manage a customer's subscriptions and services on their behalf in the Partner Center by establishing a reseller relationship with your customers. If they already have an account, such as if they currently use Microsoft 365, other Dynamics 365 apps, or PowerApps, for example, you can send them an invitation straight from the Partner Center. For more information, see Connect with customers in Partner Center (indirect providers/distributors) and Connect with customers (indirect resellers).

When the customer's internal administrator receives the invitation link and navigates to it, they must acknowledge that they have read the Microsoft Customer Agreement and that they can authorize you as their reseller on behalf of their organization. For more information, see Confirm customer acceptance of the Microsoft Customer Agreement.

When a customer accept this reseller relationship, delegated administration privileges are granted to the partner. For more information, see Delegated Administrator Access to Business Central Online.

If you are working as an indirect reseller, your indirect provider (distributor) must associate your customers with you in their Partner Center.

Acting as a delegated administrator

As a reselling partner, you have delegated administration access to their Business Central and other services. For more information, see Delegated Administrator Access to Business Central Online.

Manage technical support

As a reselling partner, you must be the first line of support for your Business Central customers. You must set up your support contact information, and help the internal administrators troubleshoot any issues that users find. For more information, see Managing Technical Support.

Extending trials

An organization can sign up for a free trial of Business Central. When they first sign up for Business Central, they get access to an evaluation version that does not include all capabilities in Business Central. They can then switch to the 30 day trial experience to enable all capabilities.

However, sometimes a 30 day trial is not enough to decide if they want to buy Business Central. In that case, they can extend their trial with an additional 30 days. For more information, see Need More Time to Decide Whether to Subscribe? in the business functionality content for Business Central.

> **NOTE**
> If you are a reselling partner, we recommend that you set up demo environments for prospects that need longer time to decide if they want to buy Business Central. You can also use demo environments to help customers train their employees, for example. Using the 30 days trials for training should be limited to just that short period. However, demo environments cannot be used for production. For more information, see Preparing Demonstration Environments.

If the prospect wants to extend the trial further than those 30 days, they must contact a partner. The partner can extend it another 30 days if the delegated administrator signs into the prospect's Business Central and runs the **Extend Trial Period** guide.

After those additional 30 days, the prospect must either purchase Business Central or abandon Business Central. At this point, they will have had up to 90 days with the trial experience.

> **TIP**
> As a reselling partner, you can suggest your prospects sign up for a trial, but you can also help set up a customized demonstration environment based on a sandbox environment or a trial environment. In both cases, you can easily add or remove functionality based on your prospects' expectations. For more information, see Preparing Demonstration Environments.

The Dynamics 365 Business Central Premium Trial

In the Partner Center, you can find a special license type called the **Dynamics 365 Business Central Premium Trial** license, which is a very different way to give a prospect or an existing customer a trial experience using their own data. If you assigned the **Dynamics 365 Business Central Premium Trial** license to a customer's account in the Partner Center, then that also expires after 30 days. You cannot extend the Premium trial, but you can add one more Premium trial license to give the customer an extra 30 days of trial. But when the second Premium trial expires, then the customer must either convert their trial to a paid subscription, or they must sign up for the viral trial as described above.

For more information, see Offer your customers trials of Microsoft products in the Partner Center documentation. If you have technical difficulties assigning this license, contact Partner Center support. For more information, see Report problems with Partner Center.

Caution

Make sure you understand the limitations of this type of trial, before you offer it to a prospect or customer. It is easy to convert this type of trial to a paid subscription, but if the prospect needs more than 30 days to decide, or if they want to add more than 25 users, then the viral trial is probably a better fit for them.

Data and access when a trial or subscription ends

Trials can expire, and so can a apid subscription, such as if the customer does not renew the subscription or stops payments, or if the customer cancels the subscription.

When a trial or a subscription expires, two special periods kick in:

- *Grace period*
- *Data retention period*

The *grace period* is a period of 30 days when the customer can still access the product without any restrictions. After the grace period, the subscription enters the *data retention period*, which is another 90 days during which

only the admin users of the Azure Active Directory tenant can login into the product. After those 90 days, Microsoft deletes the data automatically.

This policy is general across Microsoft's online offerings, as you can see in this example from Microsoft 365: What happens to my data and access when my Microsoft 365 for business subscription ends?.

See Also

The Business Central Administration Center
The Business Central Administration Center API
Submitting support requests on behalf of your customer
Resources for Help and Support for Dynamics 365 Business Central
How does Microsoft handle database sizes?
Version numbers in Business Central
Get Started as a Reseller of Business Central Online
Deliver consulting services as a VAR: aka.ms/BusinessCentralConsultingServices
Monitoring and Analyzing Telemetry

Get Started as a Reseller of Business Central Online

2/17/2021 • 9 minutes to read • Edit Online

If you want to build your business on Dynamics 365 Business Central online, you must get set up as a reseller in the Microsoft Partner Center. In this article, we take you through the first four steps in your journey.

Step 1: Become a partner

Becoming a Microsoft partner gives you access to the Microsoft resources needed to sell solutions. The steps in this section below are required to gain access to the programs that enable are intended to help you get set up to sell Business Central.

In order to be able to service Business Central licenses and provide help and support to your customers, your company must have a Microsoft Partner ID (MPN ID), and you must enroll in the Cloud Solution Provider (CSP) program.

Join the Microsoft Partner Network

Microsoft Partner Network (MPN) membership unlocks our best resources to differentiate your business, take your product to market, and sell your solutions. To become a partner, you must join the Microsoft Partner Network (MPN), at which time you will be assigned an MPN ID. MPN membership is free to all partners; you can enroll in the MPN here.

Once signed up, you will get an MPN ID – your gateway to access all the membership resources and benefits for your partnership with Microsoft. There is no cost to obtain a MPN ID as a Network member, and with options to upgrade to an Action Pack subscription or work toward a competency, you can access even more benefits.

Set up your Partner Center account

Once you have joined the Microsoft Partner Network (MPN), you can set up your Partner Center (PC) account. The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Your Partner Center account provides you with access to pricing information, tools and services, and enables you to manage admin credentials for your company's work account. Partner Center is also where you can purchase or renew subscriptions to Microsoft Action Packs, create a business profile to receive and manage sales leads from Microsoft, and see if you qualify for co-selling opportunities.

Enroll in the CSP program

The Cloud Solution Provider (CSP) program helps your company to be more involved in your customers' businesses, beyond reselling licenses. In CSP, you can choose to enroll as an *indirect reseller* or a *direct bill partner*.

In most cases, you will enroll as an *indirect reseller* and then work with an *indirect provider*, also referred to as a *distributor*, who then manages all interaction with Microsoft in terms of licensing and technology, so that you can focus on sales and support. If you decide to enroll as a *direct bill partner* in order to fully own the end-to-end relationship with both customers and Microsoft, make sure that you meet the eligibility requirements. For more information, see Enroll in the Cloud Solution Provider program in the Microsoft Partner Center content.

The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Some indirect providers (distributors) provide their resellers with a custom portal that optimizes and enhances the experience beyond the Partner Center. They can also provide indirect resellers with an API to automate some of the customer onboarding steps. Contact your indirect provider to find out more.

Both indirect resellers and direct bill partners can access and support their customers' Business Central by setting up a reseller relationship with them.

To service customers in a specific country, your partner company's Azure Active Directory (Azure AD) tenant and CSP account must be registered in the regional CSP market that covers that country. For more information, see Cloud Solution Provider program regional markets and currencies.

> **NOTE**
>
> When you buy Business Central offers on behalf of your CSP customers, the CSP offer must be available in both your own tenant's country and in your customer's tenant's country. For example, if your tenant is located in Slovakia and the customer's tenant is in Germany, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.
>
> Similarly, if your tenant is located in Germany and the customer's tenant is in Slovakia, you will not be able to sell Dynamics 365 Business Central Premium to that customer, because this offer is currently not available in Slovakia.

In the Microsoft Partner Center documentation, you can learn how to request a reseller relationship with customers, assign licenses to users, and create new subscriptions. Business Central is one of the subscriptions that you can create, and there are Business Central-specific license types that you can assign to users.

You must also assign certain roles to users in your own organization so that they can support your customers.

Add users from your own organization

In most cases, your partner company includes employees with different responsibilities, and you can assign different roles depending on people's responsibility. This way, you can be very explicit about who from your staff can access your customers' data, for example.

For each user, you can assign permissions for 2 categories of tasks:

- Managing your organization's account

  This controls access to the functionality of the Partner Center portal

- Assisting your customers

  This controls access to your customers' environments

When you establish a reseller relationship with a customer in Partner Center, you must specify which people from your organization will assist that customer's tenant as either *Admin agent* or *Helpdesk agent*. These users can manage implementation, support, and troubleshooting tasks for your customers. Once the reseller relationship with a customer is established, they will be able to login into the Business Central environments of the customer without a license. The number of partner users that can access customer environments is not restricted.

Both roles will provide your users with exactly the same level of access to the customer's Business Central environment. However, they have very different capabilities to manage the customer's Active Directory and other subscriptions that the customer might have. For more information, see Admin roles.

This way of accessing customer resources is called *delegated administration*, and the partner users are therefore called *delegated administrators* or *delegated admins* in daily shorthand. For more information, see Delegated Administrator Access to Business Central Online.

> **NOTE**
>
> These users cannot provide accounting services for the customers. For this purpose, the customers must use the **External Accountant** license, which is also available via CSP.

Caution

In the Microsoft 365 admin center and Microsoft Azure Management portal, both customers and partners can invite external users (guests) into their Active Directory. When a partner user is added as a guest to the customer's Azure AD, they can no longer log in as a delegated admin into the customer's Business Central. In order to log in, the local user (guests or native) must have a valid Business Central license assigned to them.

Step 2: Go to market

When you become a Microsoft Partner Network member, you gain access to membership benefits that can help you build and grow your business. For more information, see Explore your Go-To-Market with Microsoft offers in the Partner Center docs.

As a Dynamics 365 reseller, you benefit from Microsoft's investments in an always up-to-date modern platform, you can bundle recognized apps from the Microsoft commercial marketplace into an offering that fits the needs of your customers, reach more customers by using Microsoft's commercial marketplace to promote your packaged consulting service offerings or customization services, and streamline your own processes and build tools with Power BI, Power Automate, and Power Apps connected to Business Central.

Specifically for going to market with Business Central, Microsoft provides resources and guidance on the Business Applications for small and medium-sized businesses (SMBs) site. For more information, see Business Central Go-To-Market resources.

Step 3: Give powerful demos

You can create a trial environment based on content in cdx.transform.microsoft.com.

For more information, see Preparing Demonstration Environments of Dynamics 365 Business Central.

Step 4: Help your customers get started

When you onboard customers to Business Central, you have access to their account as a delegated administrator and can help them get things set up. For more information, see Administration of Business Central Online.

Connect with customers

As a CSP partner, you can manage a customer's subscriptions and services on their behalf in the Partner Center by establishing a reseller relationship with your customers. If they already have an account, such as if they currently use Microsoft 365, other Dynamics 365 apps, or PowerApps, for example, you can send them an invitation straight from the Partner Center. For more information, see Connect with customers in Partner Center (indirect providers/distributors) and Connect with customers (indirect resellers).

When the customer's internal administrator receives the invitation link and navigates to it, they must acknowledge that they have read the Microsoft Customer Agreement and that they can authorize you as their reseller on behalf of their organization. For more information, see Confirm customer acceptance of the Microsoft Customer Agreement.

When a customer accept this reseller relationship, delegated administration privileges are granted to the partner. For more information, see Delegated Administrator Access to Business Central Online.

If you are working as an indirect reseller, your indirect provider (distributor) must associate your customers with

you in their Partner Center.

The customer can decide to remove their partner's delegated admin privileges from their tenant, but still retain the relationship with their partner for subscription and license renewal purposes. Removing delegated admin privileges will block the partner's access to the customer's Business Central. The access can be restored if the partner sends the reseller relationship request to the customer again.

If the customer wants to cancel their relationship with their partner, the partner must remove the relationship in Partner Center.

Get Business Central right for the customer

The default version of Business Central is just that - a default version. In many cases, you'll enhance the default version with apps from the Microsoft commercial marketplace. But you can also customize pages for a profile and change which UI elements are visible. For more information, see Customize Business Central in the business functionality content.

If your customer wants more tweaks, you can create customizations of profiles and pages in code. For more information, see Customizing the User Interface for User Roles in the developer content.

Step 5: Configure the support experience

As a Business Central reselling partner, you are an administrator of your customers' Business Central tenants, and you are the first line of support. This means that you will get requests for support from your customers that you must triage, investigate, and either resolve or escalate to Microsoft. For more information, see Technical Support for Business Central.

Step 6: Maintain your customers' Business Central

As a Business Central reselling partner, you are the administrator of the Business Central tenants of your customers. You are expected to help your customers maintain their solution, including setting the upgrade window, monitoring telemetry, updating customizations, and managing apps.

For more information, see Administration of Business Central Online and The Business Central Administration Center.

See also

Administration of Business Central Online
Deployment of Dynamics 365 Business Central on-premises
Trials and Sign-ups for Business Central Online
Licensing in Dynamics 365 Business Central
Learn how to partner with indirect providers in the Cloud Solution Provider program

Country/regional availability and supported languages

2/17/2021 • 4 minutes to read • Edit Online

This page lists the countries/regions where Dynamics 365 Business Central is available and which languages are supported.

> **NOTE**
> In countries where Microsoft has not delivered a localization, partners can build localizations using translation and localization apps that are published on AppSource. These apps are built on top of the international (W1) version of Dynamics 365 Business Central.

The following table provides the list of all countries/regions where Dynamics 365 Business Central is available or in planning for future availability, and states whether the localization is provided by Microsoft or partner(s).

Countries and regions

COUNTRY/REGION	LOCALIZED BY	AVAILABILITY	ENVIRONMENT COUNTRY IN TENANT ADMIN CENTER
Australia	Microsoft	Available	AU
Austria	Microsoft	Available	AT
Belgium	Microsoft	Available	BE
Brazil	Partner	Available	BR
Canada	Microsoft	Available	CA
Colombia	Partner	Available	CO
Croatia	Partner	Available	HR
Czechia	Microsoft	Available	CZ
Denmark	Microsoft	Available	DK
Estonia	Partner	Available	EE
Faroe Islands (Denmark)	Microsoft	Available	DK
Finland	Microsoft	Available	FI
France	Microsoft	Available	FR
Germany	Microsoft	Available	DE

COUNTRY/REGION	LOCALIZED BY	AVAILABILITY	ENVIRONMENT COUNTRY IN TENANT ADMIN CENTER
Greenland (Denmark)	Microsoft	Available	DK
Hong Kong SAR	Partner	Available	HK
Hungary	Partner	Available	HU
Iceland	Microsoft	Available	IS
Ireland	Partner	Available	IE
Indonesia	Partner	Available	ID
India	Microsoft	Preview	IN
Italy	Microsoft	Available	IT
Japan	Partner	Available	JP
Latvia	Partner	Available	LV
Lithuania	Partner	Available	LT
Malaysia	Partner	Available	MY
Mexico	Microsoft	Available	MX
Netherlands	Microsoft	Available	NL
New Zealand	Microsoft	Available	NZ
Norway	Microsoft	Available	NO
Peru	Partner	Available	PE
Philippines	Partner	Available	PH
Poland	Partner	Available	PL
Portugal	Partner	Available	PT
Russia **(on-premises only)**	Microsoft	Available	(Not listed)
Serbia	Partner	Available	RS
Singapore	Partner	Available	SG
Slovenia	Partner	Available	SI
South Africa	Partner	Available	ZA

COUNTRY/REGION	LOCALIZED BY	AVAILABILITY	ENVIRONMENT COUNTRY IN TENANT ADMIN CENTER
South Korea	Partner	Available	KR
Spain	Microsoft	Available	ES
Sweden	Microsoft	Available	SE
Switzerland	Microsoft	Available	CH
Taiwan	Partner	Available	TW
Thailand	Partner	Available	TH
United Arab Emirates	Partner	Available	AE
United Kingdom	Microsoft	Available	GB
United States	Microsoft	Available	US
Vietnam	Partner	Available	VN

Important information regarding planned availability

Due to the nature of the joint effort where Microsoft is providing the international (W1) cloud service and partners are providing localization app(s), the above planned availability estimates represent the earliest possible availability and can be exceeded if localization partners have not successfully published their app(s) to AppSource by this time.

Supported languages

To maximize productivity Dynamics 365 Business Central supports many languages. It is important to know that language support requires translation of *platform* captions and *application* captions (UI). Translated platform captions are provided by Microsoft for all supported languages. Application languages are provided by both Microsoft and partners, depending on the language.

Application languages delivered by Microsoft are provided as language apps, available on AppSource, that can be installed if needed. Partners can also provide application translation for any platform supported language. To use one of these languages the app publisher must be contacted through AppSource.

This table gives an overview of supported languages and how application languages are provided.

LANGUAGE	LOCALE	TRANSLATION APP PROVIDED BY
Bulgarian	bg-BG	Partner, Available on AppSource (pending)
Czech (Czech Republic)	cs-CZ	Microsoft, Available on AppSource
Danish (Denmark)	da-DK	Microsoft, Available on AppSource
German (Austria)	de-AT	Microsoft, Available on AppSource

LANGUAGE	LOCALE	TRANSLATION APP PROVIDED BY
German (Switzerland)	de-CH	Microsoft, Available on AppSource
German (Germany)	de-DE	Microsoft, Available on AppSource
English (Australian)	en-AU	Microsoft, Available on AppSource
English (Canada)	en-CA	Microsoft, Available on AppSource
English (Great Britain)	en-GB	Microsoft, Available on AppSource
English (New Zealand)	en-NZ	Microsoft, Available on AppSource
English (United States)	en-US	Built-in
English (South Africa)	en-ZA	Supported by platform, no app available
Spanish (Spain)	es-ES	Microsoft, Available on AppSource
Spanish (Mexico)	es-MX	Microsoft, Available on AppSource
Estonian (Estonia)	et-EE	Partner, Available on AppSource
Finnish (Finland)	fi-FI	Microsoft, Available on AppSource
French (Belgium)	fr-BE	Microsoft, Available on AppSource
French (Canada)	fr-CA	Microsoft, Available on AppSource
French (Switzerland)	fr-CH	Microsoft, Available on AppSource
French (France)	fr-FR	Microsoft, Available on AppSource
Croatian (Croatia)	hr-HR	Partner, Available on AppSource
Hungarian (Hungary)	hu-HU	Partner, Available on AppSource
Icelandic (Iceland)	is-IS	Microsoft, Available on AppSource
Italian (Switzerland)	it-CH	Microsoft, Available on AppSource
Italian (Italy)	it-IT	Microsoft, Available on AppSource
Japanese (Japan)	jp-JP	Partner, Available on AppSource
Korean (Korea)	ko-KR	Partner, Available on AppSource
Latvian (Latvia)	lv-LV	Partner, Available on AppSource

LANGUAGE	LOCALE	TRANSLATION APP PROVIDED BY
Lithuanian (Lithuania)	lt-LT	Partner, Available on AppSource (pending)
Norwegian (Norway)	no-NO	Microsoft, Available on AppSource
Dutch (Belgium)	nl-BE	Microsoft, Available on AppSource
Dutch (Netherlands)	nl-NL	Microsoft, Available on AppSource
Polish (Poland)	pl-PL	Partner, Available on AppSource
Portuguese (Brazil)	pt-BR	Partner, Available on AppSource (pending)
Portuguese (Portugal)	pt-PT	Partner, Available on AppSource
Romanian	ro-RO	Partner, Available on AppSource (pending)
Russian (Russia)	ru-RU	Microsoft, Available on AppSource
Slovak (Slovakian)	sk-SK	Partner, Available on AppSource (pending)
Slovenian (Slovenia)	sl-SI	Partner, Available on AppSource
Serbian (Serbia)	sr-latn-RS	Partner, Available on AppSource
Swedish (Sweden)	sv-SE	Microsoft, Available on on AppSource
Turkish (Turkey)	tr-TR	Partner, Available on AppSource (pending)
Traditional Chinese (Hong Kong SAR)	zh-HK	Partner, Available on AppSource
Traditional Chinese (Taiwan)	zh-TW	Partner, Available on AppSource
Vietnamese (Vietnam)	vi-VN	Partner, Available on AppSource

NOTE

Application translations provided by Microsoft cover features in the international version (W1). Local functionality is not translated to all languages and will be provided in English and native language, as has been done historically. As a result, you may experience a mix of translations if using Dynamics 365 Business Central in a foreign language.

See Also

Rules and Guidelines for AL Code
Checklist for Submitting Your App

Delegated Administrator Access to Business Central Online

2/17/2021 • 5 minutes to read • Edit Online

As a Business Central reselling partner, you must set up your employees to work in Partner Center, and you must assign employees to support your customers. When you request a reseller relationship with a customer, you can choose to include *delegated administration* privileges for Azure Active Directory (Azure AD) and Office 365 in the request email that you send to the customer.

You must already have set up users in your own tenant in Partner Center so that the **Assists your customers as** field specifies the relevant role for this user to be able to login in to your customers' Business Central environments as either *Admin agent* or *Helpdesk agent*. These roles are used when the customer accepts the relationship, so you can assign the right people to the customer's Azure AD tenant.

When a customer grants the delegated administration privilege to a partner:

- The **Admin Agent** group is assigned to the **Global Administrator** role in the customer's Azure AD tenant.
- The **Helpdesk Agent** group is assigned to the **Helpdesk Administrator** role in the customer's Azure AD tenant.

Based on the roles assigned, members of both groups can sign in to the customer's Azure AD tenant, Microsoft 365 services, Business Central administration center, and Business Central tenants by using their partner credentials. For more information, see Delegated admin privileges in Azure AD in the Partner Center documentation.

For certain tasks, you can access the Business Central administration center, which is a powerful tool for you to manage your customers' tenants. From the administration center, you can manage upgrades and access the tenants as the delegated administrator. For more information, see The Business Central Administration Center.

> **TIP**
> Always include the domain or the Azure Active Directory ID of the customer in the URL when you log in as a *delegated admin*, such as in `https://businesscentral.dynamics.com/contoso.com/admin`. This way, you always know exactly which customer you are trying to access.

Caution

In the Microsoft 365 admin center and Microsoft Azure Management portal, both customers and partners can invite external users (guests) into their Active Directory. When a partner user is added as a guest to the customer's Azure AD, they can no longer log in as a delegated admin into the customer's Business Central. In order to log in, the local user (guests or native) must have a valid Business Central license assigned to them.

Restricted access to Business Central as delegated administrator

When you sign in to your customers' Business Central as the *delegated administrator* from the Business Central administration center, you have access to all areas of their Business Central. However, because you are not registered as a regular user, there are certain tasks that you cannot do.

The following tasks are *not* available to the delegated administrator:

- Set up jobs to run as scheduled tasks in the job queue

- Use the **Edit in Excel** action or interact with Business Central data in Excel using the Business Central add-in for Excel.

  You can still use the **Open in Excel** action to view data in Excel.

- Use the **Invite External Accountant** assisted setup guide

  Instead, you can add the external user in the Azure portal and assign this user the **External Accountant** license.

- Change the experience to Premium

- Use the **Cloud Migration Setup** assisted setup guide to migrate data from Business Central on-premises to Business Central online

  Instead, a user who is assigned the SUPER permission set in Business Central can run the assisted setup guide.

- Access a web service by using a Web Service Access key.

Managing delegated permissions as a partner

Delegated administrators are not visible in the customer's Azure AD user list and cannot be managed by the customer's internal admin. However, when a delegated admin logs into a Business Center environment on behalf of a customer, they are automatically created as a user inside the Business Central environment. This means that the actions performed by a delegated admin are logged in Business Central, such as posting documents and, associated with their user ID.

If a customer removes delegated permissions from you, you can still manage their subscription from the Partner Center, such as adding or removing licenses for their subscription, but you will no longer be able to log into and manage their Business Central environment, Azure AD, and other services. You will also not be able to manage their users (add/remove/assign licenses) from the **Customer** page in the Partner Center.

Managing delegated permissions as an internal administrator

As a Microsoft customer organization, you can have multiple partners registered as your resellers. It is not unusual for a single organization to use one partner as the delegated admin for their Microsoft 365 subscription and another for Business Central, for example. However, as soon as the delegated administration right is granted in the Microsoft 365 admin center, you cannot restrict partner access to a specific service only. The delegated admin access applies to all Microsoft services that your organization subscribes to.

If you do not need delegated admin help continuously, you can restrict access for the partner users into your environment. There are two approaches that you can use to restrict delegated admin access to a Business Center environment:

- Disable a specific delegated admin user within the Business Central environment. For more information, see How to remove a user's access.
- Revoke delegated administration rights from all partner users at once in the Microsoft 365 admin center, without breaking the reseller relationship with the partner.

In the Microsoft 365 admin center, internal administrators can find information about their partner relationships in the Settings/Partner Relationship menu. On the same page, you can remove delegated permissions from the partner, to restrict their access to Business Central and other services, while still keeping the reseller relationship with them.

If you then want to allow access to your environment again, you can ask the partner to share the "Request a reseller relationship" invitation link with you again.

For more information, see Customers delegate administration privileges to partners.

See also

Administration of Business Central Online
Get Started as a Reseller of Business Central Online
Exporting Databases

Licensing in Dynamics 365 Business Central

2/17/2021 • 3 minutes to read • Edit Online

Business Central licenses can only be purchased through CSP. Microsoft offers several types of paid licenses (users):

- Essential
- Premium
- Team Member
- External Accountant

Customers can also subscribe for an evaluation version by using self-service sign-up (also known as IW or viral sign-up). This subscription comes with 10000 licenses and allows customers to evaluate the functionality of Business Central using non-production companies.

Business Central does not use the classic Dynamics NAV license files (*.flf). Instead, permissions are generated based on entitlements.

We define license permissions (per object) in the **Entitlements** table. Entitlements are grouped in the **Entitlement Set** table, and then each entitlement set is associated with one of the four Azure Active Directory (Azure AD) service plans.

This means that when a user purchases, for example, an Essential license and tries to sign in to Business Central, we retrieve the user's service plan (in this case Essential) from Azure AD and then load the corresponding entitlements as license permissions.

> **NOTE**
>
> For more information about the different types of licenses and how licensing works in Business Central, see the Dynamics 365 Licensing Guide, which you can download from https://go.microsoft.com/fwlink/?LinkId=866544.

Entitlements and user groups

Entitlements are permissions that describe which objects in Business Central a customer is entitled to use according to their Azure Active Directory role or the license they purchased from Microsoft.

The **User Group Plan** table stores the mapping between the service plans and the user groups. Based on this mapping, the service determines which user group is assigned to a user by default when a user logs in to Business Central for the first time. The user group assigns a specific license (or makes the user a member of a specific Azure AD role).

When a user logs in to Business Central, the service applies the intersection of the entitlements that are associated with the user's service plan (or Azure AD role) and the permissions that are defined for that user. Entitlements always have higher priority over permissions. For example, even is the user is given SUPER permissions by the admin but has the Team Member license assigned – the user can still only access the objects defined by the Team Member entitlements.

You can verify how entitlements, licenses, and user groups work together by looking at the **Effective Permissions** page, which you can access from the **User** page. For more information, see Create Users According to Licenses in the business functionality content for Business Central.

Entitlements in production and sandbox environments

In versions earlier than 2020 release wave 2 (version 17.0), entitlements are only enforced in production environments so that prospective and existing customers could explore the functionality of the Premium subscription in a sandbox environment without having to purchase a Premium license.

However, partners that create and test their extensions using sandbox environments of that type could miss errors in their code that would be related to entitlements or access, so that those issues would not be found until they deployed their extensions to the production environments.

Also, you can explore such functionality by creating an evaluation company in either a production or a sandbox environment using the free evaluation subscription that is available from https://dynamics.microsoft.com/en-us/business-central. Alternatively, you can use the **Business Central Premium - Trial** subscription that is available through the CSP program as described here. By using these subscription types, customers and partners can explore the capabilities included with the Premium subscription.

With 2020 release wave 2, the license checks for entitlements are now also enforced in the sandbox environments.

Reassigning licenses

The licensing terms in the Online Services Terms document do not allow temporary assignment of a license that belongs to one user to another user.

Most, but not all, licenses can be reassigned. Except as permitted in this paragraph or in the Online Service-specific Terms, the customer cannot reassign a license on a short-term basis, meaning within 90 days of the latest assignment. The customer can reassign a license on a short-term basis to cover a user's absence or the unavailability of a device that is out of service. Reassignment of a license for any other purpose must be permanent. When a customer reassigns a license from one device or user to another, they must block access and remove any related software from the former device or from the former user's device.

See Also

Get Started as a Reseller of Business Central Online
Deployment of Dynamics 365 Business Central
Working with Sandboxes and Entitlements
Embed App Overview

Operational Limits for Business Central Online

2/17/2021 • 6 minutes to read • Edit Online

To ensure the availability and quality of Business Central services, there are limits on certain operations. This article describes the limits and, in some cases, the strategy behind them.

> **TIP**
> Telemetry is gathered on some of the operations that have a limit. The telemetry provides insight into operations for which limits were exceeded. For more information, see Monitoring and Analyzing Telemetry.

Client connection limits

SETTING	DESCRIPTION	LIMIT
Reconnect period	The time during which a client can reconnect to the service after being disconnected.	10 minutes

Data handling limits

SETTING	DESCRIPTION	LIMIT
Max items in object graph	The maximum number of objects to serialize or deserialize.	10,000
Max file size	The maximum size of files that can be uploaded to or downloaded from the service.	350 MB
Maximum stream read size	The maximum number of bytes that can be read from a stream (InStream object) in a single AL read operation. Examples include READ or InStream.READTEXT method calls. This setting pertains to UTF-8 and UTF-16 text encoding; not MS-DOS encoding.	1,000,000 bytes

Database connection limits

SETTING	DESCRIPTION	VALUE
Search timeout	The time (in seconds) that a search operation on lists in the client continues before it's stopped. When the limit is reached, the following message displays in the client: **Searching for rows is taking too long. Try to search or filter using different criteria.**	10 seconds

SETTING	DESCRIPTION	VALUE
SQL command timeout	The contextual time-out for a SQL command.	30 minutes
SQL connection idle timeout	The time that a SQL connection can remain idle before being closed.	5 minutes
SQL connection timeout	The time to wait for the service to connect to the database. When the time is exceeded, the attempt is canceled and an error occurs. This setting also applies to begin, rollback, and commit of transactions.	1.5 minutes
Long running SQL query threshold	The amount of time that an SQL query can run before a warning telemetry event occurs. If this threshold is exceeded, the following event is logged: Action completed successfully, but it took longer than the given threshold.	1000 ms

Asynchronous task limits

SETTING	DESCRIPTION	LIMIT
Maximum concurrent running scheduled tasks	The maximum number of tasks that can run simultaneously for an environment. If there are many jobs running at the same time, you might experience that the response time for clients gets slower. If the value is too low, it might take longer for scheduled tasks to process.	3
Page background task default timeout	The default amount of time that page background tasks can run before being canceled. Page background tasks can be also given a timeout value when enqueued at runtime. This limit is used when no timeout is provided when the page background task is enqueued.	2 minutes
Page background task max timeout	The maximum amount of time that page background tasks can run before being canceled. Page background tasks can be also given a timeout value when enqueued at runtime. If a page background task is enqueued with a timeout greater than this limit, this limit is ignored.	10 minutes

SETTING	DESCRIPTION	LIMIT
ChildSessionsMaxConcurrency	The maximum number of child sessions that can run concurrently per parent session. When the value is exceeded, additional child sessions will be queued and run when a slot becomes available as other child sessions are finished.	5
ChildSessionsMaxQueueLength	The maximum number of child sessions that can be queued per parent session. If the value is exceeded, an error occurs.	100

Report limits

SETTING	DESCRIPTION	LIMIT
Default max documents	The maximum number of documents that can be merged in report by default Users can override this setting on a report-basis from the report request page. If exceeded, the report will be canceled. Developers can override this setting by using MaximumDocumentCount property of a report. Client users can do the same when running a report from the report request page	200
Max documents	The maximum number of documents that can be merged in report. If exceeded, the report will be canceled.	500
Default max execution timeout	The maximum execution time that it can take to generate a report by default. Users can override this setting on a report-basis from the report request page. If exceeded, the report will be canceled. Developers can override this setting by using the ExecutionTimeout property of a report. Client users can do the same when running a report from the report request page.	6 hours
Max execution timeout	The maximum execution time that it can take to generate a report. If exceeded, the report will be canceled.	12 hours

SETTING	DESCRIPTION	LIMIT
Default max rows	The maximum number of rows that can be processed in a report by default. Users can override this setting on a report-basis from the report request page. If exceeded, the report will be canceled. Developers can override this setting by using the MaximumDataSetSize property of a report. Client users can do the same when running a report from the report request page.	500,000
Max rows	The maximum number of rows that can be processed in a report. If exceeded, the report will be canceled by the server.	1,000,000

Query limits

SETTING	DESCRIPTION	LIMIT
Max execution timeout	The maximum execution time that it can take to generate a query. If exceeded, the query will be canceled.	30 minutes
Max rows	The maximum number of rows that can be processed in a query. If exceeded, the query will be canceled.	1,000,000

OData request limits (per environment)

SETTING	DESCRIPTION	LIMIT
Max concurrent requests	The maximum number of OData V4 requests the server instance can actively process at the same time. Requests that exceed the limit will wait in the queue until a time slot becomes available.	5
Max connections	The maximum number of simultaneous OData requests on the server instance, including concurrent and queued requests. When the limit is exceeded, a 429 (Too Many Requests) error occurs.	100
Max page size	The maximum number of entities returned per page of OData results.	20,000 entities per page
Max request queue size	The maximum number of pending OData V4 requests waiting to be processed. When the limit is exceeded, a 429 (Too Many Requests) error occurs.	95

SETTING	DESCRIPTION	LIMIT
Rate	The number of OData requests per minute that are allowed. An HTTP response code `429 - Too Many Requests` is returned if limits are exceeded.	Sandbox: 300 requests/minute Production - 600 requests/minute
Operation timeout	The maximum amount of time that the service gives a single OData request. When the limit is exceeded, an HTTP response code `408 - Request Timeout` is returned. After 8 minutes, the session is canceled.	8 minutes

SOAP request limits (per environment)

SETTING	DESCRIPTION	LIMIT
Max concurrent requests	The maximum number of SOAP requests the server instance can actively process at the same time. Requests that exceed the limit will wait in the queue until a time slot becomes available.	5
Max connections	The maximum number of simultaneous SOAP requests on the server instance, including concurrent and queued requests. When the limit is exceeded, a `429 (Too Many Requests)` error occurs.	100
Max message size	The maximum permitted size of a SOAP web service requests	65,536 KB
Max request queue size	The maximum number of pending SOAP requests waiting to be processed. When the limit is exceeded, a 429 (Too Many Requests) error occurs.	95
Rate	Specifies how many SOAP requests per minute are allowed. An HTTP response code `429 - Too Many Requests` is returned if limits are exceeded.	Sandbox: 300 requests/minute Production: 600 requests/minute
Operation timeout	The maximum amount of time that the service gives to a single SOAP request. When the limit is exceeded, HTTP response code `408 - Request Timeout` is returned.	8 minutes

See Also

Working with API Rate Limits
Microsoft API Terms of Use

Major Updates of Business Central Online

2/17/2021 • 14 minutes to read • Edit Online

This article provides an overview of what you need to know about how a major Business Central update rolls out. It includes key dates, actions you need take, and answers some common questions.

Timelines for major updates

The following figure illustrates the key milestones and dates for rolling out a major update. The timeline and dates are loosely based on 2020 release wave 2. The same timeline applies to all other major updates, though dates will differ.

The following table describes the milestones with example dates for the two release waves in any given calendar year.

MILESTONE	EXAMPLE DATE WAVE 1	EXAMPLE DATE WAVE 2	DESCRIPTION
Update is available	April 1	October 1	The date when the new major version of Business Central becomes generally available
Update starts rolling out	April 15	October 15	The default date when Microsoft starts upgrading your environments. Once the update is scheduled, you can change that date, within allowed date range, to a date, which fits you better.
Last scheduled update date	May 31	November 30	The last date you can choose to extend your upgrade date to, typically 30 days after the update is available but extended to 60 days for 2020 release wave 2.

IMPORTANT

As announced in the blog post, in response to COVID-19, Microsoft made some changes to the update schedules in April, 2020. Specifically for Business Central, existing customers were given 60 days to upgrade after the new version was made generally available. The extended 60 days update window also applies to 2020 release wave 2 updates.

Dates and times differ significantly across countries and regions. Make sure that you have set up notifications in the Business Central administration center so that you're notified when the next major update is available.

Update is available

In the release plans, you can see when the next major update is generally available. Typically, the update is announced on the Dynamics 365 blog, and Microsoft starts rolling out the update to existing environments. On the same day, new customers signing up for a trial and all newly created environments (sandboxes and production) are directed to the new version.

The existing environments are scheduled to be updated to the new version gradually across the world. Microsoft strives to schedule all environments soon after the official release date, but for some environments, the update might not be available for a few weeks. In extreme cases, a given environment might be scheduled for update up to one month after the release date.

When the update becomes available for an environment, all notification recipients for that environment receive email notifications. A notification about the update availability is also shown in the Business Central administration center itself. Starting this day, you can set the date for when your environment should be updated via the Business Central administration center (schedule update). You can choose any date between that date and the date that is shown as the last available date for the update. The last available date can be between one and four weeks away. For more information, see Managing Major and Minor Updates of Business Central Online.

Behind the scenes

Starting from the official release date, Microsoft begins scheduling updates. Scheduling doesn't occur for all environments around the world simultaneously. During scheduling, Microsoft sets a default update date for each environment, typically 14 days in the future from the date when you received the notification. You can see this date in the environment details page in the Business Central administration center.

> **IMPORTANT**
>
> If you don't change the default date, **Microsoft updates the environment automatically** any day between the scheduled update date and the date that is shown as the last possible update date in your e-mail notification. If you don't want your environment to be updated automatically, change the update date to the one that fits you better.

You can change the default date set by Microsoft at any point of time, including changing it to the current date. In some cases, Microsoft can suspend or postpone all updates beyond this last possible date. For more information about what happens in such cases, see Postponed updates.

When the scheduled update date arrives, the update runs automatically within the update window that you've specified for this environment. All users will be disconnected from this environment, and all sign-in attempts during the update will be blocked with the message `Service is under maintenance`. We strongly encourage that you set an update window for all production environments so that updates don't start during business hours. For more information, see Set the update window for each environment.

> **NOTE**
>
> When you select a current date for your update, but the update window defined for this environment has already passed, the update will start within that time window, but on the following day to the one you defined for your environment. For example, if you're changing the Scheduled update date to the current date at 6pm, and your update window is set to 1 AM - 7 AM, the update will not start immediately, but around 1 AM on the next day.

Environments fail to update for various reasons, such as per-tenant extension compatibility issues, AppSource app compatibility issues, or internal update issues. Any environment that fails to update will be automatically restored to the original application version. Within one hour, the environments are automatically rescheduled for new update attempts for seven days. If you consider the issue resolved and want to try the update again, change the date to an earlier date or the current date.

If Microsoft can't do the update on the selected date, you'll be notified by email that the environment is rescheduled to be updated seven days later. You can change that date in the Business Central administration center to any other allowed date, including the current date.

Delayed scheduling of updates

In some cases, even after the update is available in your area, you may still not be able to set the update date (schedule environment update). This condition can happen for one of the following reasons:

- Your environment has not yet updated to the latest minor update of the previous version of Business Central. All environments must be updated to the last available minor update of the previous version of Business Central before they can be scheduled to be updated to the next major version. You can check the version information in the **Troubleshooting** section of the **Help and Support** page in Business Central as well as in the **Version Management** section in Business Central administration center. For more information, see Version numbers in Business Central.

  Microsoft is actively working on updating all environments to the latest minor update as soon as possible. In most cases, your environment will be scheduled for the major update soon, and you'll get a chance to change the date to the one that fits you better. In the unlikely situation that your environment is updated to the last minor update around or even after the last selectable update date, you'll still get a least seven days to schedule the update.

- You have just created a new sandbox environment as a copy of your production environment. In this case, the sandbox environment is created on the same version as the production environment it was copied from. If your newly created sandbox environment is running on the last minor update of the previous version, Microsoft will schedule it for update automatically within one hour.

  You'll receive email notification and will see the notification in Business Central administration centerwhen it happens. The scheduled update date for this environment will be set to seven days from the current date, so that you have enough time to change the date to one that fits you better (including the current date).

- Your per-tenant extensions are not compatible with the next major update.

  Before rolling out the next major update, as well as during the update, Microsoft routinely checks per-tenant extensions in all existing environments for compatibility with the next major update. When compatibility issues with the upcoming version are detected, email notifications that describe the detected issues are sent to the notification recipients.

  Ifyou discoverany such issues, apply the changes to your solution as usual using Visual Studio Code, and test the new appin a sandbox environment that runson the new major version (either in preview or officially available). If testscomplete successfully, upload the new app version into your production environment in the **Extension Management** page, setting the**Deploy to**field to**Nextmajorversion**.This way the compatible version of your app will beusedwhenyour environment is updated.For more information, see Deploying a Tenant Customization.

- The AppSource apps that are installed in your environment are not yet available for the next major version of Business CentralBusiness Central. While the AppSource apps are normally kept up-to-date by the partners who own them, it can happen that a particular app needs more time to prepare for the next major update and is not yet available for it. In this situation, please contact the app owner to understand their availability plans.

Postponed updates

In critical circumstances, Microsoft can decide to postpone the rollout of the updates, such as if a critical issue is discovered in the new major version that is being rolled out. While Microsoft is working on addressing the issue,

the updates will be postponed. You'll receive email notification about this, and you'll see the notification displayed in the Business Central administration center. The **Version Management** section for each environment will show the update rollout state as *Postponed*.

Not knowing the nature of the issue and the solution in advance, we can't predict when the updates will resume again. This means that neither the email nor the notification in Business Central administration center will contain the information about the expected resume date. Microsoft will be actively working on resuming updates as a matter of highest priority once the issue is addressed. You'll receive another email notification when updates have been resumed. The last available date will be prolonged by the number of days the update was postponed.

If it happens that you schedule the update of your environment on a date when the updates are postponed, your update will not be done. Microsoft will not send separate notification about this. You can reschedule the update to a later date, or you can wait until you have received the email notification that the updates have been resumed and schedule the update at that time. All environments that missed their scheduled update date will be rescheduled automatically to run the update within seven days from the date the updates were resumed, but you can change that date to any other allowed date, including the current date.

If you did not explicitly set a date for your environment update in the Business Central administration center, this environment will be picked up for updating automatically, shortly after the updates have been resumed. The update will of course still be executed within the specified update time window.

Prepare, test, and learn before the major update

You can prepare yourself, users, and any customizations by trying out the new version before your production environment is updated. You can do this in different ways as explained in the following sections.

Prepare for major updates by enabling select features earlier

Some new features can be enabled ahead of time on sandbox and production environments, giving you time to test and prepare for change. Most times, you can enable features weeks before preview environments for the major update are available.

When Microsoft releases features or feature design improvements as part of minor updates, some of these features are optional until the following major update. Administrators can turn these optional features on or off from the **Feature Management** page.

For more information, see Feature Management.

Prepare for major updates with preview environments

About one month before a major update, you can try out new functionality in preview environments. Preview environments are essentially Business Central online sandbox environments that you create on a preview version of the application. When you create the new sandbox environment, choose the preview version marked as *(Preview)* from the version list. This way, you get a new sandbox environment with a preview version of the application.

You must have access to Microsoft Collaborate in order to submit your feedback and report any potential issues that you discover in the preview version of the application. For more information about getting access to Collaborate, see Step 4: Getting access to preview bits.

> **NOTE**
> Previews roll out gradually across the world, so if the option is not showing up for you today, please try again tomorrow.

The newly created preview sandbox environment contains demonstration company data. Trying the preview on a copy of your current production data is not yet supported; nor is testing the upgrade from your current

version to the preview. However, you can use the newly created sandbox environment for exploring and learning the new product capabilities, as well as validating that your per-tenant extensions are still working as expected.

If you run your tests on a preview environment one month before the announced major release of Business Central, it is more likely that the coming updates of your production environments will go smoother. This way, you, your customers, and your code are better prepared for the official release.

We expect to update the preview version only if we discover critical issues before the major update is generally available for production environments. Apart from these potential fixes, we do not expect any further changes to the product between the preview and the official release. This means that you can start your testing and learning activities immediately, without waiting for the official release.

> **NOTE**
> You will be able to test the update on a copy of your production data in a sandbox environment when we release the new update in production in April or October, respectively. When the official release becomes available, you can continue your tests on that version. You will no longer be able to create new preview sandboxes.

> **IMPORTANT**
> The preview version as well as all sandbox environments that are based on it will be removed 30 days after the official release becomes available.

For more information, see Prepare for major updates with preview environments.

Prepare for major updates just before the production environment is updated

Starting on the date when you're notified that the new major update is available, you can test the new version by using a sandbox environment that you then schedule to be updated. Start by copying your production environment into a sandbox on the same version as your production environment. All newly created environments are automatically included in the update process within one hour, so you'll receive email notification that the update is available, and you'll be able to schedule the newly created sandbox for update within one hour after it was created. By default, the newly created environments are scheduled to run the update within seven days from the date they were created, but you can change that date to any other allowed date, including the current date.

If you change the update date to the current date, the update will start within the closest available update time window you specified for the environment. If you want to start the update of your sandbox environment immediately, you can set the update time window for this sandbox environment to be 24 hours.

If any errors are detected during the update, you'll receive email notification that describes the detected issues.

Any environments that fail to update due to per-tenant extension compatibility issues or any other issues will be automatically restored to the original application version. Within one hour, they are automatically rescheduled for another update attempt. Scheduled update date is again set to seven days in the future. If you address the compatibility issues earlier, you can change the date to an earlier date, including the current date. This pattern repeats until your environment is updated successfully.

Overview of the timeline for preparing for the next major update

The following table describes the suggested milestones with example dates for the two release waves in any given calendar year.

MILESTONE	EXAMPLE DATE WAVE 1	EXAMPLE DATE WAVE 2	DESCRIPTION

MILESTONE	EXAMPLE DATE WAVE 1	EXAMPLE DATE WAVE 2	DESCRIPTION
Previews are available	March 1	September 1	You create a new sandbox environment based on the new preview for test purposes. For more information, see Prepare for major updates with preview environments.
A few days before you know that the update is announced	March 31	September 30	You create a new sandbox environment based on your existing production environment (copy your production environment into a sandbox) and wait for it to be updated to the new version.
Update available in your region	April 6	October 5	The major update is made available. You're notified about it via e-mail. Go to the Business Central administration center and set the update date for your sandbox environment to the current date. The sandbox will be updated within the closest available update time window that you set for it. Now, you can test your existing extensions and your production data against the new version. We recommend that you set the update date for your production environment a few days or weeks in the future, to help make sure that it does not get updated automatically before you had a chance to test the new version and your extensions in your sandbox environment.
Preview sandboxes are deleted	May 1	November 1	30 days after the new major update is announced, the preview sandboxes are deleted. There will be no option to keep these sandboxes or export data from them.

See also

Managing Major and Minor Updates of Business Central Online
Working with Administration Tools
The Business Central Administration Center

Version numbers in Business Central

2/17/2021 • 2 minutes to read • Edit Online

Business Central online and on-premises consists of different components that must work together. If you are a an enduser, this doesn't matter in the course of your normal work day. But if you are an administrator, knowing the version numbers is important for troubleshooting, development, and on-premises upgrade scenarios.

You can use the information about which version the tenant is on to help you troubleshoot an issue that a user has reported, for example. This information is listed in the **Troubleshooting** section of the **Help and Support** page in Business Central in the following format:

VERSION	EXAMPLE	DESCRIPTION
Platform <major>.<minor>.<build>.<revision>	16.0.12345.0	Specifies the full platform version, which includes client and server components.
Application <major>.<minor>.<build>.<revision>	16.1.23456.0	Specifies the full version number for the application, including the major version number and build number.

In the Business Central administration center, the version information is rendered slightly differently:

VERSION	EXAMPLE	DESCRIPTION
Platform <major>.<minor>	16.1	Specifies the major and minor version of the platform, which includes client and server components.
Application <major>.<minor>.<build>.<revision>	16.1.23456.0	Specifies the full version number for the application.

The numbers are updated based on Microsoft's builds. In the default version of Business Central online, platform and application have the same major version number but different build numbers. If you perform a technical upgrade of Business Central on-premises, then platform and application will have different versions.

The following list describes the meaning of each of the numbers in a full version number:

- `major` is the major version of Business Central
  - `16` is the Business Central 2020 release wave 1 update in April 2020 and forward
  - `15` is the Business Central 2019 release wave 2 update in October 2019 and forward
  - `14` is the Business Central April 2019 release
  - `13` is the Business Central October 2018 release
  - `12` is the April 2018 launch of Business Central
- `minor` is the monthly update number, such as 0, 1, or 5.
- `build` is the five digit build number, such as 23456.
- `revision` is set to 0 for the original release and can remain at 0. However, if the tenant is patched with a hotfix, then that build number can be applied.

In other words, if you see a version number such as `16.1.23456.26323`, then it means major version *16*, update number *1*, build number *23456*, and hotfix number *26323*.

The same version numbers are used to identify artifacts for running Business Central on Docker.

See Also

Managing Technical Support
Installing a Cumulative Update
Administration of Business Central Online

Enabling Upcoming Features Ahead of Time

2/17/2021 • 7 minutes to read • Edit Online

This document describes how administrators can turn on new features using the Feature Management page.

About Feature Management

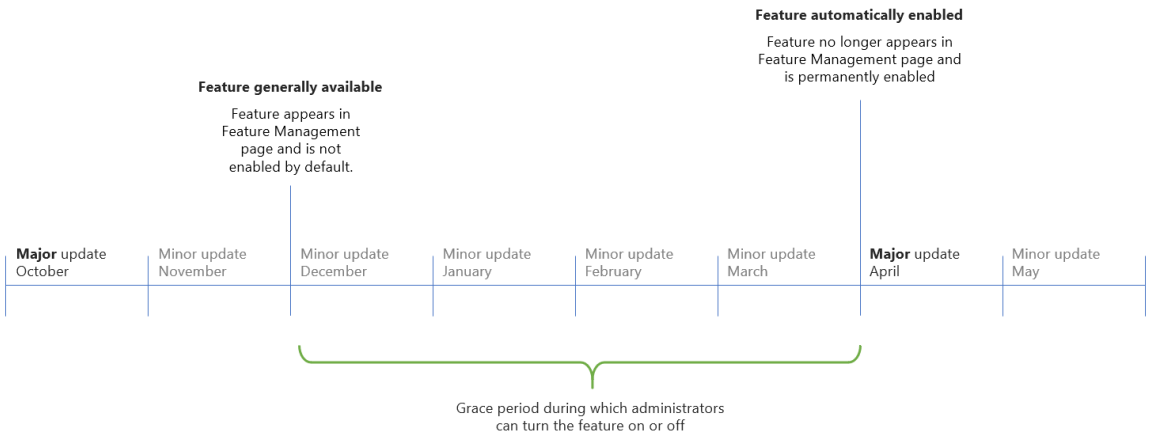
Some new features can be enabled ahead of time on sandbox and production environments. This capability allows you to benefit as early as possible from feature improvements and innovative new features. It gives you the time you need to test and prepare your organization for change.

When Microsoft releases features or feature design improvements as part of minor updates, some features aren't immediately enabled. Administrators can learn about these features and independently enable each feature from the **Feature Management** page. Once a feature is enabled, it becomes available for all users on that environment no matter how they access Business Central.

These features are only optional for a while. The option period typically starts from the minor update in which they're made generally available. The period ends when the features become mandatory and are automatically enabled in a later major update. To see the approximate date and service update when each feature is expected to become mandatory, see the **Automatically enabled from** field in the **Feature Management** page. After this date, the feature will no longer appear in the Feature Management page and can no longer be turned off.

> **IMPORTANT**
>
> The projected timeline for a feature is subject to change (see Microsoft policy).

Example timeline for an optional feature



Learn about What's new and planned.
Learn about new features available in the last minor update.

TIP

To prepare for an upcoming feature, consider enabling the feature on a sandbox environment that has a copy of production data. Invite business users to test out the change using real-world tasks. Once you and your users are satisfied with the change, you can then enable it on production environments where they can immediately benefit from that feature.

How to enable an optional feature

1. Sign in to your environment and navigate to the **Feature Management** page, or use this link: https://businesscentral.dynamics.com/?page=2610.
2. If the page isn't editable, choose **Edit List** from the action menu.
3. For any row in the list, set the **Enabled for** field to *All users*.

As soon as you enable the feature, any user that signs in to that environment experiences the change. You won't necessarily experience the change yourself until you sign out and sign in again, or start a new session.

TIP

You can try out the feature for yourself without enabling it for all users by choosing the **Try it out** link. This will open a new browser tab with the feature enabled for that session. Any new sessions in your browser will also have the feature temporarily turned on. To stop trying the feature, close your browser window or sign out.

Features that can't be turned off

Some features or feature improvements may permanently affect the state and capabilities of Business Central and can't be safely reverted. These irreversible features can't be turned off again after they've been enabled. Before you enable an irreversible feature on a production environment, we recommend that you first enable and evaluate it on a sandbox environment that has a copy of production data.

NOTE

When you choose to enable an irreversible feature, a warning dialog that describes the consequences is displayed. Choose **Yes** to proceed with permanently enabling the feature on that environment.

Scheduling Data Updates for New Features

Enabling application features that change the user experience or update data can be a disruptive process, and you might want to go at your own pace by scheduling an update per company, for example, for a time that is after your users have been trained for the new experiences.

You can schedule a data update on the **Feature Management** page by choosing the **Schedule**action, or by choosing **All Users** in the **Enable for** column. Both of those actions start the **Feature Data Update** setup guide, which allows you to review the affected data and schedule the update process. When the data update process is completed, the feature is enabled in the company where you ran the data update.

NOTE

For a feature that requires data update, data is created based on the data for the existing feature. The data for the existing feature may remain available, however, it is not synchronized with data for the new feature. Therefore, we recommend that you use one feature or the other, but not both.

FAQ about Feature Management

There are no features listed as optional. Did I do something wrong?

There may be periods where no optional features have been made available, which is perfectly normal. There will likely be few or no features listed in the **Feature Management** page immediately after a major update.

Will all new features eventually be listed on the Feature Management list?

No. We carefully select applicable features based on different criteria so that only a manageable subset of new features will appear in the list. Selected features are primarily those features that change the visuals or behaviors of the user interface and which require significant effort for business users to adjust to.

Are these features still under development or in beta/preview?

No. Features listed in the Feature Management page are considered ready and generally available. Most of these features are automatically enabled on newly provisioned environments for new customers to benefit from.

Does Microsoft provide support for optional features?

Yes. Features that are listed in the **Feature Management** page are considered ready. They follow the standard support lifecycle for the service update in which they're first made available.

Will Business Central notify me closer to the date when a feature becomes mandatory?

No. Users and administrators don't receive any in-app or email notifications about approaching dates for features becoming automatically enabled.

Do these features show in the Microsoft 365 admin center Message center?

At this time, new Business Central features are not listed in Message center.

How is feature management different to the Early Access program?

The Early Access program that is used by some Dynamics 365 apps makes a large set of new features available two months before a major update. It allows customers to enable those features in production. The most significant difference is that the Early Access program features are always two months before the major update.

How is feature management different to preview environments?

Preview environments are Business Central online sandbox environments that include all new platform and application features that will later be made available with the major update. The most significant difference is that a preview environment includes all new features bundled together. You don't have the opportunity to select which feature to enable and test.

How is feature management different to Application Areas?

Application areas are a concept where developers specify differentiated user experiences in the business application. By using application areas, developers can show or hide individual controls on a page. Like feature management, application areas concept also puts administrators in control of selecting the preferred experience tier. One difference is that there's no time period during which application areas can be optionally enabled. Another difference is that they only apply to business application controls.

Can resellers, ISVs, and developers contribute to the list of features?

No. At this time, feature management is only for features released by Microsoft.

Can I enable a feature for a single user?

No. Business Central doesn't provide the ability to enable a feature for a single user or group of users. Enabled features apply to all users of an environment.

I don't see a link to try out an optional feature. Is something wrong?

Some features don't provide a way to try out the feature for yourself and won't display a **Try it out** link. Before you enable these features, we recommend you first enable and test the features on a sandbox environment that has a copy of production data.

Are optional features also optional on new environments?

Yes. Most optional features are enabled by default on new environments for new customers to benefit from. Administrators can still turn any of these features off from the Feature Management page. Some features are

irreversible and are not enabled by default.

Are optional features automatically enabled on sandbox environments?

When you create a new sandbox environment with a copy of production data, your choice of enabled features is also copied to the sandbox. When you create a fresh sandbox, each feature is enabled by default, unless a feature is irreversible.

Is feature management applicable to on-premises deployments of Business Central?

Yes. You can turn optional features on or off in a similar way.

See also

New and planned features
Administration of Business Central Online
Major updates of Business Central Online

Managing Technical Support

2/17/2021 • 12 minutes to read • Edit Online

If users report that they are having a problem with Business Central, superusers and the internal administrator can often find a solution. The internal administrator can find technical information in the **Help and Support** page as described in the following sections, and they can then escalate relevant issues to the reselling partner.

The reselling partner can log in to their customer's Business Central as the delegated admin for troubleshooting, such as by creating a sandbox environment based on production data, and then troubleshooting in that environment.

Both internal administrators and the reselling partner can use the Business Central administration center to manage environments and updates.

Internal administrators

As the internal administrator, you can work with users to identify solutions or workarounds, such as missing setup, missing permissions, and other issues in Business Central. If users are not sure if their Business Central is working as intended, they can check the Help and Learn content for the intended behavior. For more technical problems, administrators can find technical information in the **Help and Support** page to help with this investigation. For more information, see the Finding technical information section.

Internal administrators can also create sandbox environments for deeper troubleshooting, for example, before they decide to contact their partner for technical support. The partner must have specified their support contact details in the **Help and Support** page.

Delegated administrators

Each customer of Business Central has a partner who assists with technical support when requested by the internal administrator. As the partner, you must have specified support contact details in the **Help and Support** page. For more information, see Configuring the support experience.

> **IMPORTANT**
>
> You must have set up users in your own tenant in Partner Center as either *Admin agent* or *Helpdesk agent*, and they must have *delegated administration* privileges in your customer's Business Central to support the customer. For more information, see Delegated Administrator Access to Business Central Online.

The delegated administrator can access the customer's Business Central for further troubleshooting, and they can use the Business Central administration center to analyse telemetry, create a sandbox environment for debugging, or an extra production environment for step-by-step reproduction, for example.

If the partner cannot find a solution, they can request support from Microsoft. For more information, see the Escalating support issues to Microsoft section.

Extend trials

Another task for a delegated admin is to help with extending trials. For more information, see Extending trials.

Cleaning up settings

If you end the relationship with a customer, you must remove certain settings while you still have access to that customer's Business Central administration center. These settings include the following:

- Support contact details

  For more information, see To supply your support contact information in the administration center.

- Notification recipients

  For more information, see Tenant Notifications.

- Application Insights key (if this was set up by the partner)

  For more information, see Environment Telemetry in the Business Central administration center.

Finding technical information

For Business Central online, administrators have access to a range of tools for troubleshooting. Depending on the type of problem, administrators can troubleshoot in Business Central, or they can use the Business Central administration center to analyze telemetry, for example.

In this section, we provide an overview of the most useful tools for troubleshooting the problems that users are reporting. In many cases, administrators will want to create a sandbox environment based on the production environment where users are reporting problems. That way, the administrators can troubleshoot without disturbing normal work.

- The Help and Support page

  Each company in any Business Central environment has a **Help and Support** page that can be accessed from the the question mark in the top right corner. Here you can access the latest error message, which, for example, is useful if your users complain of a confusing error message such as *Sorry, we just updated this page. Please close and reopen.*. For more information, see The Help and Support page section.

- Page inspection

  Business Central includes a page inspection feature that lets you get details about a page, providing insight into the page design, the different elements that comprise the page, and the source behind the data it displays. For more information, see Inspecting and Troubleshooting Pages.

- Environment telemetry in the Business Central administration center

  In the Business Central administration center, you can view telemetry of top-level AL events, and any errors resulting from calls through the telemetry stack. For more information, see Environment telemetry.

- Troubleshoot in a sandbox environment

  In the Business Central administration center, you can create sandbox environments for safe debugging and troubleshooting. For more information, see Sandbox environments.

- Use the Event Recorder

  Using the Event Recorder, you can record the events that are published and raised while performing the actions of your scenario. For more information, see Discoverability of Events.

- Check the data in the database

  You can view table objects in Business Central. This lets you to see the data in all rows and columns of a specific table, including any columns that are added by table extensions. From the Business Central administration center, you can also launch a list of all tables, sorted by storage size. For more information, see Viewing Table Data.

- Analyze long running operations in Application Insights

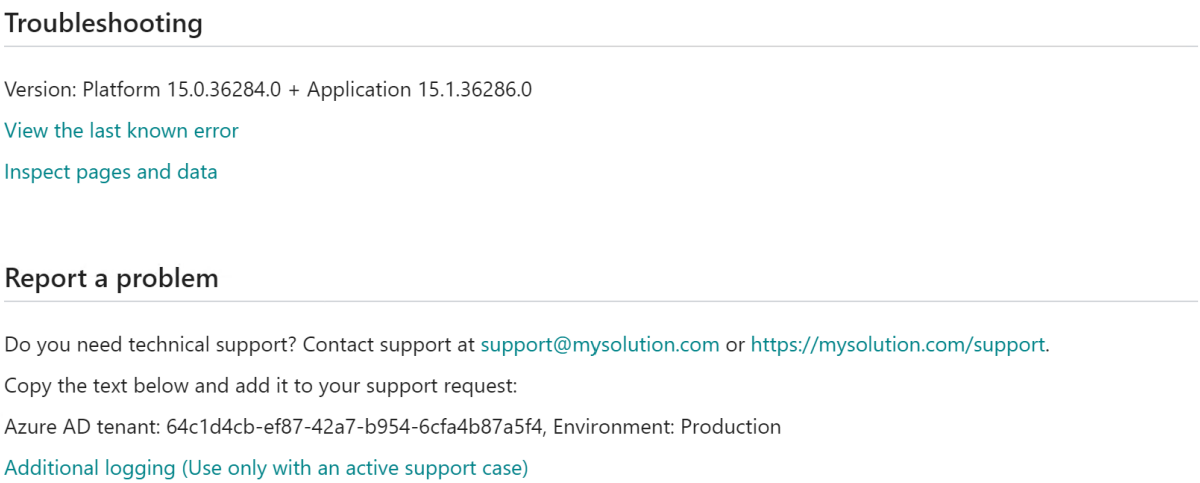
Set up Application Insights so that any SQL query that takes longer than 1000 milliseconds to execute will be sent to your Application Insights resource. For more information, see Analyzing long running operations in Application Insights.

- Debug your app or pre-tenant extension

  With Visual Studio Code and the AL Language extension you get an integrated debugger to help you inspect your code to verify that your application can run as expected. For more information, see Debugging in AL.

The Help and Support page in the Business Central company

The **Help and Support** page is a powerful tool for administrators to find technical information about Business Central, both online and on-premises. The **Troubleshooting** section gives easy access to the most recent error message, and it has a link to inspect pages for further troubleshooting.



Also in the **Help and Support** page, users can see support contact information, provided that this has been set up. For more information, see To supply your support contact information in the administration center. The **Help and Support** page also shows which version of Business Central, the specific environment is on.

For Business Central online, internal and delegated administrators also have access to this information in the Business Central administration center. You can use the Business Central administration center to easily navigate to the different environments in a tenant, and you can create sandbox environments that can help troubleshoot any issues reported by users. For more information, see The Business Central Administration Center.

Azure Active Directory tenant

When the internal administrator wants to contact the partner for support, then the **Help and Support** page encourages them to include information about their Azure Active Directory tenant ID in the email. This information is shown in the **Troubleshooting** section at the bottom of the **Help and Support** page.

The delegated administrator can use that to identify the tenant in the Partner Center and in the Business Central administration center for troubleshooting.

Version

You can use the information about which version the tenant is on to help you troubleshoot the issue that the customer has reported, for example. This information is also listed in the **Troubleshooting** section of the **Help and Support** page. For more information, see Version numbers in Business Central.

Last known error

The link behind the sentence *View the last known error* will find and show the most recent error message that was was generated by the application code. This includes errors from field validation, posting routines, and other code behind business functionality.

The information that you can get from this link includes the following:

- Text

  This is the error message that the user sees, either in a dialog window or next to a user interface element that cannot render, for example.

- Code

  This is the code snippet that threw the error.

- Callstack

  This shows how the error was triggered.

- Object

  This shows information about the runtime objects.

The link cannot open errors that were generated by the platform. So if you suspect that the issue is caused by the platform, you can try to reproduce the error in a sandbox environment before you contact Microsoft for support. For more information, see Create a sandbox environment.

> **TIP**
> If your users complain of a confusing error message such as *Sorry, we just updated this page. Please close and reopen.*, then you can often find the underlying problem either in this last known error, or by analyzing telemetry in the Business Central administration center. For example, in the case of the *Sorry, we just updated this page. Please close and reopen.* message, the underlying problem is often that two users are trying to modify the same data. So if both users open the same sales order, and both change a field, then one of them will see the *Sorry, we just updated this page. Please close and reopen.* message, because Business Central saves changes as soon as you move to the next field or close the page.

Escalating support issues to Microsoft

Sometimes the tenant has run into a problem that the partner cannot resolve. In those cases, the delegated admin can use the Partner Center or the Business Central administration center to submit a support request to Microsoft.

Both internal and delegated administrators can access Business Central administration center, and then, in the **Support** menu, choose the **New Support Request** button. This logs you in to the Power Platform Admin Center. Here, you can launch the **New Support Request** guide that will help you identify potential solutions or workarounds based on how you fill in the various fields.

In the Power Platform Admin Center, both internal and delegated administrators can explore different solutions based on the keywords that they specify.

> **NOTE**
> The internal administrator cannot contact Microsoft directly. If you are an internal admin and suspect that something is wrong with your Business Central, you must contact your partner for next steps.

Submitting support requests on behalf of your customer

As the delegated administrator, if you are logged into the Business Central administration center, you can use the **New Support Request** link in the **Support** menu to submit a support request on behalf of your customer.

> **TIP**
>
> Alternatively, you can use customer-specific URLs such as `https://admin.powerplatform.microsoft.com/account/login/[customer tenant ID]`.

To start the process of submitting a new support request from the Business Central administration center

1. On the **Environments** tab of the Business Central administration center, choose the relevant environment to open the environment details.
2. In the **Support** menu, choose **New Support Request**.

   This opens a new browser tab so that you can submit the support request in the Power Platform Admin Center.

In the Power Platform Admin Center, you are automatically logged in with information about the customer tenant that you are working on behalf of. Create a new support request and fill in the fields as appropriate, but remember to use the **See solutions** button to find potential guidance or workarounds. Based on your search keywords, links to suggested documentation are shown on the **Solutions** tab.

You can find most of the necessary information in the Business Central administration center, including the tenant ID and the Business Central version numbers. For more information, see View solutions or enter a support request through the new support center in the Power Platform administration content.

> **IMPORTANT**
>
> Your company must be registered as a partner in order to submit a support request to Microsoft, and you must have the ASfP (Advanced Support for Partners) support plan. The support person can be a member of the **Helpdesk agent** group in the customer's Azure AD tenant or a global administrator. For more information, see Delegated Administrator Access to Business Central Online. Your service account manager can get you more information about getting the ASfP, and, if you already have a support plan, they can get the contract information and access ID that you must specify when you submit a new support request on behalf of your customer.

Microsoft Support will keep you updated on the status of your support request. You can also see the status in the Power Platform Admin Center. For more information, see Power Platform Admin Center.

To start the process of submitting a new support request from the Partner Center

You can also start the process from the Partner Center, where you can choose the customer you want to open a case on, and then follow the support request work flow. This will redirect the delegated administrator to the Power Platform admin center in the context of the customer's tenant.

However, you might need information from the Business Central administration center, which is why we recommend that route. For more information, see Report problems on behalf of a customer in the Partner Center content.

Report customer outages

When a customer has a situation where no users can log in to Business Central, you must take immediate action. Outages are frustrating but rare, so make sure that you verify that the users are not unable to log in due to problems with their network connection, for example. For more information, see How do I check my online service health? in the Power Platform administration content.

Internal and delegated administrators can report this outage to Microsoft by using the **Report Production Outage** action for the relevant production environment in the Business Central administration center. This action creates a support ticket for Microsoft with all the information that is needed to begin steps to resolve the issue.

> **NOTE**
> This option is not available in sandbox environments.

To report an outage

1. On the **Environments** tab of the Business Central administration center, choose the relevant environment to open the environment details.
2. In the action ribbon, choose **Support**, and then choose **Report Production Outage**.
3. In the **Report Production Outage** pane, choose the outage type:
   - Unable to log on (all users)
   - Cannot access API/Web Service
4. Enter your name, email address, and phone number. This information will be included in the support ticket.
5. Choose **Next**.
6. In the next pane, provide details about the outage, including which browsers users have tried to log in with, any companies that you can log into, and errors and correlation IDs. This information will be included in the support ticket.
7. Finally, add the date and time when the outage began. This information will also flow to the support ticket.
8. Choose the consent checkbox, and then choose **Report**.

A support request ticket is then created, and you will see a dialog box with the ticket ID. You can then monitor progress in the **Reported Outages** section. From there, you can access the tickets in the Partner Center. For more information, see the Microsoft Partner Center documentation.

See Also

Inspecting and Troubleshooting Pages
The Business Central Administration Center
Technical Support for Business Central
Provide technical support (Microsoft Partner Center)
Deployment Overview
Administration as a partner
Administration of Business Central Online
Administration of Business Central On-Premises

Special Permission Sets

2/17/2021 • 2 minutes to read • Edit Online

The following permission sets have special definitions that you should be aware of as you implement permissions and security for Business Central users.

PERMISSION SET	DEFINITION
SUPER	Can read, use, update, and delete all data and all application objects in the scope of your license. Business Central requires that at least one user is assigned this permission set in each database. The first user created is automatically assigned the SUPER Permission Set You can't modify permissions for the SUPER permission set.
SUPER (DATA)	Can read, use, update, and delete all data. You typically assign this permission set to an accounting manager who needs to work with all data but doesn't need to change Business Central. **Note:** This permission set is available only for Business Central on-premises.
SECURITY	Manage the permission sets that are assigned to your account. When assigned this permission set, you can: • Create new users and assign them any permission set that is also assigned to your account. • Remove a permission set from a user as long as the permission set is also assigned to your account. • Modify individual permission granted by a permission set as long as the permission set is also assigned to your account. The idea behind this permission set is to prohibit you from granting users more permissions than you have. The permission set is useful for SUPER users or global administrators who want to delegate permission management to team administrators. For example, a sales manager can assign permissions in sales area to sales people, sales assistant, sales coordinator, and so on.
BASIC	Grants Read access to almost all application tables and all system tables. The main purpose of this permission set is to enable the service to open and show all pages.

PERMISSION SET	DEFINITION
FOUNDATION	A prerequisite for all other permission sets. The FOUNDATION permission set grants access to system tables and application setup tables that are required for most application features to work. **Note:** This permission set is recommended when using the UI Elements Removal feature to automatically remove UI elements according to user permissions. For more information, see Removing Elements from the User Interface According to Permissions. **Note:** This permission set is available only for Business Central on-premises.

See Also

Removing Elements from the User Interface According to Permissions

Setting up the Excel Add-In for Editing Business Central Data

2/17/2021 • 9 minutes to read • Edit Online

You can set up the Business Central deployment to support an Excel add-in that enables users in the Business Central client to work with data from list pages in Excel. Users can get fresh data from Business Central into Excel and push updated data from Excel to Business Central.

Without this add-in, users can open a list page in Excel from the **Open in Excel** action on the page. But the **Open in Excel** action doesn't allow them to push changed data back to Business Central. When this add-in is set up, the **Open in Excel** action is replaced by the **Edit in Excel** action. The add-in is based on the Microsoft Dynamics Office Add-In.

> **NOTE**
> The Excel add-in is not available in the mobile apps.

This article provides guidance for how to configure Business Central so that users can edit data in Excel.

> **NOTE**
> This article does not apply to the older Excel add-in that was available for installation with the Dynamics NAV Client connected to Business Central client by using the Business Central Setup in versions older than 2019 release wave 2.

Deploy the Excel add-in for Business Central online

For Business Central online, the administrator can deploy the add-in for all users. But users can also install the add-in themselves, provided they have permission to configure their Office experience.

> **TIP**
> In some organizations, administrators cannot deploy add-ins centrally. For more information, see Determine if Centralized Deployment of add-ins works for your organization.

To deploy the Excel add-in for all users

1. As the administrator, sign in to the Microsoft commercial website and find the add-in at https://appsource.microsoft.com/product/office/WA104379629.
2. Choose the **Get it now** button.

   You'll be redirected to the Microsoft 365 admin center.
3. In the left panel, go to **Settings**, and then choose **Add-ins**.
4. In the **Configure add-in** pane, specify which users to grant access to the add-in.
5. Save your changes.

When users now choose the **Edit in Excel** action, the add-in will launch as a pane in Excel. Each user will be automatically logged in and connected to their Business Central, but if a user cannot connect automatically, you

can unblock them by asking them to follow the steps in the To configure the connection section.

To add the Excel add-in locally

1. Open Excel, and then open any Excel workbook.
2. On the **Insert** menu, choose **Office Add-ins**, and then choose **Admin managed** or **Store** as appropriate.
3. Search for *Dynamics Office Add-In*, and then install the add-in.

When the add-in is installed, it shows up as a panel in Excel. Next, you must configure the connection.

To configure the connection

1. In the Dynamics 365 Excel add-in, choose **Add server information**, and then in the **Server URL** field, enter `https://exceladdinprovider.smb.dynamics.com`.
2. Choose the OK button, and then confirm that the app reloads.
3. When prompted, sign in with your Azure Active Directory account.
4. Optionally, choose the environment and company that you want to connect to.

The add-in is now connected to your Business Central, and you can edit data and publish the changes to Business Central.

> **TIP**
> If the workbook is not automatically saved to the user's OneDrive, then recommend them to save all workbooks that they export from Business Central. When they open the workbook again, the connection is still available, so they do not have to configure the connection again.

> **NOTE**
> In certain deployments, the administrator must configure network access to unblock the Excel add-in. For more information, see Preparing Your Network for the Excel Add-In.

Troubleshooting

Sometimes, users run into problems with the Excel add-in. In this section, we provide tips for how to unblock users in certain circumstances.

ISSUE	SOLUTION OR WORKAROUND	COMMENTS
The add-in doesn't start	Check if the add-in is deployed centrally, or if the user is blocked from installing it locally.	The admin can configure Office so that users cannot acquire add-ins. In those cases, the admin must deploy the add-in centrally. For more information, see Deploy add-ins in the admin center.
Data does not load into Excel	Test the connection by opening another list in Excel from Business Central. Alternatively, open the workbook in Excel in a browser.	If the user has specified a company name that contains special characters, the add-in might not be able to connect.
Data can't publish back to Business Central.	Test the connection by opening the workbook in Excel in a browser.	Sometimes an extension can block the publishing job. If the page is extended or customized, remove the extensions, and then try again.

ISSUE	SOLUTION OR WORKAROUND	COMMENTS
The dates are wrong	Excel might show times and dates in a different format than Business Central. This does not make them wrong, and the data in Business Central will not get messed up.	

Deploy the Excel add-in for Business Central on-premises

Your on-premises deployment must meet the following prerequisites:

- Azure Active Directory (Azure AD) used to authenticate users.

  The Business Central Server instance, clients, and users must be configured for Azure Active Directory (Azure AD) authentication.

  For more information, see Authenticating Users with Azure Active Directory.

- OData enabled and uses Secure Sockets Layer (SSL) for authentication.

  For more information, see Using Security Certificates with Business Central On-Premises.

- Business Central Web Server installed and configured to use SSL (https).

  For more information, see Install Business Central and Configure SSL to Secure the Connection to Web Client.

- If your deployment is multitenant, Business Central Web client must accept host names for tenants.

  For more information, see Configure the Web Server to Accept Host Names for Tenants.

- Business Central client computers have a supported version of Excel.

  For more information, see System Requirements.

Expose the Business Central application Web API in Azure AD

APPLIES TO: Business Central On-Premises

When Business Central is configured for Azure AD authentication, the Business Central application is registered as an application in an Azure AD. Before the Excel add-in can be configured, you must configure the Business Central application in Azure AD to expose its Web API.

For information about how to expose the Web API, see Quickstart: Configure an application to expose web APIs.

Register and configure an Azure AD application for the Excel Add-in in Microsoft Azure

APPLIES TO: Business Central On-Premises

When Azure AD authentication was set up for your Business Central deployment, an Azure AD tenant was created in Microsoft Azure, and an application for Business Central was registered in the tenant. The Excel add-in requires that you add (register) a separate Azure AD application in the Azure AD tenant.

1. Register an Azure AD application for the Excel add-in.

   You add the Azure AD application by using the Azure portal. For guidelines, see Register your application with your Azure Active Directory tenant.

   When you add an application to an Azure AD tenant, you must specify the following information:

SETTING	DESCRIPTION
Name	The name of your application as it will display to your users, such as *Excel Add-in for Business Central*.
Supported account types	Specifies which accounts that you would like your application to support. For purposes of this article, select **Accounts in this organizational directory only**.
Redirect URI	Specifies the type of application that you're registering and the redirect URI (or reply URL) for your application. Select the type to **Web**, and in the redirect URL box, enter URL for signing in to the Business Central Web client, for example `https://localhost:443/BC170/SignIn`. The URI has the format `https://<domain or computer name>/<webserver-instance>/SignIn`, such as `https://cronusinternationltd.onmicrosoft.com/BC170/SignIn` or `https://MyBcWebServer/BC170/Signin`. **Important** The portion of the reply URL after the domain name (in this case `BC170/SignIn`) is case-sensitive, so make sure that the web server instance name matches the case of the web server instance name as it is defined on IIS for your Business Central Web Server installation.

When completed, the **Overview** displays in the portal for the new Excel Add-in application.

2. Grant the Excel add-in application permission to access the Business Central application Web API.

   Give the Azure AD application for the Excel add-in delegated permission to access the Business Central application Web API in Azure AD (which you exposed earlier in this article). This permission allows users of the Excel add-in to access the OData web services to read and write data.

   a. From the application's **Overview**, select **API Permissions**.
   b. Select the **Add a permission**
   c. On the **APIs my organization uses**, select the Business Central application.
   d. Select **Delegated permission**.
   e. Select the permission from the list, and then select **Add Permission**.

   For information, see Quickstart: Configure a client application to access web APIs.

3. Configure OAuth2 authentication in the Excel add-in.

   The Excel add-in requires OAuth2 implicit grant flow to be enabled on the Excel Add-in application. You configure OAuth2 in the manifest file for the Excel Add-in application. From the application's **Overview**, select **Manifest**, and then set `"oauth2AllowIdTokenImplicitFlow"` and `"oauth2AllowImplicitFlow"` to `true`:

   ```
   "oauth2AllowIdTokenImplicitFlow": true,
   "oauth2AllowImplicitFlow": true,
   ```

4. In the manifest, add the following URL entry to the `"replyUrlsWithType":`:

```
{
    "url": "https://az689774.vo.msecnd.net/dynamicsofficeapp/v1.3.0.0/*",
    "type": "Web"
}
```

Remember to add a comma before or after this entry, depending on where you add it in the list.

5. Copy the **Application (Client) ID** that is assigned to Excel add-in application.

   You can get this value from the **Overview** page for the application. You'll need it to configure the Business Central Server instance.

This completes the work you have to do in the Azure portal. The final configuration is to add the Excel add-in to the Business Central Server instances.

Configure the Business Central Server Instances

APPLIES TO: Business Central On-Premises

You add the Excel add-in to the Business Central Server instances in your deployment. You can use either the Business Central Server Administration tool or Set-NAVServerConfiguration cmdlet in the Business Central Administration Shell.

1. In the Business Central Server Administration tool, in the **Azure Active Directory** section, set the **Excel add-in AAD client ID** field to the application (client) ID for the Excel add-in application that you copied from the Azure portal.

   If you use the Set-NAVServerConfiguration cmdlet, set the `ExcelAddInAzureActiveDirectoryClientId` key.

2. In the **Client Services** section, set the **Web Client Base URL** field to the base URL of the Business Central Web client.

   This value is the root portion of all URLs that are used to access pages in the web client. The value must have the format `https://[hostname:port]/[instance]`, such as `https://MyBCWebServer/BC` or `https://cronusinternationltd.onmicrosoft.com/BC170`.

   If using the **Set-NAVServerConfiguration** cmdlet, set the `PublicWebBaseUrl` key.

3. In the **OData Services** section, set the **OData Base URL** field to the public URL for accessing OData services.

   The URL must have the following format `https://<hostname>:<port>/<instance>/ODataV4/`, such as `https://localhost:7048/BC170/ODataV4/`.

   With the **Set-NAVServerConfiguration** cmdlet, set the `PublicODataBaseUrl` key.

Prepare devices and network for the Excel Add-In

Network services such as proxies or firewalls must allow routing between each client device on which the Add-In is installed and a number of service endpoints. For a list of endpoints, see Preparing your network for the Excel Add-In.

Use the Excel Add-In

Your users can now use the Excel add-in. When a list page shows the **Edit in Excel** action, then users can open lists, such as the **Customers** page, in Excel and work with the data there. They use the add-in to update data in Business Central and get fresh data from the database. For more information, see Viewing and Editing in Excel From Business Central.

See Also

Configuring Business Central Server
Authenticating Users with Azure Active Directory

Preparing Your Network for the Excel Add-In

2/17/2021 • 2 minutes to read • Edit Online

As the administrator of Business Central online or on-premises, you can deploy an add-in so that users can work with their Business Central data in the Excel desktop app. This article includes information that helps administrators configure advanced device and network settings for the Excel add-in for Business Central when your network requirements block the add-in.

Network endpoints required by the Excel add-in

The Excel add-in must have access to Business Central for it to function and synchronize business data. Network services, such as proxies or firewalls, must allow routing between each client device on which the add-in is installed and the following services that are published by Microsoft:

- `https://az689774.vo.msecnd.net`
- `https://dc.services.visualstudio.com`
- `https://exceladdinprovider.smb.dynamics.com`
- `https://api.businesscentral.dynamics.com`

> **NOTE**
>
> Your API endpoint URL can be different from these default values. For example, if your environment is based on a vertical solution by Fabrikam, your API endpoint is `https://fabrikam.api.bc.dynamics.com`. These vertical solutions have API endpoints that are based on the format of `ApplicationName`*.api.bc.dynamics.com*. You can check if an environment uses such a URL in the Business Central Administration Center.

See also

Setting up the Excel Add-In for Editing Business Central Data
Administration of Business Central Online
Managing an Business Central Embed App in Microsoft Lifecycle Services

Setting up App Key Vaults for Business Central Online

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

AppSource apps for Business Central can be developed to get secrets from Azure Keys Vaults. The app key vault feature is readily available for use on the service by all App Source apps. However, there are some onboarding tasks required.

> **IMPORTANT**
> With Business Central online, App key vaults can only be used with AppSource apps. They're not supported with per-tenant extensions.

> **TIP**
> You must also specify secrets in a key vault if you deploy Business Central as part of the Embed App program. Especially if you must support the Outlook add-in, in which case you must specify secrets for TEMPORARYDOCUMENTSTORAGEACCOUNT and TEMPORARYDOCUMENTSTORAGEKEY.

For more information about developing extensions with key vaults, see Using Key Vault Secrets in Business Central Extensions.

Create the Azure Key Vault with secrets

In this task, you create a key vault in Azure, and add the secrets that you want to make available to your extensions. An extension can use up to two key vaults, so you can create more than one.

There are different ways to create an Azure key vault. For example, you can use the Azure portal, Azure CLI, and more.

The easiest way is to use the Azure portal. For instructions, see Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal.

For using other methods, see Azure Key Vault Developer's Guide.

Provision the key reader application in your Azure AD tenant

Your Business Central online solution is configured to use an Azure AD application for reading key vault secrets. The application is called **Dynamics 365 Business Central ISV Key Vault Reader**. Microsoft manages the key vault reader application, however, there are a couple tasks that you have to do to enable it. First, the application must be provisioned on your Azure AD tenant, as described here.

To provision the key vault reader application, use the Azure Active Directory PowerShell module.

1. Open Windows PowerShell as an administrator.
2. Install the Azure Active Directory PowerShell module.

```
Install-Module AzureAD
```

3. Import the Azure AD module.

```
Import-Module AzureAD
```

4. Connect to your Business Central Azure AD tenant.

   a. Run the following command:

   ```
   Connect-AzureAD
   ```

   b. Provide your sign-in name and password when prompted.

5. Create an Azure AD service principal using the following command:

```
New-AzureADServicePrincipal -AppId 7e97dcfb-bcdd-426e-8f0a-96439602627a
```

`7e97dcfb-bcdd-426e-8f0a-96439602627a` is the Application (client) ID of Microsoft's centralized Azure AD application.

This step provisions the application in your Azure AD tenant, where it now "lives" together with your key vaults.

Grant the key vault reader application permission to your key vaults

The next task is to grant the key vault reader application permission to read secrets from your key vaults. The steps in this task are done from the Azure portal.

1. Open the key vault in the portal.
2. Select **Access policies**, then **Add Access Policy**.
3. Set **Secret Permissions** to **Get**.
4. Choose **Select principal**, and on the right, search for either the application (client) ID **7e97dcfb-bcdd-426e-8f0a-96439602627a** or the display name **Dynamics 365 Business Central ISV Key Vault Reader**.
5. Select **Add**, then **Save**.

Contact Microsoft to enable the App Key Vault feature

Send an email to bcappkeyvaultonboard@microsoft.com to start the onboarding process. Do this step before you publish your updated extension to Partner Center.

The onboarding process involves a manual verification step that verifies that you own the AAD tenant that contains the key vaults.

Provide the following information in the email:

- Your AAD tenant ID. Obtain this information from the Azure portal by going to the Azure Active Directory Overview page.
- Your AppSource extensions, including names and App IDs, that should be enabled to read secrets from your key vaults.
- Optionally, a screenshot from the Azure portal showing the key vault and its access policies. The screenshot can help Microsoft catch configuration mistakes early in the process.

See Also

Security Considerations With App Key Vaults
Monitoring and Troubleshooting App Key Vaults
Configuring Business Central Server

Upgrading AppSource Apps in Production

2/17/2021 • 2 minutes to read • Edit Online

When an updated version of an AppSource app becomes the active version in the Dynamics 365 Business Central service, tenants do not automatically get this updated version. This upgrade must be done manually by getting the latest version of the app in AppSource.

To upgrade an AppSource app

Follow the steps below to update a tenant to the latest version of any AppSource app.

1. Log in to the Dynamics 365 Business Central browser client.
2. In the **Tell Me** box, enter **Extension Management**, and then choose the related link.
3. Locate the app that you want to update.
4. Under **Manage**, choose **Uninstall**.
5. Go to AppSource and then install the app again.

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension

Production and Sandbox Environments

2/17/2021 • 6 minutes to read • Edit Online

You can create environments of different types. Which type of environment to choose depends on what you need it for.

Production environments

Production environments are meant to be precisely that: Environments that a business can run their daily business in Business Central in, deployed on performance tiers in Azure with a guaranteed high level of availability and support.

Production environments are backed up automatically and frequently to help protect business data. For more information, see How often are production databases backed up?.

You can create additional production environments for training or performance testing, for example. However, for training purposes, in many cases organizations will prefer to create a sandbox environment with production data. You can also create additional production environments to support offices in different countries.

> **NOTE**
>
> The Premium and Essential subscription types give each Business Central customer one production environment and three sandbox environment free of extra charge. If the customer requires more production environments, they can buy additional environments through their CSP partner. Each additional production environment comes with three additional sandbox environments and 4 GB additional, tenant-wide database capacity.

Administrators can create the additional environments in the Business Central administration center. The environments can be created and used in any country or region where Business Central is available, including the countries or regions where the customer's existing environments are running. The environments quota is applied when you try to create a new environment, or copy an existing environment, in the Business Central administration center.

Sandbox environments

Sandbox environments are meant to be precisely that: Environments that you can play around with, use as a testbed for development, and delete at will. You can deploy apps straight from Visual Studio Code to a sandbox environment, and you can attach a debugging session to a sandbox.

> **IMPORTANT**
>
> Apps that are published to a sandbox from the development environment or created using Designer are published within the scope of the service node that hosts the environment. When the sandbox is upgraded, these apps are removed because the environment is moved to another node that is running the new version. However, the data of the app is not removed, so you only have to re-publish and install the app to make it available.
> Apps that are uploaded to the environments of both types (production and sandbox) using the **Upload Extension** action from the **Extension Management** page are published within a global scope. When the environment is upgraded or moved, these apps are downloaded to the service node and installed, which means that they will not disappear.

You can also safely use sandboxes for training, such as for following a learning path from Microsoft Learn, because it's a safe environment to experiment with. If anything goes wrong, you just delete the sandbox and

start over.

> **IMPORTANT**
> The automatic backup that applies to production environments does not apply to sandbox environments. If you want to export data from a sandbox environment, you can use Excel or RapidStart, but you cannot request a database export.

You can create a sandbox environment that includes data from your production environment for debugging purposes, for example. But if you want to run performance tests, or similar benchmarking, the sandbox is not reliable enough for that purpose. This is because sandboxes run in a different performance tier on Azure than production environments. Instead, create a dedicated environment based on the Production environment type - this gives you the exact experience and performance that users will experience in the actual production environment.

Sandbox environments are handy for certain types of development scenarios because the debugging endpoint is open by default. This means that you can attach Visual Studio Code to a running system and debug through running code. It also allows you to publish directly to the environment from Code.

If your organization has more than one sandbox environment, you can switch between environments by opening the App Launcher, choosing the Dynamics 365 tile, then choose the Business Central Sandbox tile. The sandbox environment picker shows the available sandboxes, so choose the one that you want to switch to.

> **NOTE**
> The Premium and Essential subscription types give each Business Central customer one production environment and three sandbox environment free of extra charge. If the customer requires more production environments, they can buy additional environments through their CSP partner. Each additional production environment comes with three additional sandbox environments and 4 GB additional, tenant-wide database capacity.

Administrators can create the additional environments in the Business Central administration center. The environments can be created and used in any country or region where Business Central is available, including the countries or regions where the customer's existing environments are running. The environments quota is applied when you try to create a new environment, or copy an existing environment, in the Business Central administration center.

Precautions for sandbox environments with production data

If a sandbox is created with a copy of a production environment, a number of precautions are taken for that sandbox:

- The job queue is automatically stopped
- Any base application integration settings are cleared
- Outbound HTTP calls from extensions are blocked by default and must be approved for each extension

  To enable outbound HTTP calls, go to the **Extension Management** page in Business Central, and choose **Configure**. Then, on the **Extension Settings** page, make sure that **Allow HttpClient Requests** is selected. This setting must be enabled for each extension.

- Any General Data Protection Regulation (GDPR) action must be handled separately and repeated for the sandbox. There is no synchronization with the production environment after the sandbox has been created.

  The internal administrator has the same tools and responsibilities for a sandbox environment as they do for a production environment. As a data processor, Business Central offers the same level of data protection and data handling restrictions that we apply to production environments.

Pre-sales performance evaluation

If you want to provide a prospect with an online environment where you want to demonstrate the performance and reliability of Business Central online in addition to demonstrating functionality, you must take a few extra steps.

To demonstrate the functionality of the default version of Business Central, without focusing on performance, you can quite simply use your own trial experience based on a Microsoft 365 demo account. We recommend that you show the full functionality of the default version by switching the tenant to the 30 day trial and the associated My Company. You can then enable the *Premium* user experience in the new My Company's **Company Information** page, populate the new company with the data required for their evaluation scenarios, and present the environment to the prospect.

To demonstrate the functionality of the service, with a focus on performance, you can take the same step as outlined above, but then also sign up the prospect for the Business Central Premium Trial offer that is available through your CSP access in the Partner Center, wait 24 hours, and then run your performance evaluation. For more information, see Preparing Test Environments of Dynamics 365 Business Central.

That 30 day trial is as close to an actual production environment performance as you can get. You only have those 29 days to run your tests and convince the prospect, of course, but provided that you have prepared everything in advance, it should give you time enough.

If the prospect is convinced and decides to buy Business Central, you can then either let them keep the environment that they are currently using, or create a new production environment for them. If the tenant is yours rather than the prospect's, then a new tenant will be provided to them.

For more information about performance and Business Central, see Performance Overview.

Creating default sandbox environments

A single, default sandbox environment can also be created in the Business Central application. For more information, see How to: Create a Sandbox Environment.

See also

Managing Environments in the Administration Center
Preparing Demonstration Environments of Dynamics 365 Business Central
Preparing Test Environments of Dynamics 365 Business Central
Steps to set up a sandbox environment and Visual Studio Code
Get started with the Container Sandbox Development Environment

Preparing Demonstration Environments of Dynamics 365 Business Central

2/17/2021 • 8 minutes to read • Edit Online

As a Business Central reselling partner, you might want to have an environment that you can show prospects as part of pre-sales demonstrations. Depending on your requirements, you have several different options that are described in this article.

Microsoft 365 demo plus Business Central

For repeatable demos that showcase Business Central together with Microsoft apps and services in an independent Azure organization (tenant) that you fully control, get a Microsoft 365 demo account that you create on https://aka.ms/CDX, which has replaceddemos.microsoft.com. You have access to the site if you are enrolled with the Microsoft Partner Center.

Such Microsoft demo accounts give you environments that you can use for demos and training without the risk of introducing changes to an existing production environment, for example. For more information about demo accounts in general, see Offer your customers trials of Microsoft products in the Partner Center documentation.

To get a demo environment based on Microsoft 365 content packs

1. Log in to aka.ms/cdx using your partner account.
2. Choose the **My Environments** tab, and then, under **My tenants**, choose the **Create Tenant** button.
3. As the type, choose **Quick Tenant**.
4. As the period, choose either a quarter or a full year.

   Your account in CDX has access to a limited quota of tenants that can last three months or a full year. For more information, see the CDX FAQ.
5. As the location, choose the region that is closest to your location.

   This determines the location in which the tenant is deployed, but you will be able to add Business Central environments to it for any country/region afterwards.
6. Choose one of the available content packs and then choose the **Create Tenant** button.

   You can choose the **Dynamics 365 Business Central** content pack if the location you specified is *North America*, *Asia/Pacific*, or *Europe, Middle East, Africa*. This content pack includes demonstration users and Microsoft 365 licenses.

   You can also choose any of the other content packs that might better match your prospects' industry or interests, such as the **Microsoft 365 Business Premium Demo Content** or **Microsoft Teams for Consumer Goods** content packs. These content packs give you a dedicated demonstration organization in Azure that includes multiple demo users with Microsoft 365 licenses.

   > **NOTE**
   > When demand for a specific content pack is high, the **Create Tenant** button is disabled, and you must choose a different content pack.

Next, you sign up for a Business Central trial.

7. In an dedicated browser window, go to the https://dynamics.microsoft.com/business-central/ page, choose the **Get started** button, and then choose the **Sign up now** button.

   Use the new administrator account that you got as part of your demo account, typically called something like `admin@CRMbc123456.onmicrosoft.com`, or `admin@contoso.onmicrosoft.com`.

   > **TIP**
   > We recommend that you use profiles in the Microsoft Edge browser rather than InPrivate or Incognito browser mode. For more information, see Microsoft Edge documentation.

By now, you have a dedicated demonstration organization in Azure that includes multiple demo users with Microsoft 365 licenses, and a Business Central environment.

The initial Business Central demo environment comes with two companies:

- The CRONUS demonstration company
- An empty company with the name*My Company*, but it might have a different display name for demo purposes

You can keep using the demonstration company until the tenant expires, provided that you use it at least a couple of times per week. However, if you start the trial experience or switch to the empty *My Company*, that experience will expire after 30 days. At that point, you canextend the trial, create a new environment, or you can return to the CRONUS demonstration company.

You can now enable the *Premium* user experience in the new My Company's **Company Information** page, populate the new company with the data required for their evaluation scenarios, and present the environment to the prospect.

> **IMPORTANT**
> Environments that are based on Microsoft 365 demo accounts are intended for demonstration and training purposes. If a prospect uses such an environment to help run their business, then they risk losing their company data when demo environment or the Microsoft 365 demo account expires. If a prospect wants to use Business Central to help run their business, they should sign up for a trial using their own email account. For more information, see Dynamics 365 Business Central Trials and Subscriptions in the business functionality content for Business Central.

Customize the demo environment

Because you are logged in as the global administrator of the demonstration tenant, you can access the Microsoft 365 administration center. The following list illustrates the types of work that your demo account can do:

- Subscribe to additional apps and services that may be relevant to your demos
- Configure apps and services by accessing their administration centers
- Modify or add user accounts, and assign licenses
- Set up the organizational profile to reflect your prospect's business and brand

Your administrator account also gives you access to the Business Central administration center, where you can make further customizations, such as the following:

- Create additional environments within the available quota, depending on your needs.

  For example, you can create a Business Central environment for a specific country/region, or create a

dedicated sandbox that you then apply your solution to, such as by deploying your app straight from Visual Studio Code. For more information, seeManaging Environments.

- Specify the update window and date for your environments, so that planned updates will not interfere with your demos.
- Add your own email address to the list of notification recipients, so that you will be informed of upgrades to your Business Central environments or any issues that may impact your planned demos.
- Try out new functionality in preview environments that are available about two months before a major update. For more information, seePrepare for major updates with preview environments.

With your administrator account, you can install apps from AppSource, and you can set up integrations between Business Central and other services. For example, you can add the Business Central app for Microsoft Teams, and you can run the**Set up your Business Inbox in Outlook**assisted setup guide on behalf of the organization. For more information, seeUsing Business Central as your Business Inbox in Outlook. You can also add Dynamics 365 Customer Service,Dynamics 365 Sales,Power Automate,Power Apps, and many more.

Some content packs provide additional demonstration data for non-administrator accounts, such as example channels and conversations in Microsoft Teams that can reduce the time needed to customize the experience for a demo. You can log in with the demo user account, often called `meganb@M365B123456.onmicrosoft.com`, or something similar, for the non-administrator experience.

Use profiles in Microsoft Edge

If you demo in the new Microsoft Edge browser, you can easily switch between different browser profiles. That way, you do not have to use private mode for browsing, and you can let Microsoft Edge save passwords and sites to any of your browser profiles, including a Microsoft 365 demo account. Business Central also performs well in the Microsoft Edge for even better demos. You can also switch the page layout to *Focused* to minimize demo distraction. For more information, see Microsoft Edge documentation.

> **TIP**
> We recommend that you connect to a low-latency network for a faster response time during demos, and that you always plug in your laptop if you are about to demo anything. Performance impact may vary depending on your device and choice of browser, but being plugged in generally helps overall snappiness.

Pre-sales performance evaluation

If you want to provide a prospect with an online environment where you want to demonstrate the performance and reliability of Business Central online in addition to demonstrating functionality, you must take a few extra steps.

To demonstrate the functionality of the default version of Business Central, without focusing on performance, you can quite simply use your own trial experience based on a Microsoft 365 demo account. We recommend that you show the full functionality of the default version by switching the tenant to the 30 day trial and the associated My Company. You can then enable the *Premium* user experience in the new My Company's **Company Information** page, populate the new company with the data required for their evaluation scenarios, and present the environment to the prospect.

To demonstrate the functionality of the service, with a focus on performance, you can take the same step as outlined above, but then also sign up the prospect for the Business Central Premium Trial offer that is available through your CSP access in the Partner Center, wait 24 hours, and then run your performance evaluation. For more information, see Preparing Test Environments of Dynamics 365 Business Central.

That 30 day trial is as close to an actual production environment performance as you can get. You only have those 29 days to run your tests and convince the prospect, of course, but provided that you have prepared everything in advance, it should give you time enough.

If the prospect is convinced and decides to buy Business Central, you can then either let them keep the environment that they are currently using, or create a new production environment for them. If the tenant is yours rather than the prospect's, then a new tenant will be provided to them.

For more information about performance and Business Central, see Performance Overview.

Adding a Business Central trial to your prospect's organization

The easiest way to give a prospect a demonstration of Business Central online, is to quite simply ask them to sign up for a trial from the https://dynamics.microsoft.com/business-central/ page. This way, the prospect will be able to explore the default version of Business Central on their own or together with you.

This type of environment can be useful if you want to demonstrate the general user interface, for example, or talk about how they can add capabilities with apps from AppSource.

> **NOTE**
> If you want to show the prospect the full capabilities in Business Central, you must ask them to change the user experience to Premium in the **Company Information** page. For more information, see Change Which Features are Displayed.

If a prospect has signed up for a free trial, and they use Business Central to help run their business, then they must decide whether to subscribe within the first 60 days. If they extend their trial once, and they are still not sure, they must contact a partner.

However, in many cases, you will probably prefer to show prospects more tailored experiences with your own trial as described in the Microsoft 365 demo plus Business Central section.

See also

Preparing Test Environments of Dynamics 365 Business Central
The Business Central Administration Center
Managing Environments
Administration of Business Central Online
Deployment of Dynamics 365 Business Central
Offer your customers trials of Microsoft products

Preparing Test Environments of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

As a Business Central reselling partner, you might want to have an environment that you can use for testing or pre-sales demonstrations. You can create such environments based on free 30 day trials, or you can set up a dedicated environment if you have a Business Central subscription yourself.

Test environments based on a 30 day trial

This type of environment can be useful if you want to run benchmark tests, for example. The following procedure requires CSP access in Partner Center. If you do not have CSP access, you can work with your CSP distributor to do this for you. For more information, see the Partner Center documentation.

To create a test environment based on a 30 day trial

1. In Partner Center, create a new test customer with any test domain, such as *contoso.onmicrosoft.com*. For more information, see Add a new customer in the Partner Center docs.

   a. Fill in the fields as described in the following list:

FIELD	VALUE
Country	<Your country>
Subscription	*Dynamics 365 Business Central Premium Trial* This will give you 25 Premium licenses for 30 days for free

   b. Make a note of the Admin credentials that are shown on the last page of the customer creation wizard. You will need these credentials later.

2. In Partner Center, once the test customer has been created, create a new test user. For more information, see Create user accounts for a customer in the Partner Center documentation.

3. Assign that user 1 Premium license.

   > **NOTE**
   > It may take up to 10 minutes for the available licenses to show up on the **Users** page.

   Also, make a note of the user credentials shown on the last page of the user creation wizard. You will use this information in step 55

4. In the **Service Management** section, choose the **Dynamics 365 Business Central** link.

   This opens the Business Central administration center at the equivalent of the following URL:
   `https://businesscentral.dynamics.com/contoso.onmicrosoft.com/admin`

   In the Business Central administration center, you can create new production and sandbox environments for the test customer.

> **TIP**
> Always include the domain or the Azure Active Directory ID of the customer in the URL when you login as a *delegated admin*. This way, you always know exactly which customer you are trying to access.

5. Access Business Central as the local user that you created in step 2.

   a. Open another browser window in InPrivate or Incognito mode. This way, you can make sure that you are not logging in with your own credentials.

   > **TIP**
   > We recommend that you use profiles in the Microsoft Edge browser instead. For more information, see Microsoft Edge documentation.

   b. Go to *https://businesscentral.dynamics.com/*, and then, when you are asked to sign in, use the credentials of the user you created in step 2.

The Business Central environment is created automatically when you use the environment URL to login the first time.

Prepare for major updates with preview environments

About two months before a major update, you can try out new functionality in preview environments. For more information, see Prepare for major updates with preview environments.

See also

The Business Central Administration Center
Managing Environments
Preparing Demonstration Environments
Prepare for major updates with preview environments
Administration of Business Central Online
Deployment of Dynamics 365 Business Central
Get Started as a Reseller of Business Central Online
Offer your customers trials of Microsoft products

Prepare for major updates with preview environments

2/17/2021 • 4 minutes to read • Edit Online

About one month before a major update, you can try out new functionality in preview environments. Preview environments are essentially Business Central online sandbox environments that you create on a preview version of the application. When you create the new sandbox environment, choose the preview version marked as *(Preview)* from the version list. This way, you get a new sandbox environment with a preview version of the application.

You must have access to Microsoft Collaborate in order to submit your feedback and report any potential issues that you discover in the preview version of the application. For more information about getting access to Collaborate, see Step 4: Getting access to preview bits.

> **NOTE**
> Previews roll out gradually across the world, so if the option is not showing up for you today, please try again tomorrow.

The newly created preview sandbox environment contains demonstration company data. Trying the preview on a copy of your current production data is not yet supported; nor is testing the upgrade from your current version to the preview. However, you can use the newly created sandbox environment for exploring and learning the new product capabilities, as well as validating that your per-tenant extensions are still working as expected.

If you run your tests on a preview environment one month before the announced major release of Business Central, it is more likely that the coming updates of your production environments will go smoother. This way, you, your customers, and your code are better prepared for the official release.

We expect to update the preview version only if we discover critical issues before the major update is generally available for production environments. Apart from these potential fixes, we do not expect any further changes to the product between the preview and the official release. This means that you can start your testing and learning activities immediately, without waiting for the official release.

> **NOTE**
> You will be able to test the update on a copy of your production data in a sandbox environment when we release the new update in production in April or October, respectively. When the official release becomes available, you can continue your tests on that version. You will no longer be able to create new preview sandboxes.

> **IMPORTANT**
> The preview version as well as all sandbox environments that are based on it will be removed 30 days after the official release becomes available.

Practice and test

Once you have the preview, start using it:

1. Review the new functionality. Try it out, and begin training employees on the new features that are coming.

2. Validate your extensions.

   Upload and install your extensions into the sandbox environments created on the preview version and run through the functionality. Verify that the customization continues to work and is compatible with the new version.

   In rare cases, if you discover any changes required for your per-tenant extension to become compatible with the next release, apply the changes to your app, test it again on a sandbox environment running on a preview version. Then, if tests complete successfully, upload the app into your production environment, setting **Deploy to** field to **Next major version**. This way the compatible version of your app will be used when you schedule upgrade of your production environment to the new major update once it becomes available.

3. Test the quality. If you run into issues related to the preview, please provide feedback on Collaborate.

Finally, as always, if you have ideas for features you would like to see in future releases of Business Central, let us know at https://aka.ms/bcideas.

Provide feedback on the preview

We need your feedback on the preview! Let us know about your experiences with the new version. You can provide feedback on Microsoft Collaborate:

1. Browse to aka.ms/collaborate
2. If you have not registered before, please complete the registration form to access Collaborate
3. Choose **Engagements**, and then choose **Join**
4. Under the **Ready! for Dynamics 365 Business Central program**, find **2020 release wave 2 Preview** in the list of available engagements, and choose **Join**

Once you have joined the preview engagement, you can submit your feedback by filling in the form. You must specify if you are reporting a technical issue, a translation issue, or a documentation issue. The default severity is set to 3, which is the normal severity for most defects and means that the system is not crashing and users are not blocked from doing their work, but you can change that. When we have triaged your issue, you can see changes in Collaborate.

> **TIP**
> If you get stuck trying to join Collaborate, please contact dyn365bep@microsoft.com with as many details about the error that you got and other circumstances.

See also

Major Updates of Business Central Online
Managing Major and Minor Updates of Business Central Online
Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Tenant Notifications
Step 4: Getting access to preview bits

Monitoring and Analyzing Telemetry

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

Business Central emits telemetry data for various activities and operations on tenants and extensions. Whether running Business Central Online or On-premises, you can set up your tenants to send telemetry to Application Insights. Application Insights is a service hosted within Azure that gathers telemetry data for analysis and presentation. For more information, see What is Application Insights?. Monitoring telemetry gives you a look at the activities and general health of your tenants. It helps you diagnose problems and analyze operations that affect performance.

Tenant-level and extension-level telemetry

Application Insights can be enabled on two different levels: tenant and extension. When enabled on the tenant, either for a Business Central online tenant or on-premises Business Central Server instance, telemetry is emitted to a single Application Insights resource for gathering data on tenant-wide operations.

With Business Central 2020 release wave 2 and later, Application Insights can also be enabled on a per-extension basis. An Application Insights key is set in the extension's manifest (app.json file). At runtime, certain events related to the extension are emitted to the Application Insights resource. This feature targets publishers of per-tenant extensions to give them insight into issues in their extension before partners and customers report them.

Available telemetry

In Application Insights, telemetry from Business Central is logged as traces. Currently, Business Central offers telemetry on the following operations:

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
App key vault secrets	Provides information about the retrieval of secrets from Azure Key Vaults by extensions.	✓ [1]	✓ [1]	✓	See...
Authorization	Provides information about user sign-in attempts. Information includes success or failure indication, reason for failure, user type, and more.	✓			See...

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
Company lifecycle	Provides information about creating, copying, and deleting of companies.	✓	✓		See...
Configuration package lifecycle	Provides information about operations done on configuration packages, including exporting, importing, applying, and deleting.	✓	✓		See...
Database lock timeouts	Provides information about database locks that have timed out.	✓	✓		See...
Email	Provides information about the success or failure of sending emails.	✓	✓		See...
Extension lifecycle [2]	Provides information about the success or failure of extension-related operations, like publishing, synchronizing, installing, and more.	✓	✓	✓	See...
Extension update	Provides information about errors that occur when upgrading an extension.	✓	✓	✓	See...
Field monitoring trace	Provides information about the usage of the field monitoring feature.	✓	✓		See...

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
Job queue	Provides information about creating and running job queue entries.	✓			See...
Long running AL method trace	Provides information about long running AL methods.	✓	✓		See...
Long running operation (SQL query)	Provides information about SQL queries that take longer than expected to execute.	✓	✓	✓	See...
Page views	Provides information about the pages that users open in the modern client.	✓			See...
Permissions	Provides information about adding, removing, and assigning permission sets.	✓			See...
Report generation	Provides information about the execution of reports.	✓	✓	✓	See...
Incoming web service requests	Provides information about the execution time of incoming web service requests.	✓	✓	✓	See...
Outgoing web service requests	Provides information about the execution time of outgoing web service requests.	✓	✓	✓	See...

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
Web service access key authentication	Provides information about the authentication of web server access keys on web service requests.	✔	✔		See...

[1]This signal is only emitted to the Application Insights resource that's specified in the extension.

[2]Introduced in Business Central 2020 release wave 1, version 16.3. For extension telemetry, this signal was introduced in 2020 release wave 2, version 17.1.

Enabling Application Insights

Sending telemetry data to Application Insights requires you have an Application Insights resource in Azure. Once you have the Application Insights resource, you can start to configure your tenants and extensions to send telemetry data to your Application Insights resource. The configuration is different for Online and On-premises:

- For Business Central Online, Application Insights is enabled by using the Administration Center. For more information, see Enable Sending Telemetry to Application Insights.
- For Business Central On-premises, see Enable Sending Telemetry to Application Insights.
- For extensions, see Sending Extension Telemetry to Azure Application Insights.

Viewing telemetry data in Application Insights

Telemetry from Business Central is stored in Azure Monitor Logs in the *traces* table. You can view collected data by writing log queries. Log queries are written in the Kusto query language (KQL). For more information, see Logs in Azure Monitor and Overview of log queries in Azure Monitor.

As a simple example, do the following steps:

1. In the Azure portal, open your Application Insights resource.
2. In the **Monitoring** menu, select **Logs**.
3. On the **New Query** tab, type the following to get the last 100 traces:

```
traces
| take 100
| sort by timestamp desc
```

About Custom Dimensions

Each trace includes a `customDimensions` column. The `customDimensions` column, in turn, includes a set dimensions that contain metrics specific to the trace. Each of these custom dimensions has a limit of 8000 characters. If a dimension's value exceeds 8000 characters, additional dimensions will be added to the trace for containing the excess characters. There can be up to two additional parameters, each with a maximum 8000 characters. The additional dimensions will have the names that `<custom_dimension_name>_1` and `<custom_dimension_name>_2`, where `<custom_dimension_name>` is the name of the original dimension. For example, if the custom dimension is `extensionCompilationDependencyList`, then the additional dimensions would be `extensionCompilationDependencyList_1` and `extensionCompilationDependencyList_2`.

> **NOTE**
>
> The 8000 character limit is governed by the Application Insights API.

Application Insights sample code

On the Business Central BCTech repository on GitHub, samples of KQL code are available to make it easy to get started using Application Insights.

For more information, see Business Central BCTech repository on GitHub.

See also

Telemetry Event IDs
Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Tenant Notifications
Introduction to automation APIs
LogMessage Method

Analyzing App Key Vault Secret Trace Telemetry

2/17/2021 • 6 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

App key vault telemetry gathers information about the acquisition of secrets in Azure Key Vaults by extensions at runtime. For an overview of app key vaults and secrets, see Using App Key Vaults with Business Central Extensions.

The app key vault secret process has two operations: *initialization* and *retrieval*. The telemetry data provides information about the success or failure for each of these operations. There are various conditions that cause a failure. The failure messages provide insight into the cause of the failure, helping you identify, troubleshoot, and resolve issues.

Initialization

Initialization is the first stage. It verifies the configuration of the app key vault provider in the extension and on the service. This stage is initiated by the `TryInitializeFromCurrentApp` method call in the extension code. Some conditions that cause failures in this stage include:

- The extension doesn't specify a key vault in it's app.json file.
- The Azure Key Vault Client Identity settings are incorrect. For example, it could be that the application (client) ID that you specified for the key vault reader application in Azure is wrong.
- The Business Central Server lacks permission to the private key of the Azure Key Vault client certificate.

Retrieval

Retrieval is the second stage, and occurs after a successful initialization. In this stage, the service tries to get a secret from a specified key vault. This stage is initiated by the `GetSecret` method call in the extension code. Some conditions that cause failures include:

- The secret name requested by the extension is doesn't exist or isn't valid.
- The key vault doesn't exist.
- The application ID doesn't have permission to read from the key vault.

For more information about using key vault secrets with extensions, see App Key Vaults with Business Central Extensions.

App Key Vault secret initialization succeeded

Occurs when an extension secret was successfully initialized.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault initialization succeeded: '{keyVaultUri}'.**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the request, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0014**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.
keyVaultUrls	Specifies the DNS name of the Azure key vault that was used in the request. The keyVaultUris are specified in the app.json file of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

App Key Vault initialization failed

Occurs when a key vault failed to be initialized.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault initialization failed.**

DIMENSION	DESCRIPTION OR VALUE
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed request, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0015**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.
failureReason	Specifies the error that occurred.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

App Key Vault secret retrieval succeeded

Occurs when a secret used by an extension is successfully retrieved from an Azure Key Vault.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault secret retrieval succeeded from key vault '{keyVaultUri}'.**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0016**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.
keyVaultUrl	Specifies the DNS name of the Azure key vault that was used in the request. The keyVaultUris are specified in the app.json file of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

App Key Vault secret retrieval failed

Occurs when an extension failed to retrieve a secret from a specified Azure key vault.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault secret retrieval failed from key vault '{keyVaultUri}'.**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0017**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.

DIMENSION	DESCRIPTION OR VALUE
keyVaultUrl	Specifies the DNS name of the Azure key vault that was used in the request. The keyVaultUris are specified in the app.json file of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

App Key Vaults with Business Central Extensions
Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Authorization Trace Telemetry

2/17/2021 • 10 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

Authorization telemetry provides information about the authorization of users when they try to sign in to Business Central. This telemetry data can help you identify problems a user might experience when signing in.

Authorization signals are emitted in two stages of sign-in. The first stage is the initial authorization, before the CompanyOpen trigger is run. In this stage, the system verifies that the user account is enabled in the tenant and has the correct entitlements. The telemetry data includes:

- Success or failure of the sign-in attempt
- Reason for failure
- Type of user (such as normal, administrator, or delegated user)
- Whether the user belongs to the tenant or is an invited user

The next stage occurs after a successful authorization attempt, when trying to open the company (that is, when the CompanyOpen trigger run). The telemetry data indicates whether the company opened successfully or failed (for some reason).

> **NOTE**
> Business Central 2020 release wave 1, version 16.1, introduces changes to some `operation_Name` and `message` dimension values. The differences from earlier versions are indicated in the following sections.

AuthorizationSucceeded(PreOpenCompany)

Occurs when a user has been successfully authorized.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later: **AuthorizationSucceeded(PreOpenCompany)** Before Version 16.1: **Authorization steps prior to the open company trigger succeeded.**
operation_Name	**AuthorizationSucceeded(PreOpenCompany)** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
authorizationStatus	**Succeeded**
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
entitlementSetIds	Specifies the entitlements that the user has in Business Central.
eventId	**RT0003**
guestUser	**true** indicates that the user is a guest user on the tenant. **false** indicates the user belongs to the tenant.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
userType	Specifies whether the user is a **Delegated_admin**, **Internal_Admin**, or **Normal user**. See UserType.

UserType

VALUE	DESCRIPTION	SEE MORE
Delegated_admin	Indicates that the user is a delegated administrator on the tenant. Delegated administrators are typically reserved for partners. Delegated administrator privileges are granted to users by the customer. You grant these privileges by setting up a Partner Relationship in the Microsoft Partner Center.	Delegated Administrator Access to Business Central Online Customers delegate administration privileges to partners
Internal_Admin	Indicates that the user is an internal administrator on the tenant. As an internal administrator, the user is assigned the **Global admin** role in the Microsoft 365 admin center.	Administration as an internal administrator in Business Central Assign admin roles in Microsoft 365 admin center
Normal user	Indicates that the user is a normal user in the tenant, based on the license.	Create Users According to Licenses

> **NOTE**
>
> The client type is not known when the server emits the pre-open company events (RT0001 and RT0003). You need to join to data for the events RT0002/RT0004 if you need both userType and clientType.

AuthorizationFailed(PreOpenCompany)

Occurs when a user sign-in has failed authorization.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later (depending on the cause): • **AuthorizationFailed(PreOpenCompany): User is disabled.** • **AuthorizationFailed(PreOpenCompany): User has no entitlements.** Before Version 16.1: **Authorization steps prior to the open company trigger failed, see failureReason column for details.**
operation_Name	**Authorization Failed (Pre Open Company)** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
authorizationStatus	**Failed**
failureReason	Specifies why the sign-in failed. See Troubleshooting failures section for details.
guestUser	**true** indicates that the user is a guest user on the tenant. **false** indicates the user belongs to the tenant.
eventId	**RT0001**
userType	Specifies whether the user is a **Delegated_admin**, **Internal_Admin**, or **Normal user**. See UserType.

Troubleshooting failures

The user was successfully authenticated in Azure Active Directory but the user account is disabled in Business Central.

This message occurs when the user's account is valid, but the account is disabled. If you open the user account in Business Central, you'll see the **State** field is set to **Disabled**.

Resolution

Enable the user account by setting the **State** field to **Enabled**. For more information, see Create Users According to Licenses.

A user successfully authenticated in Azure Active Directory but the user does not have any entitlements in Business Central.

This message occurs if the user has an account, but the account hasn't been assigned any entitlements.

Entitlements are part of the license. Entitlements are permissions that describe which objects in Business Central a user can use, according to their Azure Active Directory role or license. For an explanation of entitlements, see Business Central entitlements explained

Resolution

Entitlements are assigned to the user account in the Microsoft 365 admin center or Microsoft Partner Center. They aren't assigned in Business Central. To assign entitlements to a user, see one of the following articles:

- From Microsoft 365 admin center, see Add users individually or in bulk to Microsoft 365.
- From the Microsoft Partner Center, see User management tasks for customer accounts.

Authorization Succeeded (Open Company)

Occurs when the company has opened successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later: **Authorization Succeeded (Open Company)** Before version 16.1: **Authorization steps in the open company trigger succeeded.**
operation_Name	**Authorization Succeeded (Open Company)** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
authorizationStatus	**Success**
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that opened the session, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.

DIMENSION	DESCRIPTION OR VALUE
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0004**
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to open the company**. The time has the format hh:mm:ss.sssssss. Doesn't apply to the **Cancellation report generation** trace.
sqlExecutes	Specifies the number of SQL statements that the report executed**. Doesn't apply to the **Cancellation report generation** trace.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements**. Doesn't apply to the **Cancellation report generation** trace.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to open the company**. The time has the format hh:mm:ss.sssssss. Doesn't apply to the **Cancellation report generation** trace.

** From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Authorization Failed (Open Company)

Occurs when a company has failed to open.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later (depending on the cause): • **Authorization Failed (Open Company): Invalid company name.** • **Authorization Failed (Open Company): User has no permission to company.** • **Authorization Failed (Open Company): The tenant is locked.** • **Authorization Failed (Open Company): The license has expired or the trial period has ended.** • **Authorization Failed (Open Company): The user's license is not valid for use on production companies.** Before version 16.1: **Authorization steps in the open company trigger failed, see failureReason column for details.**
operation_Name	**Authorization Failed (Open Company)** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
authorizationStatus	**Failed**
companyName	Specifies the name of the company that the user tried to open.
failureReason	Specifies why the sign-in failed. See Troubleshooting failures section for details.
eventId	**RT0002**

Troubleshooting failures

The company name is not valid, because the name is either empty or exceeds the maximum allowed length.

This message occurs when a user tries to sign in to a company whose name exceeds the maximum allowed length.

Resolution

This message typically occurs when a user tries to access a specific company in Business Center by entering a URL in the browser address, for example, `https://businesscentral.dynamics.com/?company=CRONUS%20International%20Ltd.`. If the name exceeds 30 characters, then this message occurs. Make sure that the user has the proper name of the company.

The user does not have permission to access the company.

This message occurs when a user account in Business Central doesn't have the proper permissions to the company.

Resolution

In Business Central, open the user account and modify the permissions the user to give them access to the company. For more information, see Assign Permissions to Users and Groups.

> **TIP**
> A good starting point is to look at the **Effective Permissions** that the user has on the company. You can do this from the user card by selecting **Effective Permissions** and setting the **Company** to the company in question.

The company doesn't exist.

This message occurs when a user tries to sign in to a company, but the company isn't found in Business Central.

Resolution

This message typically occurs when a user tries to access a specific company in Business Center by entering a URL in the browser address, for example, `https://businesscentral.dynamics.com/?company=CRONUS%20International%20Ltd.`. Make sure that the user has the proper name of the company.

User cannot open the company because the tenant is locked.

This message indicates that the tenant has been locked by Microsoft, typically for security reasons like preventing repeated malicious sign-in attempts. The tenant isn't accessible by any user.

Resolution

For help with resolving this issue, read the following articles or contact Microsoft Support:

- Troubleshoot account lockout problems with an Azure AD Domain Services managed domain
- "It looks like your account has been blocked" error when signing in to Microsoft 365

The user can't sign in to the company because the assigned license has expired or the trial period has ended.

This message occurs for one the following reasons:

- The license being used has expired.
- The license was trial license and the trial period has ended. Trial licenses are typically assigned when customers subscribe for an evaluation version by using self-service sign-up (also known as IW or viral sign-up). This license has a time limit.

Resolution

Renew the existing license or obtain a new license. Licenses are purchased through the Cloud Solution Provider (CSP) program. For more information, see the Cloud Service Provider site and the Microsoft Dynamics 365 Business Central Licensing Guide.

You can't open the company, because it is a production company. Your license isn't valid for use on production companies.

This message occurs because the license doesn't allow the user to open production companies. For example, the user may be using a trial license that is only valid on the evaluation version.

Resolution

Obtain a license that can be used on production companies. Licenses are purchased through the Cloud Solution Provider (CSP) program. For more information, see the Cloud Service Provider site and the Microsoft Dynamics 365 Business Central Licensing Guide.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Company Lifecycle Trace Telemetry

2/17/2021 • 11 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.1

Company lifecycle telemetry gathers data about the success or failure of the following company-related operations:

- creating a company
- copying a company
- deleting a company

Failed operations result in a trace log entry that includes a reason for the failure.

Company created

Occurs when the company has been successfully created.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company created: {companyName}** `{companyName}` indicates the name of the new company.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.

DIMENSION	DESCRIPTION OR VALUE
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0001**
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to create the company. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to create the company. The time has the format hh:mm:ss.sssssss.

Company creation canceled

Occurs when creating a company was canceled.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company creation canceled: {companyName}** `{companyName}` indicates the name of the company being created.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.

DIMENSION	DESCRIPTION OR VALUE
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0002**
failureReason	**Operation was canceled**
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server create the company before being canceled. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to create the company before being canceled. The time has the format hh:mm:ss.sssssss.

Company creation failed

Occurs when a company failed to be created.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company creation failed: {companyName}** `{companyName}` indicates the name of the company being created.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0003**
failureReason	Specifies the exception that indicates the cause of the failure.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server create the company before it failed. The time has the format hh:mm:ss.ssssss.
sqlExecutes	Specifies the number of SQL statements that were executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to create the company before it failed. The time has the format hh:mm:ss.ssssss.

Company copied

Occurs when a company has been copied from another company successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company copied: {companyNameSource} to {companyNameDestination}** • `{companyNameSource}` is name of the company that was copied. • `{companyNameDestination}` is the name of new company that was created.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyNameDestination	Specifies the name of the new company.
companyNameSource	Specifies the name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LT0004**
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to copy the company. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.

DIMENSION	DESCRIPTION OR VALUE
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to copy the company. The time has the format hh:mm:ss.sssssss.

Company copy canceled

Occurs when a copying a company was canceled.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company copied canceled: {source company name} to {destination company name}** • `{companyNameSource}` is name of the company that was being copied. • `{companyNameDestination}` is the name of new company that was being created.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed operation, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0005**
failureReason	**Operation was canceled**.
result	**Failure**

DIMENSION	DESCRIPTION OR VALUE
serverExecutionTime	Specifies the amount of time it took the server to copy the company. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to do the operation. The time has the format hh:mm:ss.sssssss.

Company copy failed

Occurs when a company failed to be copied from another company.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company copy failed: {companyNameSource} to {companyNameDestination}** • `{companyNameSource}` is name of the company that was being copied. • `{companyNameDestination}` is the name of new company that was being created.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types

DIMENSION	DESCRIPTION OR VALUE
eventId	**LC0006**
failureReason	Specifies the exception that indicates the cause of the failure.
result	**Failure**
serverExecutionTime	Specifies the amount of time on the server. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed.
sqlRowsRead	Specifies the number of table rows that by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to create the company before it failed. The time has the format hh:mm:ss.sssssss.

Company deleted

Occurs when a company has been deleted successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company deleted: {companyName}** `{companyName}` indicates the name of the company that was deleted.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0007**
result	**Success**
serverExecutionTime	Specifies the amount of time it took on server. The time has the format hh:mm:ss.ssssss.
sqlExecutes	Specifies the number of SQL statements that the operation executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to delete the company. The time has the format hh:mm:ss.sssssss.

Company deletion canceled

Occurs when deleting a company failed was canceled.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company deletion canceled: {companyName}** `{companyName}` indicates the name of the company that was being deleted.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.

DIMENSION	DESCRIPTION OR VALUE
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0008**
failureReason	**Operation was canceled**
result	**Failure**
serverExecutionTime	Specifies the amount of time it took on server. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to delete the company before being canceled. The time has the format hh:mm:ss.sssssss.

Company deletion failed

Occurs when a company failed to be deleted.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company deletion failed: {companyName}** `{companyName}` indicates the name of the company that was being deleted.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.

DIMENSION	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0009**
failureReason	Specifies the exception that caused the failure.
result	**Failed**
serverExecutionTime	Specifies the amount of time it took on server. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to delete the company before it failed. The time has the format hh:mm:ss.sssssss.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Configuration Package Lifecycle Trace Telemetry

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, version 17.2, and later

Configuration package telemetry gathers data about the following operations on configuration packages:

- Export
- Import
- Apply
- Delete

For information about working with configuration packages, see Prepare a Configuration Package in the Business Central Application Help.

Configuration package export started

Occurs when an export operation on a configuration package is started.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package export started: {alPackageCode}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3F**
alExecutionId	Specifies the ID of the export operation.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the export of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the export of configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package being exported.

Common custom dimensions

The following table explains additional custom dimensions that are common to all configuration package traces.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**RapidStart**
alDataClassification	**SystemMetadata**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Configuration package exported successfully

Occurs when an export operation on a configuration package completes successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package exported successfully: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3G**
alExecutionId	Specifies the ID of the export operation.
alExecutionTimeInMs	Specifies the number of milliseconds it took to complete the export operation.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the export of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the export of configuration packages.
alObjectType	**CodeUnit**

DIMENSION	DESCRIPTION OR VALUE
alPackageCode	Specifies the ID of the configuration package that was exported.
See common custom dimensions	

Configuration package import started

Occurs when an import operation on a configuration package is started.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package import started: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3H**
alExecutionId	Specifies the ID of the import operation.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the import of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the import of configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package being imported.
See common custom dimensions	

Configuration package imported successfully

Occurs when an import operation on a configuration package completes successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package imported successfully: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3I**
alExecutionTimeInMs	Specifies the number of milliseconds it took to complete the import operation.
alExecutionId	Specifies the ID of the import operation.
alPackageCode	Specifies the ID of the configuration package that was imported.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the import of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the import of configuration packages.
alObjectType	**CodeUnit**
See common custom dimensions	

Configuration package apply started

Occurs when an apply operation on a configuration package is started.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package apply started: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3N**
alExecutionId	Specifies the ID of the apply operation.
alObjectId	**8611**, which is the ID of the base application codeunit that handles applying configuration packages.
alObjectName	**Config. Package Management**, which is the name of the base application codeunit that handles applying configuration packages.
alObjectType	**CodeUnit**

DIMENSION	DESCRIPTION OR VALUE
alPackageCode	Specifies the ID of the configuration package being applied.
See common custom dimensions	

Configuration package applied successfully

Occurs when an apply operation on a configuration package completes successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package applied successfully: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3O**
alExecutionId	Specifies the ID of the apply operation.
alExecutionTimeInMs	Specifies the number of milliseconds it took to complete the apply operation.
alErrorCount	Specifies the number of errors that occurred when applying the configuration package.
alFieldCount	Specifies the number of fields that were included in the migration table of the applied configuration package.
alObjectId	**8611**, which is the ID of the base application codeunit that applies configuration packages.
alObjectName	**Config. Package Management**, which is the name of the base application codeunit that applies configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package that was applied.
alRecordCount	Specifies the number of records that were included in the applied configuration package.
See common custom dimensions	

Configuration package deleted successfully

Occurs when a configuration package is deleted successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package deleted successfully: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3P**
alObjectId	**8611**, which is the ID of the base application codeunit that deletes configuration packages.
alObjectName	**Config. Package Management**, which is the name of the base application codeunit that handles deleting configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package that was deleted.
See common custom dimensions	

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Prepare a Configuration Package in the Business Central

Analyzing Database Lock Timeout Trace Telemetry

2/17/2021 • 5 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.0

Database lock timeout telemetry gathers information about database locks that have timed out. The telemetry data allows you to troubleshoot what caused these locks.

In the client, when a lock has timed out, the user is presented with a message, similar to the following message:

The operation could not complete because a record in the [table name] table was locked by another user. Please retry the activity.

Two types of trace events are emitted to Application Insights:

- The first is a **Database lock timed out** event. This event includes general information about the lock request. This event includes information like the AL object and code that is impacted by the lock, the extension involved, and more.
- The **Database lock timed out** event then triggers one or more **Database lock snapshot** events. **Database lock snapshot** events provide details about SQL sessions that hold database locks at the time of lock timeout, including the session that caused the lock timeout. These events include specific details about the SQL lock request on the database, like the type, status, mode, and the table.

> **TIP**
> When analyzing database lock timeout telemetry, it's useful to look at combined data from the **Database lock timed out** event and **Database lock snapshot** events. You can combine data from different events by using *joins* in your Kusto queries. For an example, see LockTimeouts.kql in the **Microsoft/BCTech** repository on GitHub. For more general information about using joins, see Joins in Azure Monitor log queries in the Microsoft Azure documentation.

Database lock timed out

Occurs when a database lock has timed out.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Database lock timed out**
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.

DIMENSION	DESCRIPTION OR VALUE
alExecutingMethodScope	Specifies the AL action that is running the transaction that caused the lock.
alObjectId	Specifies the ID of the running AL object that requested the lock.
alObjectName	Specifies the name of the running AL object that requested the lock. not shown
alObjectType	Specifies the type of the running AL object that requested the lock, such as a page or report.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0012**
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
extensionId	Specifies the AppID of the extension that was involved in the lock.
extensionName	Specifies the name of the extension that was involved in the lock.
extensionVersion	Specifies the version of that was involved in the lock.
sessionId	Specifies the ID of the session that requested the lock.
snapshotId	Specifies the ID of the database snapshot. This ID is used to identify associated **Database lock snapshot** trace events.

DIMENSION	DESCRIPTION OR VALUE
sqlServerSessionId	Specifies the ID of the SQL server session that requested the lock.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Database lock snapshot

Occurs when a database lock has timed out. Each **Database lock snapshot** trace event is associated with a specific **Database lock timed out** trace event.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Database lock snapshot: {snapshotId}** The value of the `{snapshotId}` maps to the `snapshotId` dimension of the **Database lock timed out** trace event that triggered this event.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alExecutingMethodScope	Specifies the AL action that is running the transaction that caused the lock.
alObjectId	Specifies the ID of the running AL object that requested the lock.
alObjectName	Specifies the name of the running AL object that requested the lock.
alObjectType	Specifies the type of the running AL object that requested the lock, such as a page or report.
alStackTrace	The stack trace in AL.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0013**
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
extensionId	Specifies the AppID of the extension that was involved in the lock.
extensionName	Specifies the name of the extension that was involved in the lock.
extensionVersion	Specifies the version of that was involved in the lock.
sessionId	Specifies the ID of the session that requested the lock.
snapshotId	Specifies the ID of the database snapshot. All messages in the snapshot share this ID.
sqlLockRequestMode	Specifies the lock mode that determines how concurrent transactions can access the resource. For more information, see Lock Modes.
sqlLockRequestStatus	Specifies the current status of the lock, which can be one of the following values: • `CNVRT` - means that the lock is transitioning from another mode, but the conversion is blocked by another process that holds a lock with a conflicting mode. • `GRANT` - means that the lock is active. • `WAIT` - means that the lock is blocked by another process that holds a lock with a conflicting mode.
sqlLockResourceType	Specifies the database resource affected by the lock. For example, `DATABASE`, `FILE`, `OBJECT`, `PAGE`, `KEY`, and more.
sqlServerSessionId	Specifies the ID of the SQL server session that requested the lock.
sqlTableName	Specifies the name of table on which the lock was held.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Email Trace Telemetry

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, update 17.2, and later

Email telemetry gathers data about the following operations:

- An email was sent successfully
- An attempt to send an email failed

Before you can collect this data, you'll have to set up email. For more information, see Set Up Email in the Business Central application help.

Email sent successfully

Occurs when an email was successfully sent from the client.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Email sent successfully**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**Email**
alConnector	Specifies the email-provider connector used to send the email. Possible values include: • Current User • Microsoft 365 • SMTP • Other custom connectors installed by extensions. The connector is specified on the email accounts that are set up in Business Central. For more information, see Adding Email Accounts.
alDataClassification	**SystemMetadata**
alEmailMessageID	Specifies the GUID assigned to email, like C7A56676-9F3F-4044-90F0-D7F3196AC366.
alObjectId	**8888**, which is the ID of the system application codeunit that sends emails.

DIMENSION	DESCRIPTION OR VALUE
alObjectName	**Email Dispatcher**, which is the name of the system application codeunit that sends the emails.
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**AL0000CTV**
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Failed to send email

Occurs when an email failed to be sent from the client.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Failed to send email.**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**Email**
alConnector	Specifies the email-provider connector used to send the email. Possible values include: • Current User • Microsoft 365 • SMTP • Other custom connectors installed by extensions. The connector is specified on the email accounts that are set up in Business Central. For more information, see Adding Email Accounts.

DIMENSION	DESCRIPTION OR VALUE
alDataClassification	**SystemMetadata**
alEmailMessageID	Specifies the GUID assigned to email, like C7A56676-9F3F-4044-90F0-D7F3196AC366.
alObjectId	**8888**, which is the ID of the system application codeunit that sends emails.
alObjectName	**Email Dispatcher**, which is the name of the system application codeunit that sends the emails.
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**AL0000CTP**
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

TIP

You can also view failed emails in the **Email Outbox** page in the Business Central client.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Extension Lifecycle Trace Telemetry

2/17/2021 • 30 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.3. For extension telemetry, this signal was introduced in 2020 release wave 2, version 17.1

Extension lifecycle telemetry gathers data about the success or failure of the following extension-related operations:

- Compiling an extension
- Synchronizing an extension
- Publishing an extension
- Installing an extension
- Updating an extension
- Uninstalling an extension
- Unpublishing an extension

Failed operations result in a trace log entry that includes a reason for the failure.

Data is gathered when the operations are done using:

- Extension Management page in the client.
- Manage Apps page in the Business Central administration center.
- Extension management PowerShell cmdlets from the Business Central Administration Shell.

Behavior overview

- Compiling, publishing, and unpublishing extensions

  Data for these operations is only recorded for extensions that are published in the tenant scope.

  - For on-premises, it includes extensions that are published by running the Publish-NAVApp cmdlet with the `-Scope Tenant` parameter.
  - For online, it includes per-tenant extensions uploaded from the **Extension Management** page in the client. It doesn't include Microsoft extensions or AppSource extensions.

- Synchronizing extensions

  - For on-premises, data for this operation is recorded when an extension is synchronized by using the Sync-NAVApp cmdlet.
  - For online, data is recorded when an extension is installed from the **Extension Management** page in the client. Or, when upgraded from the Manage Apps page in the Business Central administration center.

- Installing and uninstalling extensions

  - For on-premises, data for these operations is recorded when an extension is installed or uninstalled by using the Install-NAVApp cmdlet or Uninstall-NAVApp cmdlet. Or, when an extension is installed or uninstalled from the **Extension Management** page in the client.
  - For on-premises, data for these operations is recorded when an extension is installed or uninstalled from the **Extension Management** page in the client.

- Updating an extension
  - For on-premises, data for this operation is recorded when an extension is upgraded by using the Start-NAVAppDataUpgrade cmdlet.
  - For online, data is recorded when an extension is updated from the Manage Apps page in the Business Central administration center.
- For some operations, you might experience that certain custom dimensions aren't available.

  The reason is that custom dimensions are added to the signal gradually, as the information is retrieved. If the operation fails before the custom dimension is retrieved, it isn't included in the result.

  For example, if you try to uninstall an extension using the Ininstall-NAVApp cmdlet, and the specified extension name is wrong, the operation fails. In this case, the `extensionid` and `extensionVersion` will be excluded from the results.

ENVIRONMENT/SERVER-SIDE OPERATIONS

Extension compiled successfully

Occurs when an extension compiles successfully on the service. An extension compiles when it's published or when it's repaired by using the Repair-NAVApp cmdlet.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension compiled successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0020**

DIMENSION	DESCRIPTION OR VALUE
extensionCompilationDependencyList	Specifies details about the extensions on which the compiled extension has dependencies. **Note:** If the value exceeds 8000 characters, one or two additional dimensions will be included in the trace to cover the complete dependency list. For more information, see About Custom Dimensions.
extensionCompilationResult	**Compilation succeeded without errors or warnings.**
extensionName	Specifies the name of the extension that was compiled.
extensionId	Specifies the AppID of the extension that was compiled.
extensionPublishedAs	Specifies whether the compiled extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the compiled extension.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Extension failed to compile

Occurs when an extension failed to compile on the service. An extension compiles when it's published or when it's repaired by using the Repair-NAVApp cmdlet.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to compile: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0021**
extensionCompilationDependencyList	Specifies details about the extensions on which the compiled extension has dependencies. **Note:** If the value exceeds 8000 characters, one or two additional dimensions will be included in the trace to cover the complete dependency list. For more information, see About Custom Dimensions.
extensionCompilationResult	Specifies details about the error that occurred during compilation.
extensionName	Specifies the name of the extension that failed to compile.
extensionId	Specifies the AppID of the extension that failed to compile.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the compiled extension.
failureReason	Specifies the error that occurred when compiling the extension.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Extension published successfully

Occurs when an extension published successfully on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension published successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0014**
extensionId	Specifies the AppID of the extension that was published.
extensionIsRad	Specifies whether the extension that was RAD published. **True** indicates the extension was RAD published. **False** indicates normal publishing. RAD (Rapid Application Development) publishing is done from the AL development environment. RAD publishing is a partial publishing operation that only publishes objects application objects that have changed during development. For more information, see Working with Rapid Application Development.
extensionName	Specifies the name of the extension that published.
extensionId	Specifies the AppID of the extension that published.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the published extension.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.

DIMENSION	DESCRIPTION OR VALUE
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to publish

Occurs when an extension failed publish on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0015**
extensionId	Specifies the AppID of the extension that was published.

DIMENSION	DESCRIPTION OR VALUE
extensionIsRad	Specifies whether the extension that was RAD published. **True** indicates the extension was RAD published. **False** indicates normal publishing. RAD (Rapid Application Development) publishing is done from the AL development environment. RAD publishing only is a partial publishing operation that only publishes objects application objects that have changed during development. For more information, see Working with Rapid Application Development.
extensionName	Specifies the name of the extension that published.
extensionId	Specifies the AppID of the extension that published.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the published extension.
failureReason	Specifies the error that occurred when publishing.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension unpublished successfully

Occurs when an extension was unpublished successfully on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension unpublished successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0018**
extensionId	Specifies the AppID of the extension that was unpublished.
extensionName	Specifies the name of the extension that was unpublished.
extensionId	Specifies the AppID of the extension that was unpublished.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the unpublished extension.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to un-publish

Occurs when an extension fails to unpublish on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to un-publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0019**
extensionId	Specifies the AppID of the extension that failed to unpublish.
extensionName	Specifies the name of the extension that failed to unpublish.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the unpublished extension.
failureReason	Specifies the error that occurred when unpublishing.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

TENANT OPERATIONS

Extension synchronized successfully

Occurs when an extension synchronizes successfully on the tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension synchronized successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0012**
extensionId	Specifies the AppID of the extension that was synchronized.
extensionName	Specifies the name of the extension that was synchronized.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionSynchronizationMode	Specifies whether the extension was synchronized in one of the following modes: • **Add** - The database schema defined by the objects in the extension are added to the database schema of the tenant database. This mode is typically used mode after you publish an extension for the first time. • **Clean** - The database schema defined by all versions of the extension will be removed from the database and all data is lost. This mode is typically used when an extension will no longer be used and all versions unpublished. • **Development** - This mode is acts similar to Add, except it is intended for use during development. It lets you to sync the same version of an extension that is already published. However, to run this mode, only one version the App can be currently published. • **ForceSync** - This mode like **Add** except it supports destructive schema changes (like removing fields, renaming them, changing their datatypes, and more). It is typically used during development, and is the mode used when an extension is published and installed from the AL development environment. For more information about the modes, see Sync-NAVApp cmdlet -Mode.
extensionVersion	Specifies the version of the extension was synchronized.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension synchronized failed

Occurs when an extension fails to synchronize on the tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to synchronize: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0013**
extensionId	Specifies the AppID of the extension that failed to synchronize.
extensionName	Specifies the name of the extension that failed to synchronize.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionSynchronizationMode	Specifies whether the extension was synchronized in one of the following modes: • **Add** - The database schema defined by the objects in the extension are added to the database schema of the tenant database. This mode is typically used mode after you publish an extension for the first time. • **Clean** - The database schema defined by all versions of the extension will be removed from the database and all data is lost. This mode is typically used when an extension will no longer be used and all versions unpublished. • **Development** - This mode is acts similar to Add, except it is intended for use during development. It lets you to sync the same version of an extension that is already published. However, to run this mode, only one version the App can be currently published. • **ForceSync** - This mode like **Add** except it supports destructive schema changes (like removing fields, renaming them, changing their datatypes, and more). It is typically used during development, and is the mode used when an extension is published and installed from the AL development environment. For more information about the modes, see Sync-NAVApp cmdlet -Mode.
extensionVersion	Specifies the version of the extension was synchronized.
failureReason	Specifies the error that occurred when synchronizing the extension.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension installed successfully

Occurs when an extension installs successfully on a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0010**
extensionId	Specifies the AppID of the extension that was installed.
extensionName	Specifies the name of the extension that failed to synchronize.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension was installed.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to install

Occurs when an extension failed to install on a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0011**
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that failed to uninstall.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension was installed.
failureReason	Specifies the error that occurred when the extension was installed.
result	**Failed**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

DIMENSION	DESCRIPTION OR VALUE
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension un-installed successfully

Occurs when an extension is successfully uninstalled from a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension un-installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
doNotSaveData	Specifies whether the uninstall operation was run with option not to save the data in database table fields that are added by the extension. When using the Uninstall-NAVApp cmdlet, this condition is set with the -DoNotSaveData switch parameter.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0016**
extensionId	Specifies the AppID of the extension that was uninstalled.
extensionName	Specifies the name of the extension that was uninstalled.

DIMENSION	DESCRIPTION OR VALUE
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension was uninstalled.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to un-install

Occurs when an extension failed to uninstall on a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to un-install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
doNotSaveData	Specifies whether the uninstall operation was run with option not to save the data in database table fields that are added by the extension. When using the Uninstall-NAVApp cmdlet, this condition is set with the -DoNotSaveData switch parameter.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0017**
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that failed to uninstall.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension that failed to uninstall.

DIMENSION	DESCRIPTION OR VALUE
failureReason	Specifies the error that occurred when the extension was uninstalled.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension updated successfully

Occurs when an extension updates successfully on the service.

> **NOTE**
> Data is also recorded for any upgrade code that was run. For more information, see Analyzing Extension Update Trace Telemetry.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension updated successfully: {extensionName} version {extensionVersion} by {extensionPublisher}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0022**
extensionCulture	Specifies the language version for which the extension that was upgraded. The value is a language culture name, such as **en-US** or **da-DK**. If a language wasn't specified when the extension was installed, then en-US is used by default.
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that was being upgraded.
extensionPublisher	Specifies the extension's publisher.
extensionVersion	Specifies the new version of the extension being upgraded.
extensionVersionFrom	Specifies the old version of the extension being upgraded.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to update

Occurs when an extension failed to update on the service.

> **NOTE**
> Data is also recorded for any upgrade code that was run. For more information, see Analyzing Extension Update Trace Telemetry.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to update: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0023**
extensionCulture	Specifies the language version for which the extension that was upgraded. The value is a language culture name, such as **en-US** or **da-DK**. If a language wasn't specified when the extension was installed, then en-US is used by default.
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that was being upgraded.
extensionPublisher	Specifies the extension's publisher.
extensionVersion	Specifies the new version of the extension being upgraded.

DIMENSION	DESCRIPTION OR VALUE
extensionVersionFrom	Specifies the old version of the extension being upgraded.
failureReason	Specifies the error that occurred during upgrade.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

See also

Upgrading Extensions
Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Extension Update Trace Telemetry

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.2

Extension upgrade telemetry gathers information about failures that occur during extension upgrades. Specifically, it provides information about exceptions that are thrown by code run by upgrade codeunits. The trace events include the following information:

- The codeunit that threw the exception.
- AL stack trace.
- Exception message.

This data can help you identify, troubleshoot, and resolve issues with per-tenant and AppSource extensions that are blocking data upgrade.

Extension Update Failed: exception raised in extension

Occurs when an extension upgrade fails because of an exception in an upgrade codeunit.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension Update Failed: exception raised in extension {extensionName} by {extensionPublisher} (updating to version {extensionTargetedVersion})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.
alStackTrace	The stack trace in AL.
companyName	The display name of the Business Central company that was used at time of execution.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0010**
extensionName	Specifies the name of the extension that was being upgraded.
extensionId	Specifies the AppID of the extension that was being upgraded.
extensionTargetedVersion	Specifies the new version of the extension being upgraded.
extensionVersion	Specifies the old version of the extension being upgraded.
failureReason	Specifies the exception that was thrown by the upgrade code. Some exception messages can contain customer data. As a precaution, Business Central only emits information that's classified as SystemMetadata. Exception messages that contain other data classifications, like customer data, are not shown. Instead, the following message is shown: "Message not shown because the NavBaseException(string, Exception, bool) constructor was used."
failureType	**DataUpdate**
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Example


```
{"Telemetry schema version":"0.3","telemetrySchemaVersion":"0.3","Component
version":"16.0.12582.0","componentVersion":"16.0.12582.0","Environment
type":"Production","environmentType":"Production","deprecatedKeys":"Company name, AL Object Id, AL Object
type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension version,
Telemetry schema version, AadTenantId, Environment name, Environment type, Component, Component
version","Company name":"CRONUS International Ltd.","AL Object Id":"0","AadTenantId":"tenant1-
1","companyName":"CRONUS International Ltd.","Client type":"Background","aadTenantId":"tenant1-
1","Component":"Dynamics 365 Business Central
Server","clientType":"Background","alObjectId":"0","extensionVersion":"5.0.0.0","component":"Dynamics 365
Business Central Server","eventId":"RT0010","extensionTargetedVersion":"5.0.0.1","Extension
version":"5.0.0.0","Extension App Id":"f0f770f4-8c9c-48de-adad-1c0281bef849","AL Stack
trace":"CustomerListExt(CodeUnit 50100).OnUpgradePerCompany line 7 - MyExtension by MyPublisher
publisher\n\"\"Upgrade Triggers\"\"(CodeUnit 2000000008).OnUpgradePerCompany line 2","Extension
name":"MyExtension","failureReason":"Attempted to divide by
zero.","extensionName":"MyExtension","alStackTrace":"CustomerListExt(CodeUnit 50100).OnUpgradePerCompany line
7 - MyExtension by MyPublisher publisher\n\"\"Upgrade Triggers\"\"(CodeUnit 2000000008).OnUpgradePerCompany
line 2","failureType":"DataUpdate","extensionId":"f0f770f4-8c9c-48de-adad-1c0281bef849"}
```


See also

Analyzing Field Monitoring Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 (update 17.1) and later

Keeping sensitive data secure and private is a core concern for most businesses. To add a layer of security, you can monitor important fields when someone changes a value. For example, you might want to know if someone changes your company's IBAN number.

To gather this data, you'll have to start field monitoring and specify the fields that you want to monitor. For more information, see Auditing Changes in Business Central - Monitoring Sensitive Fields in the Application help.

Telemetry is then logged for the following operations:

- When field monitoring is stopped or started
- When a field is added or removed for monitoring
- When a field value is changed

Sensitive field monitor status changed

Occurs when the field monitor feature is started or stopped in the company.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Sensitive field monitor status has changed to {almonitorStatus}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alDataClassification	**SystemMetadata**
almonitorStatus	**Yes** - the field monitor feature was started. **No** the field monitor feature was stopped.
alObjectId	**1392**
alObjectName	**Monitor Sensitive Field**
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**

DIMENSION	DESCRIPTION OR VALUE
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types. This dimension isn't included for Business Central on-premises environments.
eventId	**AL0000DD3**
extensionName	Specifies the name of the base application.
extensionId	Specifies the ID of the base extension.
extensionPublisher	Specifies the publisher of the extension.
extensionVersion	Specifies the version of the base application.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Sensitive field value has changed

Occurs the value of a monitored field has changed in the company.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Sensitive field value has changed: {alfieldCaption} in table {altableCaption}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.

DIMENSION	DESCRIPTION OR VALUE
alDataClassification	**SystemMetadata**
alfieldCaption	Specifies the name of the field that was changed.
altableCaption	Specifies the name of the table that the changed field is included.
alObjectId	**1392**
alObjectName	**Monitor Sensitive Field**
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types. This dimension isn't included for Business Central on-premises.
eventId	**AL0000CTE**
extensionName	Specifies the name of the base application.
extensionId	Specifies the ID of the base extension.
extensionPublisher	Specifies the publisher of the extension.
extensionVersion	Specifies the version of the base application.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

NOTE

Changes to fields in the following tables are always logged:

- Fields in the **Field Monitoring Setup** table. Many of the fields are included on the **Field Monitoring Setup** page.
- Fields in the **Change Log Setup (Field)** table. Many of the fields are included on the **Field Monitoring Worksheet** page.
- The **Contact Email** field in the **User** table.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Job Queue Lifecycle Trace Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, version 17.2, and later

Job queue lifecycle telemetry gathers data about the following operations:

- A job queue entry was enqueued.
- A job queue entry was started.
- A job queue entry finished. Provides information as to whether it was successful or failed.

For information about creating and managing job queue entries, see Use Job Queues to Schedule Tasks in the Business Central application help.

Job queue entry enqueued

Occurs when a job queue entry is sent to the job queue to eventually be run. A job queue entry is sent to the queue when its status is changed from **On Hold** to **Ready** or if it's a recurring job queue entry. Recurring job queue entries are automatically enqueued after each run.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Job queue entry enqueued: {alJobQueueId}**
severityLevel	**1**

Custom dimensions

The following table explains custom dimensions that are specific to this trace.

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E24**
alJobQueueId	Specifies the ID of the job queue entry.
alJobQueueIsRecurring	Specifies whether the job queue is recurring. **Yes** indicates it's recurring. **No** indicates it's not recurring.
alJobQueueObjectId	Specifies the ID of the object that the job queue entry runs.
alJobQueueObjectType	Specifies the type of the object that the job queue entry runs, for example **Report** or **Codeunit**.
alJobQueueStatus	**Ready** indicates it's a non-recurring job queue entry or the first run of a recurring job queue entry that's ready to run. **On Hold with Inactivity Timeout** indicates it's a recurring job query entry that's ready to run.

Common custom dimensions

The following table explains additional custom dimensions that are common to all job queue entry traces.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**AL JobQueueEntries**
alDataClassification	**SystemMetadata**
alObjectId	**1351**, which is the ID of the system application codeunit that subscribes to the telemetry events.
alObjectName	**Telemetry Subscribers**, which is the name of the system application codeunit that subscribes to the telemetry events.
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
extensionName	Specifies the name of the extension that contains the object run by the job queue entry.
extensionId	Specifies the ID of the extension that contains the object run by the job queue entry.
extensionVersion	Specifies the version of the extension that contains the object run by the job queue entry.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Job queue entry started

Occurs when a job queue entry starts to run.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Job queue entry started: {alJobQueueId}**
severityLevel	**1**

Custom dimensions

The following table explains custom dimensions that are specific to this trace.

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E25**
alJobQueueId	Specifies the ID of the job queue entry.
alJobQueueObjectId	Specifies the ID of the object that the job queue entry runs.
alJobQueueObjectType	Specifies the type of the object that the job queue entry runs, for example **Report** or **Codeunit**.
alJobQueueStatus	**In Process**
See common custom dimensions	

Job queue entry finished

Occurs when a job queue entry finishes running.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Job queue entry finished: {alJobQueueId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E26**
alJobQueueExecutionTimeInMs	Specifies how many milliseconds it took to run the job queue entry.
alJobQueueId	Specifies the ID of the job queue entry.
alJobQueueObjectId	Specifies the ID of the object that the job queue entry runs.
alJobQueueObjectType	Specifies the type of the object that the job queue entry runs, for example **Report** or **Codeunit**.

DIMENSION	DESCRIPTION OR VALUE
alJobQueueResult	**Success** indicates that the job queue entry ran successfully. **Fail** indicates that an error occurred when running the job queue.
alJobQueueStatus	**Finished** indicates that a non-recurring job queue entry has completed. **On Hold with Inactivity Timeout** indicates that a recurring the job queue entry completed.
See common custom dimensions	

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Use Job Queues to Schedule Tasks

Analyzing Long Running AL Methods Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later, version 17.1

The Business Central Server server will emit telemetry about the execution time of long running AL methods, including the time spent in the database. The signal also includes a breakdown of how much time each event subscriber added to the total time. As a partner, this data gives you insight into bad performing code and enables you to troubleshoot performance issues caused by extensions.

> **NOTE**
> To collect this telemetry for Business Central on-premises, the **AL Function Timing** and **AL Function Logging Threshold - Application Insights** settings must be configured on the Business Central Server instance. For more information, see Configuring Business Central Server. With Business Central online, this telemetry is enabled with a specific threshold on a case-by-case basis by the service.

General dimensions

The following table explains the general dimensions included in the trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Operation exceeded time threshold (AL method)**
severityLevel	2

CustomDimensions

This table describes the different dimensions of a **Operation exceeded time threshold (AL method)** operation.

COLUMN (KEY)	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID when using Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alMethod	The name of the long running AL method.
alObjectId	The type of the AL object that executed the AL method.
alObjectName	The name of the AL object that executed the AL method.
alObjectType	The type of the AL object that executed the AL method.
alStackTrace	The stack trace in AL.

COLUMN (KEY)	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the AL method, such as Background or Web. For a list of the client types, see ClientType Option Type.
companyName	The display name of the Business Central company that was used at time of execution.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type of the Business Central solution, such as Production or Sandbox.
environmentName	Specifies the environment name of the Business Central solution, such as Production or Sandbox.
eventId	**RT0018**
executionTime	Specifies the total time that it took to execute the AL method. The value has the format hh:mm:ss.sssssss.
extensionInfo	Specifies information about individual extensions that contributed to the execution time spent in the call stack up until the point at which the long-running threshold was exceeded and the trace was emitted. The following information is included for each extension: • `id` - the ID of the extension. • `extensionName` - the name of the extension. • `extensionVersion` - the version of the extension. • `extensionPublisher` - the publisher of the extension. • `subscriberExecutionCount` - the number of event subscribers executed in this extension. • `executionTime` - the total execution time for this extension in the call stack. The value has the format hh:mm:ss.sssssss.
extensionName	Specifies the name of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.
extensionPublisher	Specifies the publisher of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.

COLUMN (KEY)	DESCRIPTION OR VALUE
extensionVersion	Specifies the version of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.
extensionId	Specifies the ID of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.
longRunningThreshold	Specifies the time that defines a long-running AL method, after which the trace is emitted. The value has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Monitoring and Analyzing Long Running SQL Queries On-Premises
The Business Central Administration Center

Analyzing Long Running Operation (SQL Query) Telemetry

2/17/2021 • 4 minutes to read • Edit Online

A SQL query that takes longer than 1000 milliseconds to execute will be sent to your Application Insights resource.

To get a quick overview, you can go the Application Insights Overview dashboard.

> **NOTE**
> With Business Central On-premises, you can change the threshold that defines long running queries. For more information, see Defining Long Running SQL Queries Threshold.

There are multiple reasons that affect the time it takes SQL queries to run. For example, the database could be waiting for a lock to be released. Or, the database is executing an operation that does badly because of missing indexes. Sometimes you can look at the SQL statement that was generated by the code to see what caused the delay. This information is found in the **CustomDimension** data, specifically the **AL Stack Trace** column.

Dimensions in Application Insights

This table explains the columns included in long running query events emitted to Application Insights. Bold text indicates that the value of the columns is a constant. Some columns are standard for Application Insights. These columns are indicated by *Application Insights*.

COLUMN	DESCRIPTION OR VALUE
timestamp	Specifies the date and time that the long running query event occurred, such as 2019-08-20T07:23:07.9996696Z
message	Version 16.1 and later: **Operation exceeded time threshold (SQL query)** Before version 16.1: **Action took longer than expected**
severityLevel	**2** (This level indicates a warning. Long running queries are always recorded as warnings)
itemType	**trace**
customDimensions	(see table that follows)
operation_Name	**Long Running Operation (SQL Query)** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
operation_Id	Specifies the GUID assigned to the client operation. An operation is created whenever the user does something in the client, such as selecting an action.
operation_ParentId	Currently this column is the same as the operation_Id. This behavior might change in a future release.
session_Id	Specifies the GUID of the client session. When a client makes a connection to the Business Central Server instance, a session is created and assigned an ID.
client_Type	*Application Insights*
client_IP	*Application Insights*
client_City	*Application Insights*
client_StateOrProvince	*Application Insights*
client_CountryOrRegion	*Application Insights*
cloud_RoleName	Specifies the display name of Business Central tenant. For on-premises, this value is the same as the cloud_RoleInstance.
cloud_RoleInstance	Specifies the name of Business Central tenant.

COLUMN	DESCRIPTION OR VALUE
appId	*Application Insights*
appName	*Application Insights*
iKey	*Application Insights*
sdkVersion	*Application Insights*

CustomDimensions

This table describes the different dimensions of a **Long Running Operation (SQL Query)** operation.

COLUMN (KEY)	DESCRIPTION OR VALUE
extensionVersion	Specifies the version of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type of the Business Central solution, such as Production or Sandbox.
environmentName	Specifies the environment name of the Business Central solution, such as Production or Sandbox.
extensionName	Specifies the name of the extension.
alObjectType	The type of the AL object that executed the SQL statement
alObjectName	The name of the AL object that executed the SQL statement
alStackTrace	The stack trace in AL.
companyName	The display name of the Business Central company that was used at time of execution.
extensionId	Specifies the AppID of the extension.
eventId	**RT0005** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID when using Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as Background or Web. For a list of the client types, see ClientType Option Type.
alObjectId	The type of the AL object that executed the SQL statement.
component	Specifies the Business Central Server instance name and the platform version.
executionTime	Specifies the time that it took to execute the SQL statement**. The value has the format hh:mm:ss.sssssss.
longRunningThreshold	Specifies the amount of time that an SQL query can run before a warning event is recorded. The value has the format hh:mm:ss.sssssss. This threshold is controlled by the Business Central Server configuration setting called SqlLongRunningThreshold.
sqlStatement	Specifies the SQL statement that was executed for the long running query. The value is limited to 8192 characters. If the value exceeds 8192 characters, it will be truncated in manner that still provides the most pertinent information.
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.

** From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Example

The following code snippet shows an example of the CustomDimensions.


```
{"extensionVersion":"16.0.10962.0","telemetrySchemaVersion":"0.3","componentVersion":"15.0.40494.0","environmentType":"Production","environmentName":"Pr
Application","alObjectType":"Report","alObjectName":"Suggest Worksheet Lines","alStackTrace":"AppObjectType: Report\r\n AppObjectId: 840\r\n AL CallStacl
(Report 840).DeleteEntries line 10 - Base Application by Microsoft\r\n\"Suggest Worksheet Lines\"(Report 840).\"Cash Flow Forecast - OnPostDataItem\"(Tr
Application by Microsoft\r\n\"Cash Flow Management\"(CodeUnit 841).UpdateCashFlowForecast line 32 - Base Application by Microsoft\r\n\"Cash Flow Forecas
842).OnRun(Trigger) line 18 - Base Application by Microsoft\r\n\"Job Queue Start Codeunit\"(CodeUnit 449).OnRun(Trigger) line 11 - Base Application by M
Dispatcher\"(CodeUnit 448).HandleRequest line 30 - Base Application by Microsoft\r\n\"Job Queue Dispatcher\"(CodeUnit 448).OnRun(Trigger) line 19 - Base
Microsoft","companyName":"CRONUS USA, Inc.","extensionId":"437dbf0e-84ff-417a-965d-ed2bb9650972","aadTenantId":"8ca62103-8877-486d-88e2-
9a91303abfc6","clientType":"Background","alObjectId":"840","component":"Dynamics 365 Business Central Server","executionTime":"00:00:05.7470000","sqlSta
\"SQLDATABASE\".dbo.\"CURRENTCOMPANY$Cash Flow Forecast Entry$437dbf0e-84ff-417a-965d-ed2bb9650972\" WHERE (\"Cash Flow Forecast No_\"=@0)"}
```


See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Monitoring and Analyzing Long Running SQL Queries On-Premises
The Business Central Administration Center

Analyzing Page View Telemetry

2/17/2021 • 4 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.3

Page view telemetry gathers data about the pages that users open in the Business Central client. Each page view tells you how long it took to open the page, information about the user's environment, and more.

Use the data to gather statistics about system usage and also troubleshoot performance issues caused by the users' environments.

> **NOTE**
> In Application Insights, telemetry about page views is logged to the **pageViews** table and not the **traces** table like other Business Central traces. This also means that you can use the built-in pages in the **Usage** feature of the Application Insights to investigate how users interact with the Business Central environment. For more information, see Usage analysis with Application Insights.

Page opened: {alObjectName}

Occurs when a page has been opened in the client.

General dimensions

The pageViews table is a built-in table in Application Insights. Here are some of the fields most used in analysis of the signal:

FIELD	DESCRIPTION OR VALUE
client_Browser	Browser running on the client device.
client_OS	Operating system of the client device.
client_Type	Type of the client device.
duration	Number of milliseconds it took the application to handle the page view.
url	URL of the page view.
name	Name of the page opened in the client
session_id	An identifier for the session. Can be used to create a timeline of page views happening in the session

All fields are documented here: Application Insights PageViews Schema

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the page object that was opened.
alObjectName	Specifies the name of the page object that was opened.
alObjectType	**Page**
clientType	Specifies the type of client that opened the page such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	The display name of the Business Central company that was used when the page opened.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
designerLevel	Specifies the design level in which the page was opened. This dimension provides additional insight when the `hostType` dimension is **Designer**. • **None** The page wasn't opened in a design mode. This value is shown when the `hostType` dimension is a value other than **Designer** • **Personalization** The page opened in the personalizing mode for tailoring the page in the user's workspace only. See Personalize Your Workspace. • **Configuration** The page opened in customizing mode for tailoring the page for all users of a specific profile. See Customizing Pages for Profiles. • **Development** The page opened in design mode for developing and modifying the page from the client for all users. See Using Designer. • **Inspector** The page opened in page inspection mode for viewing page information like source table, source extension, and filters. See Inspecting Pages. • **All** - the page was opened in the personalization, configuration, development, and inspector modes.

DIMENSION	DESCRIPTION OR VALUE
deviceLocale	Specifies the preferred language that's configured for the device that opened the page. For example, this dimension could show the language setting of a browser used to view the page. The value is a language code, such as **en-US** or **da-DK**.
deviceScreenResolution	Specifies the display resolution of the device that opened the page. The value is given as {width}×{height}, with the units in pixels. For example, **1024×768** means the width is 1024 pixels and the height is 768 pixels.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventID	**CL0001** (note that this is different from other signals where the dimension is called *eventId*)
expandedFastTabs	Specifies the FastTabs on the page that were expanded when the page opened.
factboxExpanded	Specifies whether the FactBox area was shown or hidden when the page was opened. **true** indicates that the FactBox was shown. **false** indicates that the FactBox was hidden.
hostType	Specifies the host that loads the page. • **DeviceApp** The page was loaded by a Business Central mobile app, from a desktop, tablet, or phone. • **Officeadd-in** The page was loaded by Office application, like Excel or Outlook. • **Designer** The page was loaded by the in-client designer, used for changing page layout. The designer has different modes. Each mode provides a different level of page customization, as specified in the `designerLevel` dimension. • **ShimBrowser** The page was loaded by the UWP (Universal Windows Platform) application. • **Browser** The page was loaded by web browser, like Microsoft Edge or Google Chrome.
message	**Page opened: {alObjectName}**
pageIsModal	Specifies whether the page was opened as a modal page. **true** indicates that the page was opened as a modal page; otherwise, **false**.

DIMENSION	DESCRIPTION OR VALUE
pageMode	Specifies whether the page was opened in one of the following modes: • **View** indicates the page was opened for viewing only • **Edit** indicates the page was opened for making changes • **Create** indicates the page was opened for creating a new entity. • **Select** indicates the page was opened for selecting an existing entity. • **Delete** indicates the page was opened for deleting an entity. The mode in which a page opened determined by different things, like the RunPageMode Property or the URL used to open the page.
pageType	Specifies the type of page, such as **Card**, **List**, **Document**, and more. For a complete list of page types and descriptions, see Page Types and Layouts.
refUri	Specifies the URI of the page.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Permission Changes Trace Telemetry

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, version 17.2, and later

Permission changes telemetry gathers data about the following operations on permission sets:

- A user-defined permission set was added or removed
- A link between a user-defined permission set and system permission set was added or removed
- A permission set was assigned to or removed from a user or user group

For information about managing permission sets, see Assign Permissions to Users and Groups.

User-defined permission set added

Occurs when a user-defined permission set is created.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**User-defined permission set added: {alPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E2A**
alNumberOfUserDefinedPermissionSets	Specifies the total number of user-defined permission sets.
alPermissionSetId	Specifies the ID assigned to the permission set.

Common custom dimensions

The following table explains additional custom dimensions that are common to all permission traces.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**AL PermissionSet**
alDataClassification	**SystemMetadata**
alObjectId	**1351**, which is the ID of the system application codeunit that subscribes to the telemetry events.

DIMENSION	DESCRIPTION OR VALUE
alObjectName	**Telemetry Subscribers**, which is the name of the system application codeunit that subscribes to the telemetry events.
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

User-defined permission set removed

Occurs when a user-defined permission set is deleted.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**User-defined permission set removed: {alPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E2B**
alNumberOfUserDefinedPermissionSets	Specifies the total number of user-defined permission sets.
alPermissionSetId	Specifies the ID of the permission set that was deleted.
See common custom dimensions	

Permission set link added

Occurs when a user-defined permission set is created from a copy of a system permission set, and the **Notify on Changed Permission Set** option is selected. The **Notify on Changed Permission Set** option creates a link between the system permission set and the user-defined permission set.

General dimensions

DIMENSION	DESCRIPTION OR VALUE

DIMENSION	DESCRIPTION OR VALUE
message	**Permission set link added: {alSourcePermissionSetId} -> {alLinkedPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E28**
alSourcePermissionSetId	Specifies the ID of the system permission set that was copied to create the user-defined permission set.
alLinkedPermissionSetId	Specifies the ID of the user-defined permission that was created from a copy of the system permission set.
alNumberOfUserDefinedPermissionSetLinks	Specifies the total number of user-defined permission sets that are linked to system permission sets.
See common custom dimensions	

Permission set link removed

Occurs when a user-defined permission set, which is linked to a system permission set, is deleted.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Permission set link removed {alSourcePermissionSetId} -> {alLinkedPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E29**
alSourcePermissionSetId	Specifies the ID of the system permission set that the deleted user-defined permission set was linked to.
alLinkedPermissionSetId	Specifies the ID of the deleted user-defined permission set.
alNumberOfUserDefinedPermissionSetLinks	Specifies the total number of user-defined permission sets that are linked to system permission sets.
See common custom dimensions	

Permission set assigned to user

Occurs when a permission set is assigned to a user.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Permission set assigned to user: {alPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E2C**
alPermissionSetId	Specifies the ID of the permission set that was assigned to a user.
See common custom dimensions	

Permission set removed from user

Occurs when a permission set is removed from a user.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Permission set removed from user: {alPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E2D**
alPermissionSetId	Specifies the ID of the permission set that was removed from the user.
See common custom dimensions	

Permission set assigned to user group

Occurs when a permission set is assigned to a user group.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Permission set assigned to user group: {alPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E2E**
alPermissionSetId	Specifies the ID of the permission set that was assigned to the user group.
alUserGroupId	Specifies the ID of the user group that the permission set was assigned to.
See common custom dimensions	

Permission set removed from user group

Occurs when a permission set is removed from a user.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Permission set removed from user group: {alPermissionSetId}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E2F**
alPermissionSetId	Specifies the ID of the permission set that was removed from the user group.
alUserGroupId	Specifies the ID of the user group that the permission set was removed from.
See common custom dimensions	

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Report Generation Telemetry

2/17/2021 • 10 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Report generation telemetry gathers data about reports that are run on the service. It provides information about whether the report dataset generation succeeded, failed, or was canceled. For each report, it tells you how long it ran, how many SQL statements it executed, and how many rows it consumed.

You use this data to gather statistics to help identify slow-running reports.

Success report generation

Occurs when a report dataset generates without any errors.

General dimensions

The following table explains the general dimensions of this trace.

DIMENSION	DESCRIPTION OR VALUE
operation_Name	**Success report generation** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
message	Version 16.1 and later: **Report rendered: {report ID} - {report name}** Before version 16.1: **The report {report ID} '{report name}' rendered successfully**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.

DIMENSION	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0006** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
numberOfRows	Specifies the number of rows/records generated for the report dataset.
reportingEngine	Specifies the reporting engine used to generate the report, such as **ProcessingOnly**, **Rdlc**, or **Word**. This dimension was added in version 17.3
result	**Success**
serverExecutionTime	Specifies the amount of time it took the service to complete the request[1]. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the report executed[1].

DIMENSION	DESCRIPTION OR VALUE
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements[1].
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took for the system to generate the dataset and render the report[1]. The time has the format hh:mm:ss.sssssss.

From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Failed report generation

This operation occurs when the report dataset couldn't be generated because of an error.

General dimensions

The following table explains the general dimensions of the **Failed report generation** operation.

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later: **Report rendering failed: {report ID} - {report name}** Before version 16.1: **The report {report ID} '{report name}' couldn't be rendered**
operation_Name	**Failed report generation** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.

DIMENSION	DESCRIPTION OR VALUE
cancelReason[2]	Specifies why the report was canceled.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0006** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
numberOfRows	Specifies the number of rows/records generated for the report dataset.
result	Specifies the title of the exception that was thrown, such as **NavNCLDialogException**.
serverExecutionTime	Specifies the amount of time used by service on the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the report executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.

DIMENSION	DESCRIPTION OR VALUE
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to generate the dataset and render the report before it failed. The time has the format hh:mm:ss.sssssss.

[2] Available in Business Central 2020 release wave 2 and later only.

Analyzing report generation failures

When a report fails to generate, the `result` column in the CustomDimensions will include the title of the exception that was thrown by the service or the AL code.

Cancellation report generation

This operation occurs when the report dataset generation was canceled. There are various conditions that can cancel a report. The **Cancellation report generation** operation emits different trace messages for each condition.

General dimensions

The following table explains the general dimensions of the **Cancellation report generation** operation.

DIMENSION	DESCRIPTION OR VALUE
operation_Name	**Cancellation report generation** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
message	Specifies the reason why the report was canceled. See Analyzing cancellation messages section for details.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.

DIMENSION	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0007** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Analyzing report cancellations

The cancellation messages indicate events that caused the report to be canceled. The telemetry can help identify slow-running reports - reports that take longer than expected to run and generate a large number of rows.

> **NOTE**
> The service evaluates cancellation events in a specific order, and the evaluation is done every five seconds. For more information, see Report Generation and Cancellation Flow.

Cancellation event received. Requesting cancellation of the action.

This message occurs when the session canceled a report as it was being generated.

Received a cancellation request from the user. Requesting cancellation of the action.

This message occurs when a user canceled a report in the client as it was being generated.

The action took longer to complete ({0}) than the specified threshold ({1}). Requesting cancellation of the action.

The service is configured to cancel reports if they take longer to generate than a set amount of time. With Business Central online, you can't change the threshold. With Business Central on-premises, you change the threshold by setting the **Max Execution Timeout** parameter on the Business Central Server instance. There's no timeout for on-premises by default. For more information, see Configuring Business Central Server.

The rendering of the word report has been cancelled because it took longer than the specified threshold ({0})"

This message occurs when a report that based on a Word layout takes longer to generate than the specified threshold. The event is only relevant for Business Central online. There's no timeout for on-premises.

The number of processed rows exceeded ({0} rows) the maximum number of rows ({1} rows). Requesting cancellation of the action.

The service is configured to cancel reports if they generate more than a set number of rows. With Business Central online, you can't change this threshold. With Business Central on-premises, you change the threshold by setting the **Max Rows** parameter on the Business Central Server instance. There's no limit on rows for on-premises by default. For more information, see Configuring Business Central Server.

Report cancelled but a commit occurred

This operation occurs when the report dataset generation was canceled but a COMMIT operation occurred before the cancellation. This pattern isn't recommended. Reconsider the report design.

General dimensions

The following table explains the general dimensions of the **Report cancelled but a commit occurred** operation.

DIMENSION	DESCRIPTION OR VALUE
message	The report <ID> '<Name>' is being canceled, but a COMMIT() has been performed. This can lead to data inconsistency if the report is not idempotent
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.
cancelReason	The reason why the report was cancelled
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.

DIMENSION	DESCRIPTION OR VALUE
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0011** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionPublisher	Specifies the name of the extension publisher that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Incoming Web Services Request Telemetry

2/17/2021 • 5 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Web services telemetry gathers data about SOAP, OData, and API requests through the service. It provides information like the request's endpoint, time to complete, the SQL statements run, and more.

As a developer, you use the data to learn about conditions that you can change to improve performance. The following table provides some examples:

CONDITION	ANALYSIS
A web service request results in a long running SQL query	Adjust or fine-tune code.
Web service requests to a specific endpoint read more rows than requests to the other endpoints	Consider adding filtering to limit the rows that are read.
Fewer API type requests compared with other types	With SOAP and OData requests, computation resources are used on UI elements that aren't relevant. Instead of exposing normal pages as web service endpoints, use the built-in API pages. API pages are optimized for this scenario.
High number of requests to endpoints that include Power BI	This condition may indicate excessive Power BI integration.

For more performance guidelines, see Writing efficient Web Services.

> **NOTE**
> Business Central online and on-premises are configured with various limits on web service requests. For example, there is a request timeout and a maximum connections limit. For online, you can't change these limits, but it is helpful to know what the limits are. See Current API Limits. For on-premises, you change the limits on the Business Central Server instance. See Configuring Business Central Server. Web service calls that exceed the timeout limit result in a **408 - Request Timeout**. These calls are recorded in Application Insights with a totalTime that is equal to the timeout threshold.

General dimensions

The following table explains the general dimensions included in an incoming **Web Services Call** trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
operation_Name	**Web Services Call** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later (depending on the type): • **Web service called (API): {endpoint}** • **Web service called (ODataV4): {endpoint}** • **Web service called (ODataV3): {endpoint}** • **Web service called (SOAP): {endpoint}** Before version 16.1: • **Received a web service request of type API** • **Received a web service request of type ODataV4** • **Received a web service request of type ODataV3** • **Received a web service request of type SOAP**
severityLevel	**1**

Custom dimensions

The following table explains the custom dimensions included in a **Web Services Call** trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.[1]
alObjectName	Specifies the name of the AL object that was run by the request.[1]
alObjectType	Specifies the type of the AL object that was run by the request.[1]
category	Specifies the service type. Values include: **API**, **ODataV4**, **ODataV3**, and **SOAP**.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0008** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
httpHeaders	Introduced in version 16.3. Specifies the http headers set in the request.

DIMENSION	DESCRIPTION OR VALUE
httpMethod	Introduced in version 16.3. Specifies the HTTP method used in the request. Values include: POST, GET, PUT, PATCH, orDELETE.
httpStatusCode	Introduced in version 16.3. Specifies the http status code returned when a request has completed. This dimension further indicates whether request succeeded or not, and why. Use it to verify whether there was an issue with a request even though the request was logged as successful. The dimension displays one of the following values: • **200** OK. The request succeeded. • **401** Access denied. The user who made the request doesn't have proper permissions. For more information, see Web Services Authentication and Assign Permissions to Users and Groups. • **404** Not found. The given endpoint was not valid. For more information, see Publishing a Web Service • **408** Request timed out. The request took longer to complete than the threshold configured for the service. For information about this threshold in Business Central online, see OData request limits. For on-premises, the timeout is determined by the ODataServicesOperationTimeout setting of the Business Central Server. For more information, see Configuring Business Central Server • **429** Too Many Requests. The request exceeded the maximum simultaneous requests allowed on the service. For information about this threshold in Business Central online, see OData request limits. For on-premises, the timeout is determined by the ODataMaxConnections setting of the Business Central Server. For more information, see Configuring Business Central Server This dimension was introduced in Business Central 2020 release wave 1, version 16.3. This dimension is not populated for SOAP endpoints.
serverExecutionTime	Specifies the amount of time it took the server to complete the request**. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.[1] [2]
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.[1] [2]
totalTime	Specifies the amount of time it took to process the request.[2] The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

[1]This dimension isn't relevant for $metadata calls, like `https://localhost:7048/BC/ODataV4/$metadata`, so it won't be in the trace.

[2]From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Example trace

The following code snippet is a CustomDimensions example:

{"telemetrySchemaVersion":"0.6","componentVersion":"16.0.11329.0","environmentType":"Production","deprecatedKeys":"Company name, AL C
AL Object type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension version, Telemetry schema ve
AadTenantId, Environment name, Environment type, Component, Component version, Telemetry schema
version","serverExecutionTime":"00:00:00.3886441","component":"Dynamics 365 Business Central
Server","aadTenantId":"common","sqlExecutes":"21","sqlRowsRead":"117","totalTime":"00:00:00.3886441","alObjectType":"Page","alObjectN
Document Line
Entity","alObjectId":"6403","category":"ODataV4","endpoint":"BC/ODataV4/Company()/workflowSalesDocumentLines","httpStatusCode":"200"}

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Outgoing Web Service Request Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Outgoing web service request telemetry gathers data about outgoing web service requests sent using the AL HTTPClient module. As a partner, the data gives you insight into the execution time and failures that happen in external services that your environment and extensions depend on. Use the data to monitor environments for performance issues caused by external services, and be more proactive in preventing issues from occurring.

General dimensions

The following table explains the general dimensions included in an outgoing **Web Services Call (Outgoing)** trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Web Service Called (Outgoing): {endpoint}**
severityLevel	**1**

Custom dimensions

The following table explains the custom dimensions included in a **Web Services Call** trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alAuthenticationMethod	Specifies the user authentication used by the Business Central service . Values include: Windows, UserName, NavUserPassword, AccessControlService. For more information about the authentication types, see Authentication and Credential Types.
alHttpTimeout	Specifies the timeout defined for the request. The timeout is the time to wait before a request gets canceled. The value has the format hh:mm:ss. The timeout is defined either by the NavHttpClientMaxTimeout setting on the Business Central Server instance or by a TimeOut method call in extension code. The TimeOut method call takes precedence.
alObjectId	Specifies the ID of the AL object that made the request.
alObjectName	Specifies the name of the AL object that made the request.
alObjectType	Specifies the type of the AL object that made the request.
companyName	Specifies the display name of the Business Central company from which the request was made.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request. The endpoint is cleaned to include only the base URI.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0019**
extensionId	Specifies the appID of the extension that made the request.
extensionName	Specifies the name of the extension that made the request.
extensionVersion	Specifies the version of the extension that made the request.

DIMENSION	DESCRIPTION OR VALUE
httpHeaders	Introduced in version 17.2. Specifies the http headers set in the request.
httpMethod	Specifies the HTTP method used in the outgoing request. Values include: POST, GET, PUT, PATCH, orDELETE.
httpReturnCode	**Deprecated in version 17.2. Use the dimension httpStatusCode instead.** Specifies the http status code returned when a request has completed. This dimension further indicates whether request succeeded or not, and why. Use it to verify whether there was an issue with a request even though the request was logged as successful. The dimension displays one of the following values: • **200** OK. The request succeeded. • **404** Not found. The given endpoint wasn't valid.
httpStatusCode	Specifies the http status code returned when a request has completed. This dimension further indicates whether request succeeded or not, and why. Use it to verify whether there was an issue with a request even though the request was logged as successful. The dimension displays one of the following values: • **200** OK. The request succeeded. • **404** Not found. The given endpoint wasn't valid.
serverExecutionTime	Specifies the amount of time it took the server to complete the request, including the time to open the company. The time has the format hh:mm:ss.sssssss.
totalTime	Specifies the amount of time it took to process the request, including the time to open the company. The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Example trace

The following code snippet is a CustomDimensions example:


```
{"Telemetry schema version":"0.2","telemetrySchemaVersion":"0.2","Component version":"17.0.15765.0","Environment type":"Production","componentVersion":"17.0.15
AL Object Id, AL Object type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension version, Telemetry schema version, Compo
version":"17.0.15821.0","extensionVersion":"17.0.15821.0","Extension App Id":"11111111-aaaa-2222-bbbb-333333333333","aadTenantId":"afbb64dc-bc88-43a3-9153-dd9d
type":"CodeUnit","Extension name":"Base Application","AL Object name":"My Codeunit","component":"Dynamics 365 Business Central Server","companyName":"CRONUS US
Codeunit","alObjectType":"CodeUnit","extensionId":"11111111-aaaa-2222-bbbb-
333333333333","eventId":"RT0019","httpMethod":"GET","serverExecutionTime":"00:00:00.1971767","totalTime":"00:00:00.1971767","endpoint":"https://mycorp.dynamics
```


See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Web Service Access Key Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, version 17.3, and later

The Business Central emits telemetry data about the success or failure of authenticating web service access keys on web service requests.

In a future release, web service access key feature will be deprecated. As a partner or customer, this data lets you monitor the use of web service access keys on your environments in preparation for this change.

For information about web service access keys, see How to use an Access Key for SOAP and OData Web Service Authentication.

Authentication with web service key succeeded

Occurs when a web service access key was authenticated successfully.

General dimensions

The following table explains the general dimensions included in the trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Authentication with Web Service Key succeeded: {endpoint}**
severityLevel	**1**

Custom dimensions

The following table explains the custom dimensions included in the trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.
authenticationStatus	**Succeeded**
authenticationType	Specifies whether the service uses NavUserPassword or AccessControl (Azure AD) authentication.
category	Specifies the service type. Values include: **Api**, **ODataV4**, **ODataV3**, and **SOAP**.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request. Private and sensitive information is omitted from the endpoint.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0020** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Authentication with web service key failed

Occurs when a web service access key failed to authenticate.

General dimensions

The following table explains the general dimensions included in the trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Authentication with Web Service Key failed: {endpoint}**
severityLevel	**3**

Custom dimensions

The following table explains the custom dimensions included in the trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.
authenticationStatus	**Failed**
authenticationType	Specifies whether the service uses NavUserPassword or AccessControl (Azure AD) authentication.
category	Specifies the service type. Values include: **Api**, **ODataV4**, **ODataV3**, and **SOAP**.
component	**Dynamics 365 Business Central Server**

DIMENSION	DESCRIPTION OR VALUE
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request. Private and sensitive information is omitted from the endpoint.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0021**
failureReason	Specifies the exception that was thrown by the server. Some exception messages can contain customer data. As a precaution, Business Central only emits information that's classified as SystemMetadata. Exception messages that contain other data classifications, like customer data, are not shown. Instead, the following message is shown: "Message not shown because the NavBaseException(string, Exception, bool) constructor was used."
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Example trace

The following code snippet is a CustomDimensions example:

```
{"component":"Dynamics 365 Business Central
Server","category":"Api","AadTenantId":"common","environmentType":"Production","Component":"Dynamics 365 Business Central
Server","AL Object Id":"0","alObjectId":"0","Telemetry schema
version":"0.1","telemetrySchemaVersion":"0.1","authenticationType":"NavUserPassword","componentVersion":"17.0.15855.0","Compo
version":"17.0.15855.0","eventId":"RT0020","endpoint":"BC170/ODataV4/Company()/purchaseDocumentLines","deprecatedKeys":"Compa
name, AL Object Id, AL Object type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension
version, Telemetry schema version, Component, Component version, Telemetry schema version, AadTenantId, Environment name,
Environment type","aadTenantId":"common","Environment type":"Production","authenticationStatus":"Succeeded"}
```

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Telemetry Event IDs in Application Insights

2/17/2021 • 3 minutes to read • Edit Online

The following tables list the IDs of Business Central telemetry trace events that can be emitted in Azure Application Insights.

Application events

EVENT ID	AREA	MESSAGE
AL0000CTV	Email	Email sent successfully
AL0000CTE	Field monitoring	Sensitive field value has changed: {alfieldCaption} in table {altableCaption}
AL0000CTP	Email	Failed to send email
AL0000DD3	Field monitoring	Sensitive field monitor status has changed to {almonitorStatus}
AL0000E2A	Permissions	User-defined permission set added: {alPermissionSetId}
AL0000E2B	Permissions	User-defined permission set removed: {alPermissionSetId}
AL0000E28	Permissions	Permission set link added: {alSourcePermissionSetId} -> {alLinkedPermissionSetId}
AL0000E29	Permissions	Permission set link removed: {alSourcePermissionSetId} -> {alLinkedPermissionSetId}
AL0000E2C	Permissions	Permission set assigned to user: {alPermissionSetId}
AL0000E2D	Permissions	Permission set removed from user: {alPermissionSetId}
AL0000E2E	Permissions	Permission set assigned to user group: {alPermissionSetId}
AL0000E2F	Permissions	Permission set removed from user group: {alPermissionSetId}

Client events

EVENT ID	AREA	MESSAGE
CL0001	Page views	Page opened: {alObjectName}

Lifecycle events

EVENT ID	AREA	MESSAGE
AL0000E24	Job Queue Lifecycle	Job queue entry enqueued: {alJobQueueId}
AL0000E25	Job Queue Lifecycle	Job queue entry started: {alJobQueueId}
AL0000E26	Job Queue Lifecycle	Job queue entry finished: {alJobQueueId}
AL0000E3F	Configuration Package	Configuration package export started: {alPackageCode}
AL0000E3G	Configuration Package	Configuration package exported successfully: {alPackageCode}
AL0000E3H	Configuration Package	Configuration package import started: {alPackageCode}
AL0000E3I	Configuration Package	Configuration package imported successfully: {alPackageCode}
AL0000E3N	Configuration Package	Configuration package apply started: {alPackageCode}
AL0000E3O	Configuration Package	Configuration package applied successfully: {alPackageCode}
AL0000E3P	Configuration Package	Configuration package deleted successfully: {alPackageCode}
LC0001	Company Lifecycle	Company created: {companyName}
LC0002	Company Lifecycle	Company creation canceled: {companyName}
LC0003	Company Lifecycle	Company creation failed: {companyName}
LC0004	Company Lifecycle	Company copied: {companyNameSource} to {companyNameDestination}
LC0005	Company Lifecycle	Company copied canceled: {companyNameSource} to {companyNameDestination}

EVENT ID	AREA	MESSAGE
LC0006	Company Lifecycle	Company copy failed: {companyNameSource} to {companyNameDestination}
LC0007	Company Lifecycle	Company deleted: {companyName}
LC0008	Company Lifecycle	Company deletion canceled: {companyName}
LC0009	Company Lifecycle	Company deletion failed: {companyName}
LC0010	Extension Lifecycle	Extension installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0011	Extension Lifecycle	Extension failed to install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0012	Extension Lifecycle	Extension synchronized successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId}
LC0013	Extension Lifecycle	Extension failed to synchronize: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0014	Extension Lifecycle	Extension published successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0015	Extension Lifecycle	Extension failed to publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0016	Extension Lifecycle	Extension un-installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0017	Extension Lifecycle	Extension failed to un-install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0018	Extension Lifecycle	Extension unpublished successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})

EVENT ID	AREA	MESSAGE
LC0019	Extension Lifecycle	Extension failed to un-publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0020	Extension Lifecycle	Extension compiled successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0021	Extension Lifecycle	Extension failed to compile: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0022	Extension Lifecycle	Extension updated successfully: {extensionName} version {extensionVersion} by {extensionPublisher}
LC0023	Extension Lifecycle	Extension failed to update: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})

Runtime events

EVENT ID	AREA	MESSAGE
RT0001	Authorization	AuthorizationFailed(PreOpenCompany) : {failure reason}
RT0002	Authorization	Authorization Failed (Open Company): {failure reason}
RT0003	Authorization	AuthorizationSucceeded(PreOpenCompany)
RT0004	Authorization	Authorization Succeeded (Open Company)
RT0005	Performance	Operation exceeded time threshold (SQL query)
RT0006	Report generation	Success report generation
RT0006	Report generation	Report rendering failed: {report ID} - {report name}
RT0007	Report generation	Cancellation report generation
RT0008	Incoming Web service requests	Web service called ({category of request}): {endpoint}

EVENT ID	AREA	MESSAGE
RT0010	Extension lifecycle	Extension Update Failed: exception raised in extension {extensionName} by {extensionPublisher} (updating to version {extensionTargetedVersion})
RT0011	Report generation	Report cancelled but a commit occurred
RT0012	Performance	Database lock timed out
RT0013	Performance	Database lock snapshot: {snapshotId}
RT0014	Security	App Key Vault initialization succeeded: '{keyVaultUri}'
RT0015	Security	App Key Vault initialization failed
RT0016	Security	App Key Vault secret retrieval succeeded from key vault '{keyVaultUri}'
RT0017	Security	App Key Vault secret retrieval failed from key vault: '{keyVaultUri}'
RT0018	Performance	Operation exceeded time threshold (AL method)
RT0019	Outgoing Web service requests	Web Service Called (Outgoing): {endpoint}
RT0020	Web service key request	Authentication with Web Service Key succeeded: {endpoint}
RT0021	Web service key request	Authentication with Web Service Key failed: {endpoint}

See also

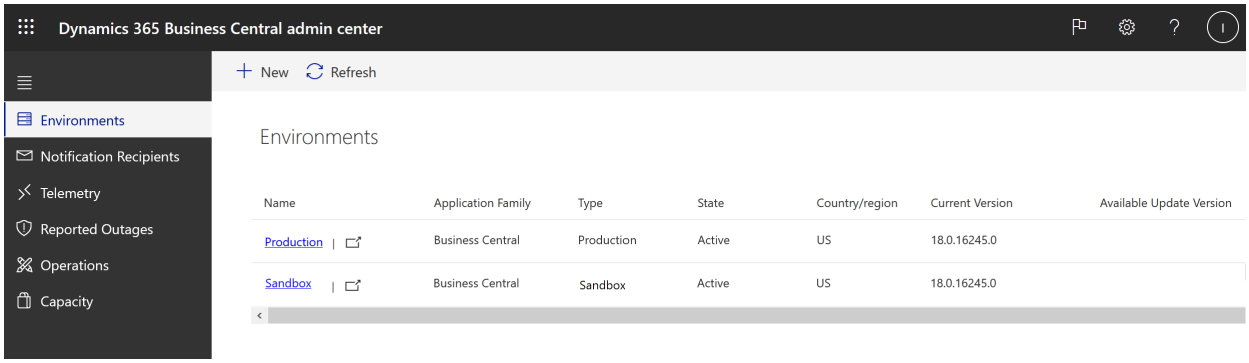
Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

The Business Central Administration Center

2/17/2021 • 3 minutes to read • Edit Online

The Business Central administration center provides a portal for administrators to do administrative tasks for a Business Central tenant. Here, administrators can:

- view and work with production and sandbox environments for the tenant.
- set up upgrade notifications.
- view telemetry for events on the tenant.



Access to the administration center

The following users are authorized to access the Business Central administration center:

- Internal tenant administrators
- Admin agent
- Helpdesk agent

Internal administrators are users who are assigned the **Global admin** role or the **Dynamics 365 Admin** role in the Microsoft 365 admin center. These users are typically system administrators, IT professionals, or super users at the customer's company. For more information, see About admin roles in the Microsoft 365 admin content.

The *admin agent* and *helpdesk agent* roles are assigned through the Microsoft Partner Center for the partner that is associated with the tenant. These roles can access the Business Central tenant as *delegated administrators*. For more information, see Administration of Business Central Online.

Internal administrators

As the internal administrator, you can choose the link in the **Settings** menu when you're signed in to Business Central.

Alternatively, you can access the administration center from the URL, use the following pattern but replace *[TENANT_ID]* with the tenant ID of your Business Central:

```
https://businesscentral.dynamics.com/[TENANT_ID]/admin
```

TIP

The tenant ID is shown in the **Help and Support** page in your Business Central.

In the administration center, you can create and monitor environments. This is also where you manage the people who must be notified of administrative events for your tenant.

Your partner can help you set up telemetry for production environments, including integration with Application Insights in Azure.

Cleaning up settings

If your organization decides to switch to another partner, you must make sure that some settings that your current partner made in your Business Central administration center are removed. This includes the following settings:

- Support contact details
  1. In the Business Central administration center, choose the relevant environment, and then, in the **Support** menu, choose **Manage Support Contact**.
  2. Verify that the values in the **Name**, **Email address**, and the **Website** fields are still relevant; if not, then delete or modify the values.
- Notification recipients
  1. In the Business Central administration center, on the left side, choose **Notification recipients**
  2. Verify that the list of email addresses are still relevant; if not, then delete or modify the values.
- Application Insights key (if this was set up by the partner)
  1. In the Business Central administration center, choose the relevant environment, and then, in the top menu, choose **Application Insights Key**.
  2. Remove the value of the **Instrumentation Key**

When you establish a relationship with a new partner, they will fill in these fields again.

Partner access to the administration center

As a partner, you can access the administration center from the Partner Dashboard in the Microsoft Partner Center:

1. Log into the Partner Dashboard.
2. Select the **Customers** link in the navigation pane.
3. Select the customer tenant that you want to do administrative tasks for.
4. Select **Service Management**.
5. Under the **Administer Services** heading, select Dynamics 365 Business Central.

You can also get to the administration center by using the URL of a tenant, as described in the previous section.

In the Business Central administration center, you can specify support information, create and remove environments, and submit support request for your customer.

From the Business Central administration center, you can access your customer's Business Central for troubleshooting, for example.

> **NOTE**
> As the partner, there are certain tasks that you cannot do in your customers' Business Central. For more information, see Acting as a delegated administrator.

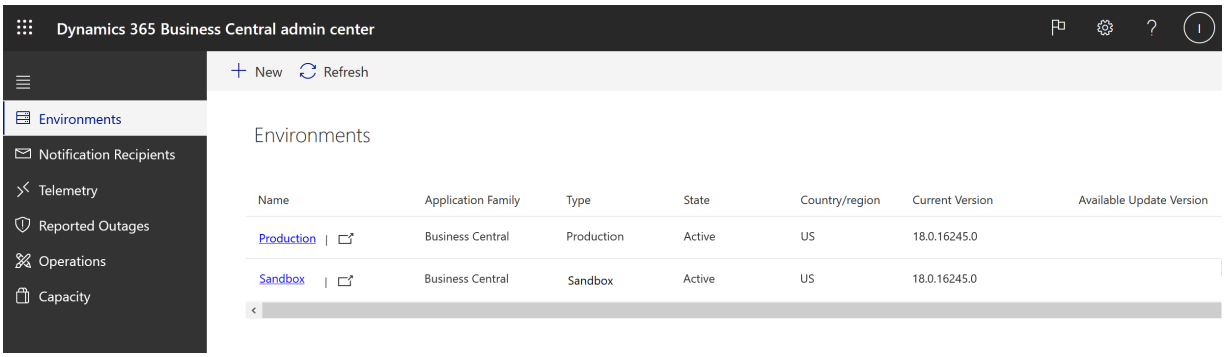
See also

Production and Sandbox Environments
Managing Environments
Tenant Notifications
Environment Telemetry
Administration Center API
Managing Technical Support
Business Central Data Security
Introduction to automation APIs
Microsoft Partner Dashboard
Add a new customer in the Partner Center
Assign licenses to users in the Partner Center
Create new subscriptions in the Partner Center
Cloud Solution Provider program - selling in-demand cloud solutions

Managing Environments

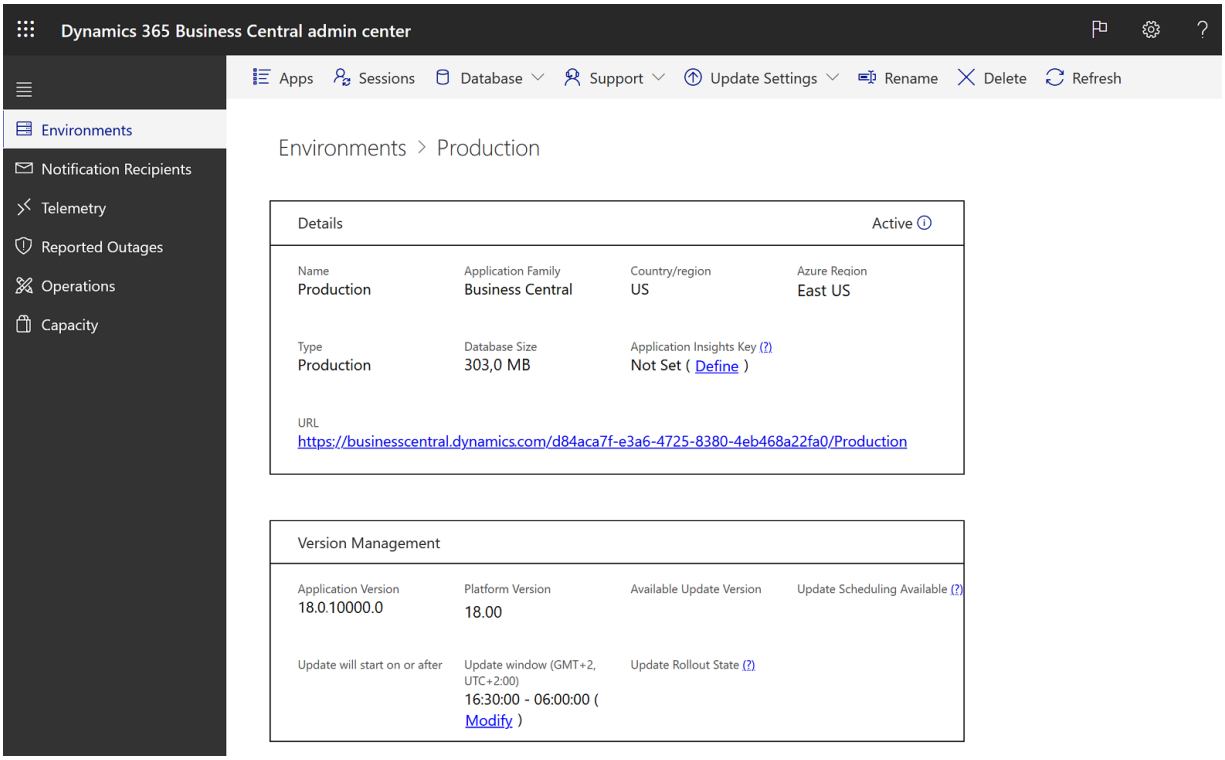
2/17/2021 • 21 minutes to read • Edit Online

The **Environments** tab of the Business Central administration center provides you with an overview of the Business Central production and sandbox environments for the tenant, and you can manage updates for each environment.



Viewing details for an environment

In the environments list, you can view more details by choosing the link in the **Name** column.



Types of environments

You can create environments of different types. Which type of environment to choose depends on what you need it for. For more information, see Production and Sandbox Environments.

Production environments

Production environments are meant to be precisely that: Environments that a business can run their daily business in Business Central in, deployed on performance tiers in Azure with a guaranteed high level of availability and support.

Production environments are backed up automatically and frequently to help protect business data. For more information, see How often are production databases backed up?.

You can create additional production environments for training or performance testing, for example. However, for training purposes, in many cases organizations will prefer to create a sandbox environment with production data. You can also create additional production environments to support offices in different countries.

> **NOTE**
> The Premium and Essential subscription types give each Business Central customer one production environment and three sandbox environment free of extra charge. If the customer requires more production environments, they can buy additional environments through their CSP partner. Each additional production environment comes with three additional sandbox environments and 4 GB additional, tenant-wide database capacity.

Administrators can create the additional environments in the Business Central administration center. The environments can be created and used in any country or region where Business Central is available, including the countries or regions where the customer's existing environments are running. The environments quota is applied when you try to create a new environment, or copy an existing environment, in the Business Central administration center.

Sandbox environments

Sandbox environments are meant to be precisely that: Environments that you can play around with, use as a testbed for development, and delete at will. You can deploy apps straight from Visual Studio Code to a sandbox environment, and you can attach a debugging session to a sandbox.

> **IMPORTANT**
> Apps that are published to a sandbox from the development environment or created using Designer are published within the scope of the service node that hosts the environment. When the sandbox is upgraded, these apps are removed because the environment is moved to another node that is running the new version. However, the data of the app is not removed, so you only have to re-publish and install the app to make it available.
> Apps that are uploaded to the environments of both types (production and sandbox) using the **Upload Extension** action from the **Extension Management** page are published within a global scope. When the environment is upgraded or moved, these apps are downloaded to the service node and installed, which means that they will not disappear.

You can also safely use sandboxes for training, such as for following a learning path from Microsoft Learn, because it's a safe environment to experiment with. If anything goes wrong, you just delete the sandbox and start over.

> **IMPORTANT**
> The automatic backup that applies to production environments does not apply to sandbox environments. If you want to export data from a sandbox environment, you can use Excel or RapidStart, but you cannot request a database export.

You can create a sandbox environment that includes data from your production environment for debugging purposes, for example. But if you want to run performance tests, or similar benchmarking, the sandbox is not reliable enough for that purpose. This is because sandboxes run in a different performance tier on Azure than production environments. Instead, create a dedicated environment based on the Production environment type - this gives you the exact experience and performance that users will experience in the actual production environment.

Sandbox environments are handy for certain types of development scenarios because the debugging endpoint is open by default. This means that you can attach Visual Studio Code to a running system and debug through running code. It also allows you to publish directly to the environment from Code.

If your organization has more than one sandbox environment, you can switch between environments by opening the App Launcher, choosing the Dynamics 365 tile, then choose the Business Central Sandbox tile. The sandbox environment picker shows the available sandboxes, so choose the one that you want to switch to.

> **NOTE**
>
> The Premium and Essential subscription types give each Business Central customer one production environment and three sandbox environment free of extra charge. If the customer requires more production environments, they can buy additional environments through their CSP partner. Each additional production environment comes with three additional sandbox environments and 4 GB additional, tenant-wide database capacity.

Administrators can create the additional environments in the Business Central administration center. The environments can be created and used in any country or region where Business Central is available, including the countries or regions where the customer's existing environments are running. The environments quota is applied when you try to create a new environment, or copy an existing environment, in the Business Central administration center.

Pre-sales performance evaluation

If you want to provide a prospect with an online environment where you want to demonstrate the performance and reliability of Business Central online in addition to demonstrating functionality, you must take a few extra steps.

To demonstrate the functionality of the default version of Business Central, without focusing on performance, you can quite simply use your own trial experience based on a Microsoft 365 demo account. We recommend that you show the full functionality of the default version by switching the tenant to the 30 day trial and the associated My Company. You can then enable the *Premium* user experience in the new My Company's **Company Information** page, populate the new company with the data required for their evaluation scenarios, and present the environment to the prospect.

To demonstrate the functionality of the service, with a focus on performance, you can take the same step as outlined above, but then also sign up the prospect for the Business Central Premium Trial offer that is available through your CSP access in the Partner Center, wait 24 hours, and then run your performance evaluation. For more information, see Preparing Test Environments of Dynamics 365 Business Central.

That 30 day trial is as close to an actual production environment performance as you can get. You only have those 29 days to run your tests and convince the prospect, of course, but provided that you have prepared everything in advance, it should give you time enough.

If the prospect is convinced and decides to buy Business Central, you can then either let them keep the environment that they are currently using, or create a new production environment for them. If the tenant is yours rather than the prospect's, then a new tenant will be provided to them.

For more information about performance and Business Central, see Performance Overview.

Create a new production environment

The Business Central administration center provides an easy method for creating environments for the tenant. For example, if you have been using a production environment for training purposes, and you've decided to start using Business Central to run the business, you can delete the original production environment and then create a new production environment.

> **NOTE**
> The Premium and Essential subscription types give each Business Central customer one production environment and three sandbox environment free of extra charge. If the customer requires more production environments, they can buy additional environments through their CSP partner. Each additional production environment comes with three additional sandbox environments and 4 GB additional, tenant-wide database capacity.

Administrators can create the additional environments in the Business Central administration center. The environments can be created and used in any country or region where Business Central is available, including the countries or regions where the customer's existing environments are running. The environments quota is applied when you try to create a new environment, or copy an existing environment, in the Business Central administration center.

To create a production environment:

1. On the **Environments** tab of the Business Central administration center, choose the **New** action on the action ribbon.
2. In the **Create Environment** pane, in the **Environment Type** list, choose **Production**.
3. In the **Country** list, select the country for the environment. The specified country determines the localization for the environment and the Azure region in which the environment is created and stored.
4. Select **Create**.

When the new production environment is created, it will be based on the latest production version of Business Central.

Create a new sandbox environment

A sandbox environment is a non-production instance of Business Central. Isolated from production, a sandbox environment is the place to safely explore, learn, demo, develop, and test the service without the risk of affecting the data and settings of your production environment.

> **IMPORTANT**
> Make sure that you understand the limitations of a sandbox before you create a new sandbox environment. For more information, see Sandbox environments section.

To create a sandbox environment:

1. On the **Environments** tab of the Business Central administration center, choose the **New** action on the action ribbon.
2. In the **Create Environment** pane, specify a name for the new environment.
3. In the **Create Environment** pane, in the **Environment Type** list, choose **Sandbox**.
4. Specify if you want the sandbox environment to contain a copy of another environment. If you choose this option, you must specify which environment to copy.

   When you create a sandbox environment as a copy of another environment, the new environment is created on the same application version as the environment that you are copying. The new environment will contain all per-tenant extensions and AppSource extensions that are installed and published in the original environment that is being copied.

   If a sandbox is created with a copy of a production environment, a number of precautions are taken for that sandbox:

   - The job queue is automatically stopped

- Any base application integration settings are cleared
- Outbound HTTP calls from extensions are blocked by default and must be approved for each extension

  To enable outbound HTTP calls, go to the **Extension Management** page in Business Central, and choose **Configure**. Then, on the **Extension Settings** page, make sure that **Allow HttpClient Requests** is selected. This setting must be enabled for each extension.
- Any General Data Protection Regulation (GDPR) action must be handled separately and repeated for the sandbox. There is no synchronization with the production environment after the sandbox has been created.

  The internal administrator has the same tools and responsibilities for a sandbox environment as they do for a production environment. As a data processor, Business Central offers the same level of data protection and data handling restrictions that we apply to production environments.

5. In the **Country** list, select the country for the environment. The specified country determines the localization for the environment and the Azure region in which the environment is created and stored.
6. Choose the relevant application version for the new sandbox environment from the **Version** list if more than one version is available.
7. Select **Create**.

> **NOTE**
> The sandbox environment won't be accessible until the **State** shows *Active*.

To delete a sandbox environment, choose the environment on the **Environments** tab of the Business Central administration center, and then choose **Delete** on the action ribbon.

Selecting a version for a new sandbox environment

If you create a sandbox that isn't a copy of an existing environment, you must specify an application version for the new environment. The version list will show the latest *production* version, which is the version used for new production environments.

The version list may also have one or more *preview* versions. Preview versions are early release candidates of upcoming releases of Business Central that are made available specifically for sandbox environments. This list gives you access to review new functionality, validate extension compatibility, and other general testing of the upcoming release.

When you create a sandbox environment on a preview version, the environment will automatically be updated to new preview versions when they become available. However, the environment won't be updated to the production version. Once a sandbox environment is on a preview version, it must stay on a preview version until it's deleted. The environment can also be deleted if an update between preview versions fails. We recommend that preview versions are used only for temporary testing of an upcoming release.

Managing Sessions

The **Manage Sessions** page displays information about active sessions on an environment and lets you cancel selected sessions.

To open the page, select **Manage Sessions**. Use the **Show session details** check box to show more or fewer details.

Cancel sessions

Canceling a session is sometimes the only way to unblock a customer. For example, a long-running report is locking data in a table, preventing warehouse employees from working.

To cancel a session, select it from the list and then select **Cancel selected sessions**.

Restoring an environment

> **NOTE**
> This feature is in preview. It might change or be removed in the future updates.

Database backups are an essential part of any business continuity and disaster recovery strategy, because they protect your data from corruption or deletion. Business Central online service uses Azure SQL Database as the underlying database backup technology for its environments. All databases are protected by automated backups that are continuously created and maintained by the Azure SQL service. The backup retention period for Business Central databases is set to 30 days for both production and sandbox environments.For more information about this backup process, see Automated backups - Azure SQL Database & SQL Managed Instance.

As an administrator, you can restore an existing environment from a time in the past, within the 30-day retention period. An environment can only be restored within the same Business Central version (minor and major).

Users who can restore environments

Permission to restore environments is limited to specific types of users: internal and delegated administrators. The following users are allowed to restore environments.

- Delegated administrators from reselling partners
- Administrators from the organization that subscribes to Business Central online

Also, these users must have the **D365 BACKUP/RESTORE** permission set assigned to their user account in the environment they're trying to export.

For more information about permissions sets and user groups, see Assign Permissions to Users and Groups.

Considerations and limitations

- Environments can only be restored if the customer has a paid Business Central subscription.
- Each environment can be restored up to 10 times in a calendar month.
- It's not possible to use the Business Central administration center to restore an environment that was previously deleted.

  If you end up in the situation where you need to restore a deleted environment, contact Microsoft Support for help. In such cases, Microsoft doesn't guarantee a restore operation will succeed or all data and extensions will be available in the restored database. So before you decide to delete an environment, it's important to ensure that the environment is no longer needed.
- An environment can only be restored within the same Azure region and country (Business Central localization) as the original environment.
- A production environment can be restored to either a **Production** or **Sandbox** type environment. A sandbox environment can only be restored a **Sandbox** type environment.
- When restoring a sandbox environment, all development extensions (that is, extensions published directly from Visual Studio Code) won't be available in the restored environment—even if they were present at the point-in-time you're restoring to). Additionally, any per-tenant extensions that depend on such development extensions will also not be available.

- Per-tenant extensions you may have uploaded that target the **next** version of the Business Central won't be available in the restored environment—even if they were uploaded at the point-in-time you're restoring to. Per-tenant extensions that were already installed will be available in the restored environment.
- Every AppSource and Business Central app in the restored environment will have the latest available hotfix installed automatically—even if the hotfix was introduced after the point-in-time you're restoring to.

Before you restore an environment

Here are a few important things to do when you're planning to restore an environment:

- Make sure you communicate the plan to restore an environment within your organization upfront, in good time.
- Typically, you want to stop users and external integrations from using the environment during restoration. Consider doing the following actions in the environment you're planning to restore:
  - Remove access to the environment for non-essential users, but make sure required users, like administrators, keep access. For more information, see Remove a user's access to the system.
  - Put all job queues to on hold. For more information, see Use Job Queues to Schedule Task.
- Consider renaming the environment. The users and external integrations won't be able to access it by its old name.

  When restoring an environment, you'll create a new environment that the database backup will be restored to. You can't use the same name for two environments of the same customer. So if you want the restored environment to have the same name as the original environment, rename the original environment before you run the restore operation. For example, you could change the name to include DONOTUSE.

  For more information, see Renaming an environment.

Restore an environment

To restore an environment, you'll have to provide a name for the environment and a date/time from which to restore the database.

1. Select **Environments** and then open the environment you want to restore.
2. Select **Restore**.
3. In the **Restore Environment** pane, specify the date and time in the past to which you want to restore the environment.
4. Select the type to be used for the restored environment.
5. Specify a name for the restored environment.
6. Select **Restore**.

   If there's no backup available for date and time you chose, select the available nearest backup, when prompted. This situation can occur, for example, if the environment was being updated to a new minor or major version during the specified time.

   > **NOTE**
   >
   > For newly created environments it may take up to 30 min for the backups to be initialized, so you may not be able to restore an environment if you have just created it.

7. When the process starts, you can go to the list of your environments and see the status of the restored environment. At first, you'll see the new environment with state **Preparing**. The original environment state remains as **Active**.

   The restore operation duration is affected by several factors. For large or highly active databases, the restore might take several hours. You can find more details about the factors that affect the recovery time at Recovery time.

   Once the restore is completed, the environment state will change to **Active**. If the restore operation fails, you can find the failure details on the **Operations** page. In this case, delete the failed environment, and then try to restore again. Contact Microsoft Support if the issue persists.

After you restore an environment

After restoring an environment, you should inspect and adjust data to prepare it for users. Consider enforcing these steps during this period:

- Remove access to the environment for non-essential users, but make sure required users, like administrators, keep access.
- Put all job queues in the restored environment to on hold immediately after restore.
- If needed, you can upload the per-tenant extensions targeting the next version of Business Central again.

The original environment will remain available and isn't affected by the restore operation. You can then get back to the original environment if you need to look up data. Or maybe you'll have to migrate some data to the restored environment. You can, for example, migrate data by using Business Central RapidStart services. For more information, see Migrate Customer Data.

> **IMPORTANT**
> You can restore your production environment into a new production environment even if doing so results in exceeding your number of environments or database capacity quotas. You can however only exceed this quota by one extra production environment, regardless of how many production environments you have available for your subscription. This capability is provided as an exception, to ensure that you can always restore your production environment in critical situations. You must return within your quota within 30 days following the restore by either removing the original production environment or by purchasing an additional production environment. Before removing the environment, we recommend you export the environment to an Azure storage container in case you need to access some data at a later point. This exception isn't available for restoring from and to sandbox environments.

When you're satisfied with the data in the restored database, enable the users, start the job queues, and let your organization know that the restore process is now completed and they can again use the environment.

Renaming an environment

> **NOTE**
> This feature is in preview. It might change or be removed in the future updates.

You can change the name of any environment. The name uniquely identifies the environment from your other environments. Before you change a name, you must consider that the name also is part of the environment's URL. The URL is used in links to the environment in various ways. So changing the name can have significant impact.

The renaming of an environment is logged and shown in the operations log.

Before you rename an environment

- Read the Environment rename considerations section to understand the consequences of renaming an

environment.

- Make sure you notify all your users, including any external service integrations you may have, about the upcoming URL change. It will let them prepare to update their references.
- Determine the best time to do the renaming. Renaming an environment requires a restart to the environment. We recommend doing this operation when no users are active in Business Central.

Rename an environment

1. Select **Environments**, then select the environment you want to rename.
2. On the **Environment Details** page, select **Rename**.
3. On **Rename environment** page, read the information.
4. Enter the new name, and then select **Rename**.
5. Confirm your intent to rename the environment.

At this point, the environment state will first change to **Preparing | Rename Scheduled**, then to **Active** again when the rename has been completed. The new name will be available immediately. The environment will no longer be accessible using the old environment name.

You can also review the log for the Rename operation on the **Operations** page afterwards.

Environment rename considerations

Changing the environment name can affect many scenarios and integrations. In the early stages of a customer implementation, it may be a low risk operation. But renaming an environment that's been used by customers for a while or integrated with many external services and components is risky. So carefully plan for it.

Here are some areas where the environment name is used, which will be affected when you change the environment name. Consider these areas and plan your communication before attempting to rename an environment:

- Web client URL, including links to web client bookmarked by users
- Deep links to specific pages within Business Central created by users or shared by them via e-mails, internal documentation or training portals, Teams channels, Word, and Excel documents. They're often exchanged among users in the same company, across companies, across environments, across tenants. Links created by users as desktop shortcuts. Links sent or created before the name change will no longer work after the name change.
- Integrations that embed the web client, for example, SharePoint apps composed of Business Central pages
- Integrations that launch the web client
- Partner-developed mobile apps, web applications, and so on. These apps likely originate from partners outside the customer's organization where the admin can't update URLS.
- Mobile apps, including Windows 10 store app for desktop/tablet. Affects only users who have modified the protocol handler to force the app to connect to environment with name other than "production". If the user keeps working with "production" on the mobile app (which is default now), and the admin renames the environment from "prod2" to "myprod", the mobile user isn't affected. Otherwise, the app would throw an error, and the user would have to exit using a newly created protocol handler link.
- Browser cache. Business Central stores the URL, including environment name, in some of its cached data. This data is cached browser-side, that is, in the user's browser and across devices. Admins typically don't have access or control this data cache. When users lose their cache, they lose the link modifications to all their pages and preferences.
- Web services URL, potentially affecting external integrations that use OData or SOAP
- Business Central add-ins and integrations with other Microsoft services

    - Outlook Add-in. The Add-In manifest that is saved to Exchange Server, either per-organization or per-user, includes the environment name.
    - Excel Add-in. Each user's Excel worksheet stores the environment name.
    - Power BI. All reports, including the default reports deployed from the Role Center, built before the rename will be affected. Also, Power BI apps installed before the rename would be affected. There's no automatic way to repair these items. The partner or user would have to manually update the connections.
    - Power Apps/Automate. All apps and flows built before rename would be affected with no automatic way to repair. The partner or user would have to manually update the connections.
    - CDS. CDS Virtual Entity setup stores environment name.
- Development scenarios
    - Publish to sandbox environment from Visual Studio Code. The launch.json file of extensions might contain the sandbox name, if different from "default". The files require source code updates.
    - CI/CD pipelines for test and deployment could be impacted by environment renames.
- Azure Application Insights logs and metrics

Log of administrative operations

The **Operations** section of Business Central administration center provides a log of operations that internal administrators and delegated administrators from the partner have made in the Business Central administration center or through the admin center API. Currently, the log includes the following operations: renaming environments restoring environments.

Use this log to see which operations were created and when. You can also access detailed error messages in this log, should any operation fail.

See also

Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Apps
Updating Environments
Managing Tenant Notifications
Introduction to automation APIs

Managing Apps

2/17/2021 • 5 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

A Business Central environment is built as a collection of apps. These apps include Microsoft apps and third-party apps, for example, apps from AppSource. The apps work together to provide customers with a broad set of features to address their various business, market, and industry needs.

Updates are frequently made available for these apps by Microsoft, partners, and ISVs. App updates add new features and fix known problems. To keep your environment up to date and running smoothly, you should check for and install the latest updates regularly.

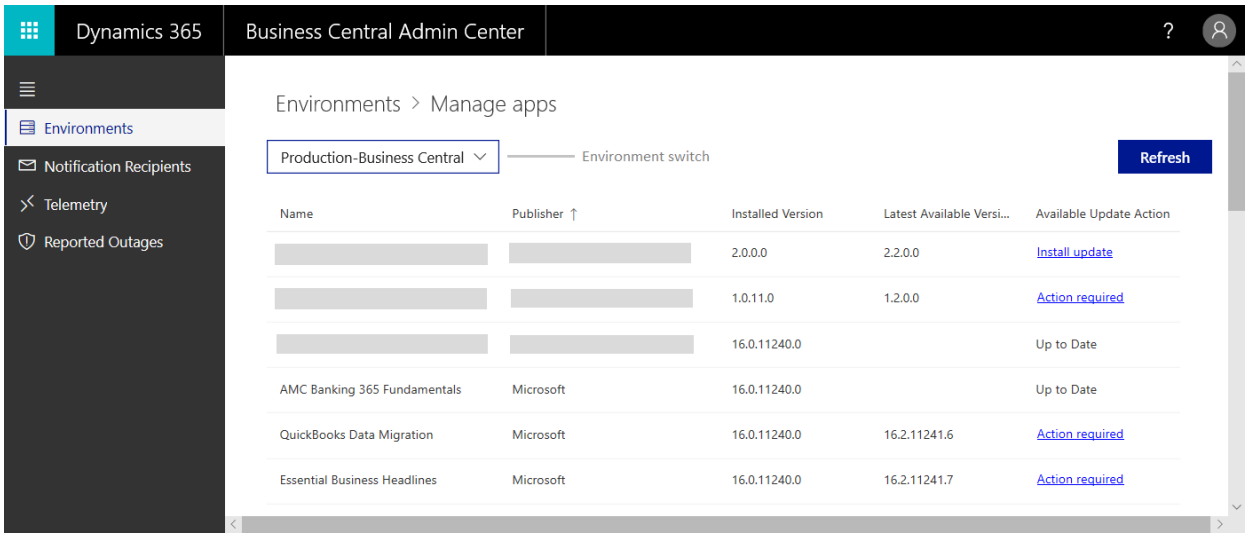
To help you manage app updates, the administration center includes the **Manage Apps** page.

Like other features in the administration center, this functionality can be used by the partner (delegated administrator) or a local customer admin.

> **NOTE**
> In the current version, it's not possible to install new apps, either main apps or their dependencies (library apps), by using the **Manage Apps**. To install the apps, continue using the **Extension Management** page within your Business Central environment.

Get an overview and check for updates

Open the **Manage Apps** page from the environment details page. Choose **Environments** > select the environment > **Manage Apps**.



The **Manage Apps** lists all the apps installed on the environment and indicates whether updates are available. When first opened, the system will start checking for updates. Wait for this operation to complete.

> **IMPORTANT**
> When an ISV provides a new version of their AppSource app, Microsoft validates it against the latest, currently available version of Business Central at the time. If the new app version passes validation, it's made available for the customers' environments that are running on that version of Business Central and greater. So if you're not seeing an AppSource app update in the list, your environment may not yet be running on the version the app was registered for.

When completed, if an update is available for an app, there are two indications:

- The **Latest Available Version** column contains the new version number of the app.
- The **Available Update Action** column contains one of the following actions:

ACTIONS	DESCRIPTIONS
Action required	This action means that you have to do something before you install the update for the app. For example, you may have to first update another app or install a new app. Select the link and read the **Requirements for App Updates** pane to see what is required. For more information, see Resolving requirements for app updates.
Install update	This action means that the app is ready to install. Select the link to start the installation.

Install an app update - the flow

We recommend you always install and test an app update on a Sandbox environment first. Make sure the app update won't disrupt the operational flow or cause problems for the users.

The following steps provide the general flow for updating an app.

1. If you don't have a sandbox environment, create one. For more information, see Create a new sandbox environment.
2. Open the Sandbox environment and select **Manage Apps**.
3. On the **Manage Apps**, find the app in the list that you want to update.
4. If the **Available Update Action** column for the app shows the **Action required** link, resolve the update requirements.

   See Resolving requirements for app updates.
5. When the **Available Update Action** column for the app shows **Install update**, select this action to install the new version of the app.

   > **IMPORTANT**
   > The update will be applied immediately after you accept the confirmation dialogue. The users can continue working during update installation, but depending on the app changes coming with the update, they may receive a message asking them to log out and login again. It is therefore recommended that you apply the updates outside of working hours.
6. Wait for the app to be installed.

   Select **Refresh** occasionally to check the status.

- If the app installs successfully, the new version displays in the **Installed version** column and the **Available Update Action** column is **Up to date**.
- If the installation fails the **Available Update Action** column changes to **Update failed**. See What to do when an update fails.

7. If the app update succeeded, sign in the Sandbox environment and test the new app version.
8. If the app update works as expected on the Sandbox, switch to the production environment, and repeat the installation steps for the app update.

> **TIP**
> Use the environment switch box at the top of the page to quickly change to your production environment.

Resolve requirements for app updates

For apps that have dependencies on other apps, you may have to update or install the dependency apps. The **Requirements for App Updates** page provide this information. The requirements are divided into two categories: **Update requirements** and **Install requirements**.

After you resolve all requirements, the app that you want to update will be ready to install.

Update requirements

The **Update requirements** category lists existing dependency apps the have updates to be installed. To resolve these requirements, do the following steps for each app:

1. Return to the **Manage Apps** page.
2. Find the app in the list and select **Install update**.
3. Wait for **Available update action** column to change to **Up to date**.

Install requirements

The **Install requirements** category lists dependency apps that haven't been installed yet. For example, a new app was introduced that app update depends on.

You can't, however, install a new app from the **Manage App** page. Use the **Extension Management** page in the client instead. Completing this step will resolve the requirement. For more information, see Installing an Extension.

What happens when an app update is installed?

The new app version is starting to install immediately, following the confirmation dialogue. The new app version will be published, synchronized, and updated in the background. This process usually doesn't take long, and users won't be interrupted. However, we still recommend you to install the updates outside of working hours.

What do I do when an update fails?

When the installation of an app update fails, the **Available Update Action** column will display the **Update failed** action. Select this action to get more information. The **App Update Details** pane provides some details about update and what might have caused the failure.

Sometimes the update could fail because of a transient problem. Select **Retry** to try to install the update again. If the installation continues to fail, contact your ISV. You can find the support details of each ISV on their app page on AppSource. Contact Microsoft support if the app publisher is **Microsoft**.

TIP

When reporting issues to Microsoft Support, always provide the **Operation ID** displayed in the error message. This will help expedite the investigations.

See also

Working with Administration Tools
The Business Central Administration Center
Production and Sandbox Environments
Managing Environments
Updating Environments
Managing Tenant Notifications
Introduction to automation APIs

Managing Capacity

2/17/2021 • 4 minutes to read • Edit Online

To help customers manage and plan their storage costs on an ongoing basis, the Business Central administration center includes the **Capacity** page. The **Capacity** page provides an overview of the total database and file storage usage, with details about the storage used by every environment. The page also displays the currently used and the maximum allowed number of production and sandbox environments available for the customer.

Customers can purchase additional database capacity and environments via their reselling partner by using the following add-ons:

- Dynamics 365 Business Central Database Capacity
- Dynamics 365 Business Central Additional Environment Add-on

Number of environments

Business Central administrators can create multiple sandbox and production type environments for various purposes, like:

- Creating more business branches
- Moving into more countries
- Expanding within their current country
- Development
- Testing changes
- Learning new product capabilities

Every Business Central customer with Business Central Premium or Essential subscriptions can use one production environment and three sandbox environments, at no extra charge.

> **NOTE**
> Existing customers with Business Central Premium or Essentials subscriptions purchased before October 1st 2020 will keep their existing environment entitlements for a period of 1 year until October 1st 2021 or until their subscription is renewed or ended, whichever occurs last. This won't increase their overall database storage capacity and won't enable extra sandbox environments. These benefits are only activated with the production environments purchased by the customers separately.

Customers can also choose to purchase any number of additional production environments via their CSP partner. Each production environment comes with three additional sandbox environments.

Production and sandbox environments can be created and used in any country where Business Central service is available, also in the country where the default Business Central environments are located. Additional environments can be created by customers, administrators, and partners by using the Business Central administration center.

When customer administrators create users in Microsoft 365 Admin Center and assign them Business Central licenses, each user, by default, gets access to all Business Central environments (sandbox and production) under the same single Business Central license, still acting within the scope of their license within each of these environments. Administrators can limit users' access to any particular environment by changing their permissions, or by removing users' access within that environment.

Storage

Storage capacity usage of Business Central is represented by **Database** and **File** on the **Capacity** page.

Business Central customers can use up to 80 GB of database storage capacity across all of their environments (production and sandbox), meaning that the sum of database capacity usage of all of their environments must not exceed 80 GB.

Some businesses have unique scenarios that may require additional storage. For those organizations that need more space, there is an option to purchase extra database capacity as an add-on to their existing Business Central subscriptions through their reselling CSP partner. Every add-on increases overall database capacity quota by 1 GB.

Every additional production environment purchased by the customer also increases tenant-wide database capacity quota by 4 GB.

> **TIP**
> Use the **Capacity** page to help manage the migration when you migrate to Business Central online. That way, you know if your migration will bring you close to or over the current limits on database size, for example.

Storage usage by environment

The **Storage usage by environment** section of the **Capacity** page provides a tenant-level view of where your organization is using storage capacity. Here you can see how much database and file storage is used by each environment. For each of your environments, you can also navigate to the Table Information page within Business Central, which let's you see the distribution of data size across tables.

> **NOTE**
> Only the environments running on version 17 (2020 release wave 2) and later are included in the database capacity calculations.

The **File usage** storage is calculated based on the size of the content of the following tables:

- Tenant Media
- Tenant Media Thumbnails

The content of the other tables is counted towards the **Database usage** storage.

> **IMPORTANT**
> When you uninstall extensions, you have a choice of deleting or leaving the extension data in the database. If you decide not to delete the data when uninstalling extensions, this data will be counted in the overall database storage you use.

Exceeding capacity quotas

Exceeding the paid database storage limit won't interrupt transaction processing within the existing environments. The existing environments that organically grow and eventually exceed the quota will still be accessible and available for the customers to continue their business operations.

However, once the capacity limits are exceeded, the customers won't be able to create new environments or copy their existing environments until the storage used by the existing environments is decreased to fit the quota or additional capacity is purchased.

Administrative actions are currently not limited by the File storage, however similar restrictions will be applied

to the File storage with one of the next releases of Business Central.

Reducing Data Stored in Databases

There are a few things that you can do to reduce the amount of data stored in a database to keep it under its current limit. For more information, see Reducing Data Stored in Databases.

See also

Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Apps
Updating Environments
Managing Tenant Notifications
Introduction to automation APIs

Managing Tenant-Specific Notifications

2/17/2021 • 2 minutes to read • Edit Online

Notifications are sent by email for administrative events that occur on the tenant. For example, notifications are sent when a major update is available for tenant environments, when an environment update has succeeded or failed, or when extensions require changes to be compatible with an upcoming release. When these and other similar events occur on the tenant, an email is sent to the notification recipients for the tenant.

Notification recipients

Notifications are sent to all email addresses that are listed in the **Notification recipients** list of the Business Central administration center. The list is managed manually by adding and removing recipients to ensure the right individuals are notified of the event.

> **NOTE**
>
> It is important that *at least* one administrator's email address has been entered as a notification recipient to ensure proper awareness of events requiring administrative attention.

> **IMPORTANT**
>
> To not miss update notifications from Microsoft, you must verify that the e-mails are not redirected to a spam folder by your e-mail software. The notifications are sent from the Microsoft Partner Center address, `msftpc@microsoft.com` and contain `Dynamics 365 Business Central` in the subject line.

Cleaning up settings

If you end the relationship with a customer where you have set up your email address as a notification recipient, you must remove the email address while you still have access to that customer's Business Central administration center.

See also

Managing Updates in the Business Central Admin Center
The Business Central Administration Center
Managing Environments
Introduction to automation APIs
Working with Administration Tools

Managing Updates in the Business Central Admin Center

2/17/2021 • 3 minutes to read • Edit Online

Business Central environments are updated according to the Business Central roadmap with two major updates in April and October each year, and monthly service updates. For more information, see Dynamics 365 Release Plans.

Updates of the base application and platform are managed by Microsoft. As an internal administrator or as a partner, you can use the Business Central administration center to specify certain parameters of the timing of updates for each environment, and you can specify the people who must receive notifications of when an update is available. You can also help prepare your solution and your users by creating preview environments so that you can get acquainted with new functionality in the product. For more information, see Major Updates of Business Central Online.

Set the update window for each environment

The update window for an environment defines a window of time during the day in which the environment can be updated. When an update is rolling out to Business Central online, regardless of whether it's the monthly service update or a major update, the update will be applied to an environment within the time frame that the update window defines. This helps to ensure that updates are applied outside of the normal business hours of the organization, for example.

The update window must be a minimum of six hours.

> **NOTE**
> Desktop users who are signed in during the update will receive an alert in Business Central.

To set the update window for an environment:

1. On the **Environments** tab of the Business Central administration center, choose the **Name** of the relevant environment to open the environment details.
2. Choose the **Set update window** action on the **Update** list on the action ribbon.
3. In the **Set update window** pane, specify the start time and the end time for the update window for the environment.

   > **NOTE**
   > The update window must be a minimum of six hours.

4. Choose **Save**.

Schedule an update date

Specifically for major updates, you can choose a specific date on which the environment is updated. The **Update version** field in the **Version Management** section of the environment details also displays the version number of the available update version.

To schedule an update date:

1. On the **Environments** tab of the Business Central administration center, choose the **Name** of the relevant environment to open the environment details.
2. Choose the **Schedule Update** action on the **Update** list on the action ribbon.
3. In the **Schedule Environment Update** pane, specify the update date.

   > **NOTE**
   > The specified date must be within a given date range that is shown in the pane.

4. Choose **Schedule Update**.

If an administrator has chosen the **Schedule Update** action but not set a date, then the update is applied automatically to each tenant environment with a default date range. The default date range is communicated in advance to tenant administrators through administrative notifications. You can then choose to override that with a custom date by following the steps that are provided above. Not selecting an update date does not prevent the environment from being updated.

Get notified of updates

For updates for which the option is available for tenant administrators to schedule the update date, a notification is sent to the notification recipients listed on the **Notification recipients** tab of the Business Central administration center. For more information, see Managing Tenant Notifications.

> **IMPORTANT**
> To not miss update notifications from Microsoft, you must add notification recipients and verify that the e-mails are not redirected to a spam folder by your e-mail software. The notifications are sent from the Microsoft Partner Center address, `msftpc@microsoft.com` and contain `Dynamics 365 Business Central` in the subject line.

Scheduling environment updates is *not* available for monthly service updates. For these, the update is applied to tenant environments as it becomes available. No notifications are sent to tenant administrators prior to the update. Notifications are sent only after the update is applied.

See also

Major Updates of Business Central Online
Prepare for major updates with preview environments
Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Tenant Notifications
Introduction to automation APIs

Exporting Databases

2/17/2021 • 5 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

From the Business Central administration center, you can export the database for Business Central online environments as BACPAC files to an Azure storage container.

Considerations before you begin

- You can only request a database export for production environments. If you want to export data from a sandbox environment, you can use Excel or RapidStart.
- You must have explicit permission to export databases. For more information, see the Users who can export databases section.
- You can't export your database to an Azure premium storage account. The steps in this article are only supported on Azure standard storage accounts.

Setting up Azure storage

Before you can export the file, you must first set up the Azure storage account container that the BACPAC file will be exported to.

Creating the storage account

The first step is to create an Azure standard storage account, if you don't already have one. To set up the export, you must first have a subscription to Microsoft Azure and access to the Azure portal.

For more information setting up an Azure storage account, see Create a storage account.

Generating a shared access signature (SAS)

The next step is to generate a shared access signature (SAS) that provides secure delegated access to your storage account. Business Central uses the signature to write the BACPAC file to your storage account.

To generate a shared access signature (SAS)

1. On the Azure storage account, choose **Shared access signature** in the navigation pane.
2. In the **Allowed services** section of the shared access signature pane, select **Blob**, and clear the other options.
3. In the **Allowed resource types** section, select **Container** and **Object**, and clear the other options.
4. In the **Allowed permissions** section, mark **Read**, **Write**, **Delete**, and **Create**, and clear the other options.
5. Select a start and end date and time for the SAS. A minimum expiration window of 24 hours from the initiation of the export is required.

> **TIP**
> It is a best practice to use near-term expiration for the account's SAS. To reduce risk of a compromised storage account, set the end date and time no later than what is needed for you to complete the database export operation. However, the SAS must be valid for a minimum of 24 hours.

6. In the **Allowed protocols** section, select **HTTPS only**.
7. Select **Generate SAS and connection string**.
8. Copy the **Blob service SAS URL**.

For more information on generating and using a SAS, see Grant limited access to Azure Storage resources using shared access signatures (SAS).

Creating the database export

When you've created the Azure storage account and generated the SAS URI, you can then create the export file from the Business Central administration center.

1. On the Environments list page, choose the relevant production environment to view the environment details.
2. On the action ribbon of the environment details, choose **Database**, and then choose **Create Database Export**.
3. In the **File Name** field, specify a name for the export file, or leave the default value.
4. In the **SAS URI** field, specify the **Blob service SAS URL** value that you copied in the previous section.
5. In the **Container Name** field, enter the name of the container in the Azure storage account to which you want the BACPAC file exported. If you've already created a container in your Azure storage account, you can enter the name of that container here. Otherwise, if the name that is specified in the **Container Name** field doesn't already exist in the Azure storage account, it will be created for you.

Once the export operation begins, the BACPAC file is generated and exported to the indicated Azure storage account. The operation may take several minutes to several hours depending on the size of the database. You can close the browser window with the Business Central administration center during the export. When the export completes, you can access the export file in the defined container in your Azure storage account. Optionally, you can import the data into a new database in Azure SQL Database or SQL Server for further processing. For more information, see Quickstart: Import a BACPAC file to a database in Azure SQL Database.

> **NOTE**
> There is a limit to the number of times you can export the database for each environment in any given month. The **Create Database Export** pane includes information about the number of exports still remaining that month.

Viewing the export history

All database export activity is logged for auditing purposes. To view the history, choose **Database**, then choose **View Export History** on the environment details page of the environment.

Users who can export databases

Permission to export databases is limited to specific types of users: internal and delegated administrators. The following users are allowed to export databases.

- Delegated administrators from reselling partners
- Administrators from the organization that subscribes to Business Central online

Also, these users must have the **D365 BACKUP/RESTORE** permission set assigned to their user account in the environment they're trying to export.

For more information about permissions sets and user groups, see Assign Permissions to Users and Groups.

Using the exported data

The BACPAC file contains data that is customer-specific business data. Technically, Business Central online is a multitenant deployment, which means that each customer tenant has their own business database while the data that defines the application is in a shared application database.

This means that if you want to export the data in order to change the customer's Business Central from an online deployment to an on-premises deployment, you must also get the application data from the installation media from the same version of Business Central that the online tenant is using. You can see the version number in the Business Central administration center or the **Help and Support** page in the customer's Business Central.

> **IMPORTANT**
> While you can import the downloaded BACPAC file into your own SQL Server instance, Microsoft does not provide support for creating a working on-premises environment from the BACPAC that you download from Business Central online.

For more information, see Quickstart: Import a BACPAC file to a database in Azure SQL Database, Migrating to Single-Tenancy From Multitenancy, and When to choose on-premises deployment.

See also

SQL Server technical documentation
Quickstart: Import a BACPAC file to a database in Azure SQL Database
Delegated Administrator Access to Business Central Online
Working with Administration Tools
The Business Central Administration Center
Managing Environments
Updating Environments
Managing Tenant Notifications
Introduction to automation APIs

Environment Telemetry in the Business Central administration center

2/17/2021 • 3 minutes to read • Edit Online

The Business Central administration center provides telemetry for the tenant environments to enable troubleshooting and support for the tenant. The Telemetry tab provides telemetry of top-level AL events, and any errors resulting from calls through the telemetry stack.

To filter the telemetry for an environment:

1. Select a base point-in-time for the timestamp of the telemetry messages.
2. Enter a number of minutes before or after the base point-in-time to set a range of time for the timestamp. A negative number indicates a number of minutes before the base point-in-time, and a positive number indicates a number of minutes following the base point-in-time. For example, a value of *-15* filters messages to a timestamp range of up to 15 minutes before the base point-in-time.
3. Choose the message type.
4. Choose the environment.
5. Select **Filter**.

> **TIP**
> If your users complain of a confusing error message such as *Sorry, we just updated this page. Please close and reopen.*, then you can often find the underlying problem by analyzing telemetry in the Business Central administration center. For example, in the case of the *Sorry, we just updated this page. Please close and reopen.* message, the underlying problem is often that two users are trying to modify the same data. So if both users open the same sales order, and both change a field, then one of them will see the *Sorry, we just updated this page. Please close and reopen.* message, because Business Central saves changes as soon as you move to the next field or close the page.

Sending telemetry to Microsoft Azure Application Insights

APPLIES TO: Business Central 2019 release wave 2 and later

You can set up your environments to send telemetry to Application Insights. Application Insights is a service hosted within Azure that gathers telemetry data for analysis and presentation. For more information, see What is Application Insights?.

For an overview of the telemetry types that are currently emitted, see Monitoring and Analyzing with Telemetry.

> **IMPORTANT**
> Currently, emitting data to Azure Application Insights resources in Germany regions, like **(Europe) Germany West Central** or **(Europe) Germany North** , doesn't work. Until this issue is fixed, the mitigation is to create an Azure Application Insights resource in a region outside of Germany. Then, when the issue has been fixed, move the resource to the preferred region.

Enable Application Insights

1. If you don't already have one, get a subscription to Microsoft Azure.
2. Create an Application Insights resource in Azure.

The Application Insights resource will be assigned an instrumentation key. Copy this key because you'll need it to enable Application Insights in the Business Central administration center.

The Application Insights resource can be in any Azure tenant that is accessible to your organization. For example, a delegated administrator from the reselling partner is the person analyzing the telemetry. But this person might not have access rights the customer's Azure instance. This scenario enables the partner to send the telemetry to their own Application Insights instance.

> **TIP**
> You can use the same Application Insights resource for multiple tenants and their different environments.

For more information, see Create an Application Insights resource.

3. In the Business Central administration center, select **Environments**, and then select the environment that you want to change.

> **IMPORTANT**
> The next steps require a restart to the environment, which is triggered automatically after step 5. Plan to do this during non-working hours to avoid disruptions.

4. On the **Environment** page, the **Application Insights Key** field shows if the environment already uses application insights.

   To enable application insights, choose the **Define** caption, and then, in the **Set Application Insights Key** pane, choose the **Enable application insights** field and enter the instrumentation key in the **Instrumentation Key** field.

> **NOTE**
> In version 15 and 16, to enable application insights, choose the **Application Insights Key** action, and then specify the instrumentation key.

5. Choose the **Save** button.

Cleaning up settings

If the Application Insights resource is tied to your partner account, and you end the relationship with a customer where you have set up telemetry based on your account's instrumentation key, you must remove the instrumentation key while you still have access to that customer's Business Central administration center.

See also

Monitoring and Analyzing Telemetry
Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Tenant Notifications
Introduction to automation APIs
Enabling Application Insights for On-Premises

The Business Central Administration Center API

2/17/2021 • 33 minutes to read • Edit Online

The Business Central administration center API enables administrators to programmatically do administrative tasks for a Business Central tenant. With the API, administrators can, for example:

- Query and work with production and sandbox environments for the tenant.
- Set up administrative notifications.
- View telemetry for events on the tenant.

For more information about administrative capabilities, see The Business Central Administration Center. This article describes the API contracts for these administrative capabilities.

> **IMPORTANT**
>
> For delegated admin access, you must add the Azure Active Directory (Azure AD) application to the **AdminAgents** group. If the Azure AD application is not added, the consent flow will show an error such as *Need pre-consent*. For more information, see Pre-consent your app for all your customers in the Graph documentation.

Location

The Business Central administration center API is located at the following URL: https://api.businesscentral.dynamics.com.

Setting up Azure Active Directory (Azure AD) based authentication

Sign in to the Azure portal to register your client application as an app and enable it to call the Business Central administration center API.

1. Follow the instructions in the Integrating applications with Azure Active Directory article. The next steps elaborate on some of the specific settings you must enable.
2. Give the application a **Name**, such as **Business Central Web Service Client**.
3. For **Application type**, choose either **Native** or **Web app/API** depending on your scenario. The code examples below assume **Native**.
4. Choose a **Redirect URI**. If it's a **Native** app, you can choose for example: **BusinessCentralWebServiceClient://auth**. If it's a **Web app/API** app, set the value to the actual URL of the web application.
5. During the registration of the app, make sure to go to **Settings**, and then under **API ACCESS**, choose **Required permissions**. Choose **Add**, and then under **Add API Access**, choose **Select an API** and search for the **Dynamics 365** option. Choose **Dynamics 365**, select **Delegated permissions**, and then choose the **Done** button.

   > **NOTE**
   >
   > If **Dynamics 365** doesn't show up in search, it's because the tenant doesn't have any knowledge of Dynamics 365. To make it visible, an easy way is to register for a free trial for Dynamics 365 Business Central with a user from the directory.

6. Make a note of both the **Application ID** and the **Redirect URI**. They'll be needed later.

Getting an access token

HTTP requests sent to the Business Central administration center API must include the Authorization HTTP header, and the value must be an access token.

The following examples show how to obtain such a token using PowerShell. Using C# is straightforward.

PowerShell example without prompt:

```
   $cred = [Microsoft.IdentityModel.Clients.ActiveDirectory.UserPasswordCredential]::new($UserName,
$Password)
   $ctx =
[Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext]::new("https://login.windows.net/$Ten
antName")
   $token =
[Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContextIntegratedAuthExtensions]::AcquireToke
nAsync($ctx, "https://api.businesscentral.dynamics.com", <Application ID>,
$cred).GetAwaiter().GetResult().AccessToken
```

PowerShell example with prompt:

```
   $ctx =
[Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext]::new("https://login.windows.net/$ten
antName")
   $redirectUri = New-Object -TypeName System.Uri -ArgumentList <Redirect URL>
   $platformParameters = New-Object -TypeName
Microsoft.IdentityModel.Clients.ActiveDirectory.PlatformParameters -ArgumentList
([Microsoft.IdentityModel.Clients.ActiveDirectory.PromptBehavior]::Always)
   $token = $ctx.AcquireTokenAsync("https://api.businesscentral.dynamics.com", <Application ID>,
$redirectUri, $platformParameters).GetAwaiter().GetResult().AccessToken
```

Error Format

If an error occurs during the execution of an API method, it will respond back with an error object. While the specifics of any error will vary from endpoint to endpoint and by the error, the error object returned should adhere to the following structure. When an error occurs that doesn't fit this structure, it typically indicates that an error occurred in sending the request or during authentication of the request. For example, it could be that the API hasn't yet received the request.

Error Response Object:

```
{
"code": string, // A stable error code describing the type an nature of the error. Ex: EnvironmentNotFound
"message": string, // A message with a readable description of the error and cause. Intended to assist with
debugging or troubleshooting the API, it's not intended to be displayed.
("target": string), // Optional - Provides information about what part of a request caused the error. Ex:
The name of a property on the request body.
("extensionData": {...}), // Optional - A key/value dictionary object containing additional information
about the error.
("clientError": [ // Optional - A nested list of error objects containing more details about the error
encountered. For instance, this may be used if multiple errors are encountered to list them all out.
  {
    "code": string,
    "message": string,
    "target": string,
    "extensionData": {...},
    "clientError": [...]
  })
]
}
```

General unhandled errors

All unknown and unhandled errors that aren't covered by the lists above will use the error code: **Unknown**

Environments

Environments are the instances of the application that have been set up for the tenant. An instance can be of either a production type or a sandbox type. The environment APIs can be used to:

- Get information about the environments currently set up for the tenant
- Get information about the used storage and allowed quotas
- Create a new environment using sample data or as a sandbox copy of the production environment
- Delete an environment.

> **NOTE**
> Special care should be taken when deleting a production environment as the data will not be recoverable

Get environments and Get environments by application family

Returns a list of all the environments for the tenant.

```
GET /admin/v2.3/applications/environments
```

Returns a list of the environments for the specified application family.

```
GET /admin/v2.3/applications/{applicationFamily}/environments
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral)

Response

Returns a wrapped array of environments.

```
{
  "value":
  [
    {
      "friendlyName": string, // Display name of the environment
      "type": string, // Environment type (for example, "Sandbox", "Production")
      "name": string, // Environment name, unique within an application family
      "countryCode": string, // Country/Region that the environment is deployed in
      "applicationFamily": string, // Family of the environment (for example, "BusinessCentral")
      "aadTenantId": Guid, // Id of the Azure Active Directory tenant that owns the environment
      "applicationVersion": string, // The version of the environment's application
      "status": string, // (enum | "NotReady", "Removing", "Preparing", "Active")
      "webClientLoginUrl": string, // Url to use to log into the environment,
      "webServiceUrl": string, // Url to use to access the environment's service API
      "locationName": string, // The Azure location where the environment's data is stored
      "platformVersion": string, // The version of the environment's Business Central platform
      "ringName": string, // Name of the environment's logical ring group (such as  Prod, Preview)
      "appInsightsKey": string // The environment's key for Azure Application Insights
    }
  ]
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

Get environment by application family and name

Returns the properties for the provided environment name if it exists.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Response

Returns a single environment if exists.

```
{
  "friendlyName": string, // Display name of the environment
  "type": string, // Environment type (for example, "Sandbox", "Production")
  "name": string, // Environment name, unique within an application family
  "countryCode": string, // Country/Region that the environment is deployed in
  "applicationFamily": string, // Family of the environment (for example, "BusinessCentral")
  "aadTenantId": Guid, // Id of the Azure Active Directory tenant that owns the environment
  "applicationVersion": string, // The version of the environment's application
  "status": string, // (enum | "NotReady", "Removing", "Preparing", "Active")
  "webClientLoginUrl": string, // Url to use to log into the environment,
  "webServiceUrl": string, // Url to use to access the environment's service API
  "locationName": string, // The Azure location where the environment's data is stored
  "platformVersion": string, // The version of the environment's Business Central platform
  "ringName": string, // Name of the environment's logical ring group (such as  Prod, Preview)
  "appInsightsKey": string // The environment's key for Azure Application Insights
}
```

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- `target: {applicationFamily}/{environmentName}`

Create new environment

Creates a new environment with sample data.

```
Content-Type: application/json
PUT /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}
```

Route Parameters

`applicationFamily` - Family to create the new environment within. (for example, "BusinessCentral")

`environmentName` - The name of the new environment. See the section below about valid environment names to see what values are allowed.

Body

```
{
  "environmentType": string, // The type of environment to create (enum | "Production", "Sandbox")
  "countryCode": string, // The country to create the environment within
  ("ringName": string), // Optional - The logical ring group to create the environment within. Currently
only Sandbox type environments may be created in a 'Preview' ring. If not provided then the production ring
will be used.
  ("applicationVersion": Version), // Optional - The version of the application the environment should be
created on. If not provided then the latest available version will be used.
}
```

> **NOTE**
>
> The values for the `countryCode`, `ringName`, and `applicationVersion` properties should be derived from the API endpoints described in the Available Applications section below.

Response

Returns newly created environment.

```
{
  "friendlyName": string, // Display name of the environment
  "type": string, // Environment type (for example, "Sandbox", "Production")
  "name": string, // Environment name, unique within an application family
  "countryCode": string, // Country/Region that the environment is deployed in
  "applicationFamily": string, // Family of the environment (for example, "BusinessCentral")
  "aadTenantId": Guid, // Id of the Azure Active Directory tenant that owns the environment
  "applicationVersion": string, // The version of the environment's application
  "status": string, // (enum | "NotReady", "Removing", "Preparing", "Active")
  "webClientLoginUrl": string, // Url to use to log into the environment,
  "webServiceUrl": string, // Url to use to access the environment's service API
  "locationName": string, // The Azure location where the environment's data is stored
  "platformVersion": string, // The version of the environment's Business Central platform
  "ringName": string, // Name of the environment's logical ring group (such as  Prod, Preview)
  "appInsightsKey": string // The environment's key for Azure Application Insights
}
```

Expected Error Codes

`DoesNotExist` - the provided value for the application family wasn't found

`destinationApplicationServiceNotFound` -- a suitable destination service couldn't be found to put the environment. Occurs if missing or if the tenant doesn't have access to the target application.

`invalidInput` - the targeted property in invalid in some way

- target: {countryCode} - the country code property can't be null or whitespace
- target: {environmentName} - the environment name property can't be null or whitespace

- target: {environmentType} - the environment type property can't be null or whitespace, and must be a valid value (Production or Sandbox)
- target: {ringName} - attempt to create a production environment on a non-production ring
- target: {applicationVersion} - the version property must be a valid version string (major.minor.build.revision)

`requestBodyRequired` - the request body must be provided

`resourceDoesNotExist` - the targeted property doesn't exist

- target: {ringName} - a ring with the provided name wasn't found

`resourceExists` an environment with the same name already exists

`environmentNameNotValid` - the environment name can't be a reserved value, must be less than 30 characters, must start with an alpha character and consist only of alpha, numberical, underscore (_), or dash (-) characters

`cannotCreateNamedEnvironment` - environments with names other than 'Production' or 'Sandbox' aren't supported on the targeted version.

`tenantAlreadyProvisioning` - can't create a new environment because another environment is currently in the process of being created.

`applicationFamilyNotAccessible` - the calling tenant doesn't have access to the targeted application family and country code

`environmentReservationFailed` -- another environment within the same application family already has this name

`maximumNumberOfEnvironmentsAllowedReached` - the limit on the number of allowed environments of the provided type has been reached

`maximumStorageCapacityUsageReached` - the limit of the storage capacity usage has been reached

Copy environment

Creates a new environment with a copy of another environment's data.

```
Content-Type: application/json
POST /admin/v2.3/applications/{applicationFamily}/environments/{sourceEnvironmentName}
```

Route Parameters

`applicationFamily` - Family of the source environment's application. (for example, "BusinessCentral")

`sourceEnvironmentName` - Name of the environment to copy from.

Body

```
{
  "environmentName": string, // The name of the new environment.
  "type": string // The type of environment to create. Currently only the value "Sandbox" is supported.
}
```

Response

Returns newly copied environment.

```
{
  "friendlyName": string, // Display name of the environment
  "type": string, // Environment type (for example, "Sandbox", "Production")
  "name": string, // Environment name, unique within an application family
  "countryCode": string, // Country/Region that the environment is deployed in
  "applicationFamily": string, // Family of the environment (for example, "BusinessCentral")
  "aadTenantId": Guid, // Id of the Azure Active Directory tenant that owns the environment
  "applicationVersion": string, // The version of the environment's application
  "status": string, // (enum | "NotReady", "Removing", "Preparing", "Active")
  "webClientLoginUrl": string, // Url to use to log into the environment,
  "webServiceUrl": string, // Url to use to access the environment's service API
  "locationName": string, // The Azure location where the environment's data is stored
  "platformVersion": string, // The version of the environment's Business Central platform
  "ringName": string, // Name of the environment's logical ring group (such as  Prod, Preview)
  "appInsightsKey": string // The environment's key for Azure Application Insights
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

`destinationApplicationServiceNotFound` -- a suitable destination service couldn't be found to put the environment. Occurs if missing or if tenant doesn't have access rights to the target application.

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{sourceEnvironmentName}```

`invalidInput` - the targeted property in invalid in some way

- target: {environmentName} - the environment name property can't be null or whitespace```
- target: {type} - the type property can't be null or whitespace, and must be a valid value (Currently only Sandbox)```

`requestBodyRequired` - the request body must be provided

`resourceExists` an environment with the same name already exists

`environmentNameNotValid` - the environment name can't be a reserved value, must be fewer than 30 characters, must start with an alpha character and consist only of alpha, numerical, underscore (_), or dash (-) characters

`cannotCreateNamedEnvironment` - environments with names other than 'Production' or 'Sandbox' aren't supported on the targeted version.

`tenantAlreadyProvisioning` - can't create a copy of an environment because another environment is currently in the process of being created.

`conflictingDeveloperExtensions` - The source environment contains 'uploaded' extensions that are already published to the destination service as 'developer' extensions. This condition will cause conflicts.

```
extensionData:
{
    "conflictingExtensions": [{
        "appId": guid, // The id of the conflicting extension
        "name": string, // The name of the conflicting extension
        "publisher": string, // The name of the person or group who published the conflicting extension
        "version": string, // The version of the conflicting extension
        "developerEnvironmentName": string // The name of the environment if the conflicting extension
exists in another environment owned by the tenant requesting the copy operation
    }],
    "sameAadTenant": bool, // Indicates if the conflicts occur on an environment that is owned by the same
tenant that is requesting the copy operation
}
```

`environmentReservationFailed` -- another environment within the same application family already has the same name

`maximumNumberOfEnvironmentsAllowedReached` - the limit on the number of allowed environments of the provided type has been reached

`maximumStorageCapacityUsageReached` - the limit of the storage capacity usage has been reached

Delete environment

Deletes the specified environment. Warning: A production environment shouldn't be deleted.

```
DELETE /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}
```

Route Parameters

`applicationFamily` - Family of the environment's application. (for example "BusinessCentral")

`environmentName` - Name of the environment to delete.

> **NOTE**
> The Create, Copy, and Delete operations are asynchronous. The response objects are returned before the underlying operation has completed. The final results of the operation are reflected in the 'status' field of the environment that the operations affect. In practice this means that polling of the 'Get Environments' endpoints must be done to determine if the given operation was successful.

Expected Error Codes

`invalidStatusCannotDeleteTenant` - can't delete the environment in its current state

`tenantDeletionInProgress` - environment is already in the process of being deleted

`ambiguousRequest` - multiple environments with the same name were found

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

Get used storage of an environment by application family and name

Returns used storage properties for the provided environment name if it exists.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/usedstorage
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Response

Returns used storage information of a single environment if exists.

```
{
  "environmentType": string, // Environment type (for example, "Sandbox", "Production")
  "environmentName": string, // Environment name, unique within an application family
  "applicationFamily": string, // Family of the environment (for example, "BusinessCentral")
  "fileStorageInKilobytes": int, // Used file storage in kilobytes
  "databaseStorageInKilobytes": int // Used database storage in kilobytes
}
```

> **NOTE**
>
> If an error occurs when calculating file storage or database storage the corresponding property will be -1.

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- `target: {applicationFamily}/{environmentName}`

Get used storage for all environments

Returns a list of used storage objects for all the environments.

```
GET /admin/v2.3/environments/usedstorage
```

Response

Returns a wrapped array of used storage objects.

```
{
  "value":
  [
    {
       "environmentType": string, // Environment type (for example, "Sandbox", "Production")
       "environmentName": string, // Environment name, unique within an application family
       "applicationFamily": string, // Family of the environment (for example, "BusinessCentral")
       "fileStorageInKilobytes": int, // Used file storage in kilobytes
       "databaseStorageInKilobytes": int // Used database storage in kilobytes
    }
  ]
}
```

Get allowed quotas

Returns different types of quotas and their limits.

```
GET /admin/v2.3/environments/quotas
```

Response

Returns quotas object.

```
{
  "productionEnvironmentsCount": int, // Maximum allowed number of production environments
  "sandboxEnvironmentsCount": string, // Maximum allowed number of sandbox environments
  "fileStorageInKilobytes": int, // Maximum allowed file storage in kilobytes
  "databaseStorageInKilobytes": int // Maximum allowed database storage in kilobytes
}
```

Available Applications

Get information about the currently available application families, countries, rings, and versions that environments can be created on. The API endpoints here should be utilized to determine what values can be used for environment creation or copying

Applications and corresponding Countries with Rings

Get a list of the currently available application families, the available countries within those families, and the available rings within the countries.

```
GET /admin/v2.3/applications/
```

Response

```
{
  "value": [
    {
      "applicationFamily": string, // The name of the family for a given Business Central offered
application. (Typically this will just be "BusinessCentral")
      "countriesringDetails": {
          "countryCode": string, // Code for a country that the application family supports creating
environments within.
          "rings": [{ // A list of logical ring groupings where environments can be created
            "name": string, // The API name of the ring (for example, PROD, PREVIEW)
            "productionRing": bool, // Indicates that the ring is a production ring. Currently there should
only be one production ring within a country.
            "friendlyName": string, // The display friendly name of the ring
          }]
      }
    }
  ]
}
```

Ring details with Versions

Gets a list of the currently available Versions that an environment can be created on within a logical ring group.

```
GET /admin/v2.3/applications/{applicationFamily}/Countries/{countryCode}/Rings/{ringName}
```

Route Parameters

`applicationFamily` - Family of the ring's application. (for example, "BusinessCentral")

`countryCode` - Code for the ring's country.

`ringName` - Name of the ring to inspect.

Response

```
{
  "value": [ // A list of the available application versions within the ring that environments can be
created on
    "<version string>",
    "<version string>",
    ...
  ]
}
```

Environment Settings

Allows you to manage environment specific settings such as the environment's AppInsights key or the update window. That is, the timeframe is considered 'ok' for updates (and thus downtime) to occur.

Get Update Settings

Returns the update settings for the environment.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/settings/upgrade
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example "BusinessCentral")

`environmentName` - Name of the targeted environment

Response

Returns the environment's update settings, or "null" if none exist

```
{
  "preferredStartTimeUtc": datetime, // Specifies the start of an environment's update window.
  "preferredEndTimeUtc": datetime, // Specifies the end of environment's update window.
}
```

> **NOTE**
> The `date` components of the values are ignored, only the time components are used.

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

Put Update Settings

Sets the update window start and end times.

```
Content-Type: application/json
PUT /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/settings/upgrade
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example "BusinessCentral")

`environmentName` - Name of the targeted environment

Body

```
{
  "preferredStartTimeUtc": datetime, // Start of environment update window
  "preferredEndTimeUtc": datetime, // End of environment update window
}
```

Response

Returns the updated settings

```
{
  "preferredStartTimeUtc": datetime, // Start of environment update window
  "preferredEndTimeUtc": datetime, // End of environment update window
}
```

> **NOTE**
>
> The `date` components of the values are ignored, only the time components are used.

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

- **invalidRange**: - the targeted logical range is invalid in some way

- target: "Preferred Upgrade Window" - the specified window is too small

`requestBodyRequired` - the request body must be provided

Put AppInsights key

Sets the key an environment uses for Azure AppInsights.

> **IMPORTANT**
>
> This process requires a restart to the environment, which is triggered automatically when you call this API. Plan to do this during non-working hours to avoid disruptions.

```
Content-Type: application/json
POST /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/settings/appinsightskey
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Body

```
{
  "key": string, // The Application Insights key for the environment
}
```

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

`requestBodyRequired` - the request body must be provided

`cannotSetAppInsightsKey` - the targeted environment's status isn't 'Active'

Telemetry

Telemetry includes the top-level AL events and any returned errors logged from the service. These events can provide necessary information and errors that can be used to troubleshoot issues happening in the tenant's environment.

Get Environment Telemetry

Returns the telemetry information for the provided environment and filters. it's recommended that you provide start and end time parameters to return a manageable data set.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/telemetry?startDateUtc=
{start}&endDateUtc={end}&logCategory={cat}
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Query parameters

`start` : datetime // The start of the telemetry entry time window to query
`end` : datetime // The end of the telemetry entry time window to query
`cat` : (All or 0) // Category of telemetry to query

Response

Returns the telemetry logs and with data column headers.

```
{
  "queryColumns": [
    {
      "name": string, // Column display name
      "ordinal": int, // Index of the column in the data set
      "dataType": ( string or 0 | numeric or 1 | datetime or 2 )
      "expectations": (none or 0 | wide or 1) // Flags enum value
    }
  ],
  "queryResults": object[][] // Raw query data results
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

`invalidInput` - the targeted property in invalid in some way

- target: {logCategory} - the provided log category isn't a valid value

Notifications

Notifications are sent to the recipient email addresses set up for the tenant. For example, notifications are sent for update availability, successful updates, update failures, and extension validations.

Get Notification Recipients

Returns a list of notification recipients.

```
GET /admin/v2.3/settings/notification/recipients
```

Response

Returns a wrapped array of recipients.

```
{
  "value":
  [
    {
      "id": GUID, // Unique identifier of the recipient
      "email": string, // Email address of the recipient
      "name": string // Full name of the recipient
    }
  ]
}
```

Expected Error Codes

`tenantNotFound` - the calling tenant information couldn't be found

Create Notification Recipient

Create a new notification recipient.

```
Content-Type: application/json
PUT /admin/v2.3/settings/notification/recipients
```

Body

```
{
  "email": string, // Email address of the recipient
  "name": string // Full name of the recipient
}
```

Response

Returns the newly created recipient.

```
{
  "id": GUID, // Unique identifier of the recipient
  "email": string, // Email address of the recipient
  "name": string // Full name of the recipient
}
```

Expected Error Codes

`invalidInput` - the targeted property in invalid in some way

- target: {name} - the name property can't be null or whitespace
- target: {email} - the email property can't be null or whitespace

`requestBodyRequired` - the request body must be provided

`tenantNotFound` - the calling tenant information couldn't be found

Delete Notification Recipient

Deletes an existing notification recipient.

```
DELETE /admin/v2.3/settings/notification/recipients/{id}
```

Route Parameters

`id` - The unique identifier of the notification recipient to delete.

Expected Error Codes

`invalidInput` - the targeted property in invalid in some way

- target: {id} - provided id can't be the empty guid

`tenantNotFound` - the calling tenant information couldn't be found

Get Notification Settings

Returns the full set of notification settings including the list of recipients.

```
GET /admin/v2.3/settings/notification
```

Response

Returns the notification settings.

```
{
  "aadTenantId": GUID, // AAD Tenant ID of the caller
  "recipients": [
    {
      "id": GUID, // Unique identifier of the recipient
      "email": string, // Email address of the recipient
      "name": string // Full name of the recipient
    }
  ]
}
```

Expected Error Codes

`tenantNotFound` - the calling tenant information couldn't be found

Reschedule Updates

Allows for the management of scheduled updates such as rescheduling the update to a run on or after a specific date within a provided range.

Get Scheduled Update

Get information about updates that have already been scheduled for a specific environment.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/upgrade
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Response

Returns information about the scheduled update for that environment.

```
{
  "environmentName": string, // The name of the targeted environment.
  "applicationFamily": string, // Family of the environment. (for example, "BusinessCentral")
  "sourceVersion": string, // The current version of the environment's application.
  "targetVersion": string, // The version of the application that the environment will update to.
  "canTenantSelectDate": boolean, // Indicates if a new update date can be selected.
  "didTenantSelectDate": boolean, // Indicates if the tenant has selected the current date for the update.
  "earliestSelectableUpgradeDate": datetime, // Specifies the earliest date that can be chosen for the
update.
  "latestSelectableUpgradeDate": datetime, // Specifies the latest date that can be chosen for the update.
  "upgradeDate": datetime, // The currently selected scheduled date of the update.
  "updateStatus": string, // The current status of the environment's update. (enum | "Scheduled" or
"Running")
  "ignoreUpgradeWindow": boolean, // Indicates if the environment's update window will be ignored
  "isActive": boolean, // Indicates if the update is activated and is scheduled to occur.
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

Reschedule Update

Reschedule an update, if able.

```
Content-Type: application/json
PUT /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/upgrade
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Body

```
{
  "runOn": datetime, // Sets the date that the upgrade should be run on.
  "ignoreUpgradeWindow": boolean // Specifies if the upgrade window for the environment should be respected.
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

`requestBodyRequired` - the request body must be provided

`resourceDoesNotExist` - no upgrade is currently scheduled for the targeted environment

`entityValidationFailed` - some unhandled error occurred in the validation of the request

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

Support Settings

Allows for the management of support settings, such as changing the contact, for a specific environment

Get Support Contact

Get information about the support contact for a specified environment.

```
GET /admin/v2.3/support/applications/{applicationFamily}/environments/{environmentName}/supportcontact
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Response

Returns information about the support contact for that environment.

```
{
  "name": string, // The name of the support contact.
  "email": string, // The email address of the support contact.
  "url": string, // A freeform url for additional support contact information such as a support website.
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

`resourceDoesNotExist` - couldn't find the necessary information to communicate with the targeted environment's API

`businessCentralCommunicationException` - an unhandled error occurred when communicating with the targeted environment's API

Set Support Contact

Sets the support contact information for a specified environment

```
Content-Type: application/json
PUT /admin/v2.3/support/applications/{applicationFamily}/environments/{environmentName}/supportcontact
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Body

```
{
  "name": string, // The name of the support contact.
  "email": string, // The email address of the support contact.
  "url": string, // A freeform url for additional support contact information such as a support website.
}
```

Response

Returns the newly updated support contact information.

```
{
  "name": string, // The name of the support contact.
  "email": string, // The email address of the support contact.
  "url": string // A freeform url for additional support contact information such as a support website.
}
```

Expected Error Codes

`applicationTypeDoesNotExist` - the provided value for the application family wasn't found

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

`requestBodyRequired` - the request body must be provided

`resourceDoesNotExist` - couldn't find the necessary information to communicate with the targeted environment's API

`businessCentralCommunicationException` - an unhandled error occurred when communicating with the targeted environment's API

Environment Outage Reporting

Enables the ability to report that an environment isn't accessible and may require attention

Get Outage Types

Gets the list of supported categories of outages

```
GET /admin/v2.3/support/outageTypes
```

Response

Returns a list with information about the supported outage types for reporting

```
{
  "value": [
    {
      "outageType": string, // The identifier of the outage type.
      "name": string, // A displayable name for the outage type.
      "description": string, // A displayable description for the outage type.
    }]
}
```

Expected Error Codes

`cannotGetOutages` - an unhandled error occurred when trying to acquire the outage types

`tenantNotFound` - the calling tenant information couldn't be found

Get Outage Questions

Gets the list of metadata about questions that need to be answered when reporting an environment outage

```
GET /admin/v2.3/support/outageTypes/{outageType}/outageQuestions
```

Response

Returns the list of question metadata for the provided outage type

```
{
  "value": [
    {
      "sequence": int, // The order in which the question should be answered
      "parentId": int, // The identifier of a toggle question whose value indicates if this question should
be shown. that is if the parent value is 'true' then this question should also be answered
      "id": string, // The unique identifier of the question
      "defaultValue": string, // The default value of the question if it has no value
      "questionType": string, // (enum | "None", "Toggle", "TextField", "DateTime")
      "questionText": string, // The question's text to display
      "required": bool, // Indicates if the question must have a value
      "onText": string, // Toggle type only - display text for when the question is toggled to 'true'
      "offText": string, // Toggle type only - display text for when the question is toggled to 'false'
      "multiline": bool // Indicates if the value is intended to contain multi-line text
    }]
}
```

Expected Error Codes

`cannotGetOutageQuestions` - an unhandled error occurred when trying to acquire the outage types

`tenantNotFound` - the calling tenant information couldn't be found

Get Reported Outages

Gets the list of outages that have been previously reported

```
GET /admin/v2.3/support/reportedoutages
```

Response

Returns the list of outages reported across all environments for the calling tenant

```
{
  "value": [
    {
      "TenantId": string, // The id of the tenant who reported the outage.
      "EnvironmentId": string, // The id of the environment that was reported to have been unavailable.
      "CreatedDate": dateTime, // The date that the report was created.
      "State": string, // The current state of the outage report.
      "MsaasTicketId": string, // The identifier of the MSaaS ticket that is associated with the outage
report.
      "IcmTicketId": string, // The identifier of the IcM ticket that is associated with the outage report.
      "AppVersion": string, // The version of the environment's application at the time of the outage
report.
      "PlatformVersion": string, // The version of the environment's platform at the time of the outage
report.
      "OutageType": string, // The category of the reported outage.
    }]
}
```

Expected Error Codes

`cannotGetReportedOutages` - an unhandled error occurred when trying to acquire the reported outages

`tenantNotFound` - the calling tenant information couldn't be found

Report Outage

Initiates an outage report indicating that an environment isn't accessible

```
Content-Type: application/json
POST /admin/v2.3/support/applications/{applicationFamily}/environments/{environmentName}/reportoutage
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Body

```
{
  "outageType": string, // The category of the outage being reported.
  "outageQuestionAnswers": [{ // The collection of answers to the questions associated with the outage type.
    "id": string, // The identifier of the question being answered.
    "answer": string // The answered value of the question.
  }],
  ("contact": string), // (Optional) - The name of the person whose to contact with updates on the outage
report
  ("email": string), // (Optional) - An email to contact with updates on the outage report
  ("phone": string), // (Optional) - A phone number to contact with updates on the outage report
  ("appVersion": string), // (Optional) - If known, the version of the targeted environment's application
  ("platformVersion": string) //(Optional) - If known, the version of the targeted environment's platform
}
```

Response

Returns information about the created outage report

```
{
    "creationStatus": string, // The status of the request to create an outage report. (enum | "Unknown",
"Created", "UpdatedExisting", "Error")
    "msaasCaseNumber": string, // The identifier of the MSaaS ticket that is associated with the outage
report.
}
```

Expected Error Codes

`requestBodyRequired` - the request body must be provided

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

`failedToReportOutage` - an unhandled error occurred when trying to report the outage

Environment Database Export

Allows for the export of an environment's Azure database. Databases are exported to an Azure Storage account provided by you. There is a limit to the number of exports that can be done within a month as shown by the 'metrics' endpoint below.

Get Export Metrics

Gets information about the number of exports allowed per month and the amount remaining.

```
GET /admin/v2.3/exports/applications/{applicationFamily}/environments/{environmentName}/metrics
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Response

Returns the metrics around the current month's database exports.

```
{
  "exportsPerMonth": int, // The total number of allowed exports of the targeted environment every month.
  "exportsRemainingThisMonth": int, // The number of exports remaining for the targeted environment that can
be performed this month.
}
```

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

Start Environment Database Export

Starts the export of an environment's database to a provided Azure storage account

```
Content-Type: application/json
POST /admin/v2.3/exports/applications/{applicationFamily}/environments/{environmentName}
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment

Body

```
{
  "storageAccountSasUri": string, // An Azure SAS uri pointing at the Azure storage account where the
database will be exported to. The uri must have (Read | Write | Create | Delete) permissions
  "container": string, // The name of the container that will be created by the process to store the
exported database.
  "blob": string, // The name of the blob within the container that the database will be exported to.
Databases are exported in the .bacpac format so a filename ending with the '.bacpac' suffix is typical.
}
```

Expected Error Codes

`environmentNotFound` - the targeted environment couldn't be found

- target: {applicationFamily}/{environmentName}

`requestBodyRequired` - the request body must be provided

`exportFailed` - the export failed because the target environment's version was too old, it wasn't a production environment, the requesting tenant is a trial, the calling user doesn't have permissions to export, or the quota of allowed exports has been used up

Get Export History

Gets information about the exports that have been done within a provided time frame, for which environment, and by whom.

```
POST /admin/v2.3/exports/history?start={startTime}&end={endTime}
```

Query parameters

`startTime` - datetime // The start of the export history entry time window to query
`endTime` - datetime // The end of the export history entry time window to query

Response

Returns a detailed list of the database exports that occurred within the provided timeframe of the `start` and `end` query parameters

```
{
  "applicationType": string, // Family of the environment. (for example, "BusinessCentral")
  "applicationVersion": string, // The version of the environment's application at the time of the export.
  "blob": string, // The name of the blob that the environment's database was exported to.
  "container": string, // The name of the Azure storage account container that the environment's database
was exported within.
  "country": string, // The country code of the environment.
  "environmentName": string, // The name of the environment that was exported.
  "storageAccount": string, // The name of the Azure storage account that the environent's database was
exported to.
  "time": dateTime, // The time that the environment's database was exported.
  "user": string // The user who initiated the export process.
}
```

NOTE

All datetime values are in UTC

Case-Invariant blocked environment names

All environment types

- invoicing
- api
- error
- navwinclient
- clickonce
- tablet
- phone
- reset
- getapp
- signout
- addremotehost
- deployment
- health
- home
- notsupported
- officeaddin
- remotesignin
- shell service
- admin

Production Environment Types

- sandbox

Sandbox Environment Types

- production

App Management

INTRODUCED IN: Business Central 2020 release wave 1

Manage the apps that are installed on the environment.

Get Installed Apps

Get information about apps that are installed on the environment.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/apps
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment.

Response

Returns information about the apps installed on the environment.

```
{
    "value":
    [
        {
            "id": Guid, // Id of the installed app
            "name": string, // Name of the installed app
            "publisher": string, // Publisher of the installed app
            "version": string, // Version of the installed app
            "state": string, // (enum | "Installed", "UpdatePending", "Updating")
            "lastOperationId": Guid, // Id of the last update operation that was performed for this app
            "lastUpdateAttemptResult": string // (enum | "Failed", "Succeeded", "Canceled", "Skipped")
        }
    ]
}
```

Get Available App Updates

Get information about new app versions that are available for apps currently installed on the environment.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/apps/availableUpdates
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment.

Response

```
{
    "value":
    [
        {
            "id": Guid, // Id of the app
            "name": string, // Name of the app
            "publisher": string, // Publisher of the app
            "version": string, // New version available of the app
            "requirements": // List of other apps that need to be installed or updated before this app can
be updated
            [
                {
                    "id": Guid, // Id of the app
                    "name": string, // Name of the app
                    "publisher": string, // Publisher of the app
                    "version": string, // Version the required app needs to be updated to or installed
                    "type": string // (enum | "Install", "Update")
                }
            ]
        }
    ]
}
```

Schedule an App Update

Schedules the installation of an app update version. The update will be installed as soon as a time slot is available.

```
Content-Type: application/json
POST /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/apps/{appId}/update
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment.

`appId` - Id of the targeted app.

Body

```
{
    "targetVersion": string // Version the installed app should be updated to
    "allowPreviewVersion": bool // Indicates whether to allow updating to an app version that is marked as a
Preview by the ISV (exact version must be specified in the targetVersion)
}
```

Response (app operation)

Returns information about the scheduled app update request.

```
{
    "id": Guid, // Id of the operation used for tracking the update request
    "createdOn": string, // Date and time the request was created
    "status": string, // (enum | "Scheduled", "Running", "Succeeded", "Failed", "Canceled", "Skipped")
    "targetVersion": string, // Version the installed app will be updated to
    "type": string // The type of app operation
}
```

Get App Operations

Gets information about app update operations for the specified app.

```
GET
/admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/apps/{appId}/operations/[{operat
ionId}]
```

Route Parameters

`applicationFamily` - Family of the environment's application as is. (for example, "BusinessCentral")

`environmentName` - Name of the targeted environment.

`appId` - Id of the targeted app.

`operationId` - Id of the app update operation. Used for getting information about a specific operation.

Response

Returns the list of app update operations for the specified app. *Note*: `operationId` is provided, the single operation is returned instead.

```
{
    "value":
    [
        {
            "id": Guid,  // Id of the operation
            "createdOn": string, // Date and time the request was created
            "startedOn": string, // Date and time the installation process started
            "completedOn": string, // Date and time the installation process completed
            "status": string, // (enum | "Scheduled", "Running", "Succeeded", "Failed", "Canceled",
"Skipped")
            "sourceVersion": string, // Version of the app that was installed before the installation
process started
            "targetVersion": string, // Version the installed app will be updated to
            "type": string, // The type of app operation
            "errorMessage": string // The error message provided when update installation fails
        }
    ]
}
```

Session Management

INTRODUCED IN: Business Central 2020 release wave 1

Manage the active sessions on an environment.

Get active sessions

Gets active sessions for an environment.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/sessions
```

Response

```
{
    value: [
        {
            environmentName: string,
            applicationFamily: string,
            sessionId: int,
            userId: string,
            clientType: string,
            logOnDate: string,
            entryPointOperation: string,
            entryPointObjectName: string,
            entryPointObjectId: string,
            entryPointObjectType: string,
            currentObjectName: string,
            currentObjectId: int,
            currentObjectType: string,
            currentOperationDuration: long
        }
        ,...
    ]
}
```

Get session details

Gets session information for a specific session id.

```
GET /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/sessions/{sessionId}
```

Response

```
{
    environmentName: string,
    applicationFamily: string,
    sessionId: int,
    userId: string,
    clientType: string,
    logOnDate: string,
    entryPointOperation: string,
    entryPointObjectName: string,
    entryPointObjectId: string,
    entryPointObjectType: string,
    currentObjectName: string,
    currentObjectId: int,
    currentObjectType: string,
    currentOperationDuration: long
}
```

Stop and delete a session

Terminates and deletes an active session.

```
DELETE /admin/v2.3/applications/{applicationFamily}/environments/{environmentName}/sessions/{sessionId}
```

See Also

Manage Apps
Microsoft Dynamics 365 Business Central Server Administration Tool

Introduction to automation APIs

2/17/2021 • 4 minutes to read • Edit Online

Automation APIs provide capability for automating company setup through APIs. Once the tenants have been created, the automation APIs can be used, in order to hydrate the tenant - to bring the tenant up to a desired state. Usually this involves creating a new company on the tenant, running RapidStart packages, installing extensions, adding users to user groups and assigning permission sets to users.

Delegated admin credentials and Dynamics 365 Business Central users with permissions, can call the APIs.

> **IMPORTANT**
>
> For delegated admin access, you must add the Azure Active Directory (Azure AD) application to the **AdminAgents** group. If the Azure AD application is not added, the consent flow will show an error such as *Need pre-consent*. For more information, see Pre-consent your app for all your customers in the Graph documentation.

Automation APIs are placed in the `microsoft/automation` API namespace. In all the examples below, parameters are marked in parenthesis `{}`. Make sure that only valid parameters are passed.

Create a company

To create a company, an automationCompany endpoint is available. To create a Company issue a POST request as shown in the following example.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/automationCompanies
Authorization: Bearer {token}
Content-type: application/json
{
    "name": "CRONUS",
    "displayName": "CRONUS",
    "businessProfileId": ""
}
```

The {companyId} must be the ID of an valid company on the tenant. Issue a GET automationCompany request to fetch existing companies.

> **NOTE**
>
> The company which is created will not be initialized.

To rename a company, issue a PATCH automationCompanies.

Upload and apply a RapidStart package

RapidStart is uploaded, installed, and applied using the APIs described below. RapidStart operations can be time consuming. To get the current status of the RapidStart packages and running operations issue a GET configurationPackages as shown in the following example.

```
GET https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/configurationPackages

Authorization: Bearer {token}
```

In the response, status for the import and apply status will be shown, as well as information about the RapidStart package.

Insert RapidStart

First step is to create the configuration package, by issuing a POST configurationPackages in the Dynamics 365 Business Central tenant. Once the configuration package is created, the RapidStart package can be uploaded. See the example below.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/configurationPackages

Authorization: Bearer {token}
Content-type: application/json
{
    "code":"{SAMPLE}",
    "packageName": "{SAMPLE}"
}
```

Upload RapidStart package

Once the configuration package is created, a RapidStart package can be uploaded with a PATCH configurationPackages. See the example below.

```
PATCH https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/configurationPackages({Id})/file('{SAMPLE}')/conte
nt

Authorization: Bearer {token}
Content-type: application/octet-stream
If-Match: *
Body: RapidStart file.
```

Import and apply RapidStart package

Once uploaded, the RapidStart package needs to be imported by issuing a POST on the bound action Microsoft.NAV.import.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/configurationPackages({Id})/Microsoft.NAV.import

Authorization: Bearer {token}
```

When the RapidStart package is imported it can applied with a POST on bound action Microsoft.NAV.apply.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/configurationPackages({Id})/Microsoft.NAV.apply

Authorization: Bearer {token}
```

Managing users, user groups, and permission sets

The automation APIs enable users to be set up in Dynamics 365 Business Central.

Modifying user properties

Get the current user properties by issuing a GET users. This will get the UserSecurityId needed on subsequent requests.

To modify the user, create a PATCH user request as shown in the example below.

```
PATCH https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/v1.0/companies({id})/users({userSecurityId})
Content-type: application/json
If-Match:*
{
    "state": "Enabled",
    "expiryDate": "2035-01-01T21:03:53.444Z"
}
```

Assign user permissions and user groups

To assign users to a user group, issue a POST request against the **userGroupMembers** entity. See the example below.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/users({userSecurityId})/userGroupMembers

Authorization: Bearer {token}
{
    "code": "D365 EXT. ACCOUNTANT",
    "companyName" :"CRONUS USA, Inc."
}
```

To retrieve the list of user groups issue a GET userGroups. This will return the information that you need for the payload above.

Assigning permission sets is identical to adding users to user groups. GET permissionSet returns information about the available permission sets. To assign a permissionSet issue a POST userPermission as shown in the following example.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/users({userSecurityId})/userPermissions

Authorization: Bearer {token}
{
    "id": "SECURITY"
}
```

Removing the permissionSet from the user is done by issuing a DELETE userPermissions on the users entity.

Handling tenant extensions

Add-on extensions which are already published to the tenant can be installed and uninstalled. Per-tenant extensions can be uploaded and installed. To get the list of all extensions on the tenant, issue a GET extensions. This will return the packageId needed for installing and uninstalling extensions.

Installing and uninstalling published add-on extensions

There are two bound actions available on the **extensions** endpoint: `Microsoft.NAV.install` and `Microsoft.NAV.uninstall`.

Issue a POST extension using the bound actions. See the example below.

```
POST https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/2.0/companies({companyId})/extensions({packageId})/Microsoft.NAV.install

Authorization: Bearer {token}
```

Upload and install a per-tenant extension

Issue a PATCH against the **extensionUpload** endpoint to upload and install the extension.

> **NOTE**
>
> Installing per-tenant extensions using Automation APIs is only possible in SaaS.

Uninstalling the extension can be done through the bound action Microsoft.NAV.uninstall, as with the add-on extensions.

Monitoring extension installation progress

To view ongoing extension installation status, issue GET extensionDeploymentStatus as shown in the following example.

```
GET https://api.businesscentral.dynamics.com/v2.0/{environment
name}/api/microsoft/automation/v1.0/companies({companyId})/extensionDeploymentStatus
```

See Also

Automation API overview

Automation Overview

2/17/2021 • 2 minutes to read • Edit Online

The following table provides an overview of the available resource types on a Dynamics 365 Business Central tenant. For an overview of how to use the automation APIs, see Introduction to Automation APIs.

RESOURCE TYPE	DESCRIPTION
automationCompany resource type	Represents an automationCompany resource type in Dynamics 365 Business Central.
company resource type	Represents a company resource type in Dynamics 365 Business Central.
configurationPackages resource type	Represents a configurationPackage resource type in Dynamics 365 Business Central.
extension resource type	Represents an extension resource type in Dynamics 365 Business Central.
extensionDeploymentStatus resource type	Represents a extensionDeploymentStatus resource type in Dynamics 365 Business Central.
permissionSet resource type	Represents a permissionSet resource type in Dynamics 365 Business Central.
user resource type	Represents a user resource type in Dynamics 365 Business Central.
userGroup resource type	Represents a userGroup resource type in Dynamics 365 Business Central.
userGroupMember resource type	Represents a userGroupMember resource type in Dynamics 365 Business Central.
userPermission resource type	Represents a userPermissions resource type in Dynamics 365 Business Central.

See Also

Introduction to Automation APIs

Using Service-to-Service Authentication with Automation APIs

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later (Online)

Starting with Business Central 2020 release wave 2, version 17, service-to-service authentication is enabled for Automation APIs. Service-to-service authentication enables external services to connect as an application, without impersonating normal users.

Usage and setup overview

Automation APIs provide capability for automating company setup through APIs. The automation APIs are used to hydrate tenants, that is, to bring them to an initial state. Service-to-service authentication is intended only for the hydration of companies.

The **D365 Automation** entitlements give access to APIs in the `/api/microsoft/automation` route by using the OAuth client credentials flow. An application token with the `Automation.ReadWrite.All` scope is needed for accessing Business Central Automation APIs.

To enable service-to-service authentication, you'll have to do two things:

- Register an application in your Azure Active Directory tenant for authenticating API calls against Business Central.
- Set up the Azure AD application in Business Central.

These tasks are described in the sections that follow.

Task 1: Register an Azure AD application for authentication to Business Central

Complete these steps to register an application in your Azure AD tenant for service-to-service authentication.

1. Sign in to the Azure portal.
2. Register an application for Business Central in Azure Active Directory tenant.

   Follow the general guidelines at Register your application with your Azure Active Directory tenant.

   When you add an application to an Azure AD tenant, you must specify the following information:

SETTING	DESCRIPTION
Name	Specify a unique name for your application.
Supported account types	Select either **Accounts in this organizational directory only (Microsoft only - Single tenant)** or **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**.
Redirect URI	Leave this option empty.

When completed, an **Overview** displays in the portal for the new application.

3. Create a client secret for the registered application as follows:

   a. Select **Certificates & secrets** > **New client secret**.

   b. Add a description, select a duration, and select **Add**.

   For the latest guidelines about adding client secrets in Azure AD, see Add credentials in the Azure documentation.

4. Grant the registered application **Automation.ReadWrite.All** permission to the **Dynamics 365 Business Central** API as follows:

   a. Select **API permissions** > **Add a permission** > **Microsoft APIs**.

   b. Select **Dynamics 365 Business Central**.

   c. Select **Application permissions**, select **Automation.ReadWrite.All**, then select **Add permissions**.

   When completed, the **API permissions** page will include the following entry:

API / PERMISSION NAME	TYPE	DESCRIPTION
Dynamics 365 Business Central / Automation.ReadWrite.All	Application	Full access to automation

   For the latest guidelines about adding permissions in Azure AD, see Add permissions to access your APIs in the Azure documentation.

Task 2: Set up the Azure AD application in Business Central

Complete these steps to set up the Azure AD application for service-to-service authentication in Business Central.

1. In the Business Central client, search for **AAD Applications** and open the page.
2. Select **New**.

   The **AAD Application Card** opens.

3. In the **Client ID** field, enter the **Application (Client) ID** for the registered application in Azure AD from task 1.
4. Fill in the **Description** field.
5. Set the **State** to **Enabled**.
6. Assign permissions to objects as needed.

   For more information, Assign Permissions to Users and Groups.

   > **TIP**
   > The system permission set and user group called **Dynamics 365 Automation** provide access to most typical objects used with automation.

7. Select **Grant Consent** and follow the wizard to the complete the setup.

TIP

Pre-consent can be done by adding the AAD application to the **Adminagents** group in the partner tenant. For more information, see Pre-consent your app for all your customers in the Graph documentation.

See Also

Automation API overview

Trials and Sign-ups for Business Central Online

2/17/2021 • 4 minutes to read • Edit Online

You can invite customers and prospects to sign up for any number of Dynamics 365 apps, including Business Central online and partner solutions based on Business Central, using the same account ID. These apps will run side-by-side with each other, will use different URLs, and will be displayed as separate tiles on the *home.dynamics.com* portal.

This means that you can show prospects a preview of what you are offering based on trials of Business Central and other Dynamics 365 apps.

Prospects and trials

Prospects can contact you if they have signed up for the Business Central free trial. This means that you can become their CSP partner. For more information, see Connect with customers.

Alternatively, you can create a more tailored trial environment based on the Business Central content pack on the cdx.transform.microsoft.com site. For more information, see Preparing Demonstration Environments of Dynamics 365 Business Central.

Some types of trials can be extended beyond 30 days. The following table outlines the types of trials that you can offer prospects:

SCENARIO	TRIAL PERIOD	CAN BE EXTENDED	CAN BE USED FOR PRODUCTION
Prospect signs up for the free trial with their work or school email account, and then switches to My Company for a 30 day trial. Their trial is based on the **Dynamics 365 Business Central for IWs** license.	30 days	Yes Once through the in-product notification + once by the partner	Yes
Partner assigns the **Dynamics 365 Business Central Premium Trial** license to an existing client in Partner Center so that they can try out the Business Central full experience in My Company	30 days	No The partner can assign a new **Dynamics 365 Business Central Premium Trial** license for a second 30 days trial	Yes
Partner creates a test customer and assigns the **Dynamics 365 Business Central Premium Trial** license to that in Partner Center. Then they invite a client or prospect to that trial.	30 days	No The partner can assign a new **Dynamics 365 Business Central Premium Trial** license for a second 30 days trial	Yes

At any point during their trial experience, when they are ready, you can convert their trial to a paid subscription using the Partner Center dashboard. For more information, see Converting trials to paid subscriptions.

The Dynamics 365 Business Central Premium Trial

In the Partner Center, you can find a special license type called the **Dynamics 365 Business Central Premium Trial** license, which is a very different way to give a prospect or an existing customer a trial experience using their own data. If you assigned the **Dynamics 365 Business Central Premium Trial** license to a customer's account in the Partner Center, then that also expires after 30 days. You cannot extend the Premium trial, but you can add one more Premium trial license to give the customer an extra 30 days of trial. But when the second Premium trial expires, then the customer must either convert their trial to a paid subscription, or they must sign up for the viral trial as described above.

For more information, see Offer your customers trials of Microsoft products in the Partner Center documentation. If you have technical difficulties assigning this license, contact Partner Center support. For more information, see Report problems with Partner Center.

Caution

Make sure you understand the limitations of this type of trial, before you offer it to a prospect or customer. It is easy to convert this type of trial to a paid subscription, but if the prospect needs more than 30 days to decide, or if they want to add more than 25 users, then the viral trial is probably a better fit for them.

Extending trials

An organization can sign up for a free trial of Business Central. When they first sign up for Business Central, they get access to an evaluation version that does not include all capabilities in Business Central. They can then switch to the 30 day trial experience to enable all capabilities.

However, sometimes a 30 day trial is not enough to decide if they want to buy Business Central. In that case, they can extend their trial with an additional 30 days. For more information, see Need More Time to Decide Whether to Subscribe? in the business functionality content for Business Central.

> **NOTE**
>
> If you are a reselling partner, we recommend that you set up demo environments for prospects that need longer time to decide if they want to buy Business Central. You can also use demo environments to help customers train their employees, for example. Using the 30 days trials for training should be limited to just that short period. However, demo environments cannot be used for production. For more information, see Preparing Demonstration Environments.

If the prospect wants to extend the trial further than those 30 days, they must contact a partner. The partner can extend it another 30 days if the delegated administrator signs into the prospect's Business Central and runs the **Extend Trial Period** guide.

After those additional 30 days, the prospect must either purchase Business Central or abandon Business Central. At this point, they will have had up to 90 days with the trial experience.

> **TIP**
>
> As a reselling partner, you can suggest your prospects sign up for a trial, but you can also help set up a customized demonstration environment based on a sandbox environment or a trial environment. In both cases, you can easily add or remove functionality based on your prospects' expectations. For more information, see Preparing Demonstration Environments.

Embed Apps and signing customers up

Specifically for partners that are in the Embed App program, signing up customers works differently. For more information, see Onboarding customers and creating environments.

See Also

Get Started as a Reseller of Business Central Online
Embed App Overview

Deploying a Tenant Customization

2/17/2021 • 4 minutes to read • Edit Online

When you have finished developing and testing your tenant customization, you must deploy the extension (.app file) containing the customization to your customer's production tenant. You must be able to log into the customer's tenant as a user with permissions to the **Extension Management** page to complete the deployment.

Use the **Upload Extension** action to deploy the extension. The extension can be deployed for the current version or for the next version of the service. In most cases it is sufficient to select the current version, unless you have developed the extension specifically for the next version.

> **NOTE**
> When you deploy an .app file for the next version, the extension will be queued up to be deployed as part of the customer's tenant being upgraded to the next version. You can typically use this in a situation where you have built an upgrade of the extension to work with the next version.

The extension you are deploying could be the initial release of the customization or an upgrade to a previous version. You must use the same steps for uploading a new extension or an extension upgrade. The service will determine if the extension needs to be upgraded based on the extension's app ID and version.

Steps for deploying your .app file

1. Log into your customer's Dynamics 365 Business Central tenant.
2. Open the **Extension Management** page.
3. Choose the **Upload Extension** action.
4. Select the browse button to select the .app file to upload. Browse to and select the extension's .app file.
5. Select if you want to deploy for the current version (most common) or next version. Select the language for the deployment.
6. Choose the **Deploy** button.
7. The extension will be deployed in the background.
   To check the status of the deployment, choose **Deployment Status** and then view the status of the extension deployment. Select the row to see additional details.
   In the deployment status details there is a **Refresh** button in the actions you must press to retrieve the most recent status and details.
8. When the extension has been successfully deployed, choose the **Refresh** button to see the new extension in the list of installed extensions.

Extension uniqueness requirements

If you are developing and deploying an updated version of a previously-deployed extension, you must keep the app ID the same and increase the version number to successfully upgrade the extension. The deployment services require that uploaded extension packages be unique across all tenants based on several sets of keys:

- Package ID
- App ID + Version
- Name + Publisher + Version

> **NOTE**
>
> These parameters are defined in the app.json file of the extension. For more information, see JSON files.

If you have successfully deployed an extension to a tenant, and then recompile the extension's source code without updating version number, this generates a new extension package file with a new package ID. If you try to upload this new extension package file to a different tenant, the upload will fail with the error similar to `An extension with same App ID and version has already been uploaded. Resolve and deploy again.`. Similarly, if an extension failed to deploy, and you try to upload a new extension package with the same version number, the upload will fail as well.

When developing a per-tenant extension from the same source code as an extension for multiple tenants, we recommend that you adjust the App ID, Name, Publisher, and Version of the extension for each tenant to maintain uniqueness. You may deploy the same extension package to multiple tenants if the package ID, app ID, name, publisher, and version are all the same.

If, when creating a new sandbox environment as a copy of a production environment, you receive a message indicating that the environment creation failed due to an existing development extension, it is related to violation of uniqueness requirements for extension packages.

Typically this situation is the result of the production environment being copied having an extension package installed with the same app ID, name, publisher, and version as a development extension published to another sandbox environment within the same application service. To resolve this situation, remove the development extension by unpublishing it via the Extension Management page of the other sandbox environment once you have made sure you have a backup of the extension development files. After this is completed you can attempt to copy a production environment to a sandbox again. It is recommended that when you create the extension package for distribution, you use a different app ID than that used for your development extension, which will help you avoid this conflict between your development sandbox and your customer's production and sandbox environments.

Upgrades and per-tenant extensions

In some cases, a customer's production environment that includes a per-tenant extension cannot be upgraded automatically by Microsoft. In those cases Microsoft contacts the reselling partner and provides guidance for how to update the extension. However, if the extension remains impossible to upgrade, Microsoft will wait for 90 days and then uninstall the extension and force-upgrade the tenant.

See Also

Getting Started with AL
AL Development Environment
FAQ for Developing in AL
Using Designer

Technical Support for Dynamics 365 Business Central

2/17/2021 • 4 minutes to read • Edit Online

Each customer of Business Central has a partner who assists with technical support when requested by the internal administrator. As a Business Central reselling partner, you are an administrator of your customers' Business Central tenants, and you are the first line of support. This means that you will get requests for support from your customers that you must triage, investigate, and either resolve or escalate to Microsoft.

In this section, you can learn about the tools that are available to you to help you troubleshoot your customers' Business Central.

Configuring the support experience

Because you are the first line of support for your customers, you must make it easy for them to contact you. To that end, there is a section in the Help and Support page in your customers' Business Central tenants where they can find this information.

> **IMPORTANT**
>
> You must have set up users in your own tenant in Partner Center as either *Admin agent* or *Helpdesk agent*, and they must have *delegated administration* privileges in your customer's Business Central to support the customer. For more information, see Delegated Administrator Access to Business Central Online.

To supply your support contact information in the administration center

1. In the Business Central administration center, choose the environment that you want to specify your contact details for, such as *Production*, and then, in the **Support** menu, choose **Manage Support Contact**.
2. Fill in the **Name**, **Email address**, and the **Website** fields, so that your users know how to contact you for technical support.
3. Optionally, choose the **Apply to all environments** checkbox if you want to add the same details to all related environments for this tenant.

Your customer can now contact you if they experience problems that they cannot resolve themselves. If you also cannot resolve a reported issue, you can escalate the issue to Microsoft. For more information, see Managing Technical Support.

On-premises deployments

In on-premises deployments of Business Central, the **Help and Support** page does not contain the section for contacting technical support. Instead, you can enter an agreement with your customer's administrator about how and when to contact you.

There are two other links in the **Help and Support** page that you can customize:

- Blog

  Specifies a link to where your customers can access a blog about their solution

- Coming soon

  Specifies a link to where your customers can access a roadmap for future changes

If you choose to not modify these settings, then the links go to Microsoft's blog and release notes.

For more information, see Configuring Business Central Web Server Instances.

Getting support for extension issues

As a partner, you must identify if the issue is caused by application logic or platform behavior:

- If the issue is caused by application logic, you must identify the publisher of the extension.
- If the extension is a per-tenant extension, as a partner, you must fix the issue.
- If the extension is an AppSource extension, you must contact the AppSource partner that developed the extension.
- If the extension is published by Microsoft, you must contact Microsoft support.

Troubleshooting and support

The Business Central administration center is your primary tool to support your customers. However, you can also log in to the customer's Business Central as the delegated admin for troubleshooting.

For more information, see Managing Technical Support.

Summary of where to file bugs and issues

As a partner, you have different support channels depending on what type of issue you want support for. The following list outlines the various channels.

ISSUE TYPE	SITE
Submit support request on behalf of your Business Central online customers	Start at the Business Central administration center where you can easily submit a support request in the Power Platform admin center
Report bug in a preview or beta version	The MS Collaborate site
Collaboration on the AL language and developer experience	The AL Developer Preview GitHub repo
As an ISV, report an issue in production code, such as a problem with Microsoft's application, upgrade, or telemetry	The Partner Center - choose the **Support** section, start a partner request, search for *Business Central*, and then choose the relevant category for *Product Support > Business Central Development* and submit your support request.
Report bug in supported in-market versions of Business Central on-premises	The Support for business site

IMPORTANT

To submit support requests on behalf of your customer, you must be a delegated admin on the customer tenant. If your company is a CSP direct bill partner, you must have *Advanced* or *Premier* support plan.

TIP

When you submit your first support ticket as a partner, you must specify details about your company's support plan. If you or your colleagues do not know these details, contact your Microsoft rep.

Non-product related questions

On occasion, as a partner, you will run into questions that are not directly related to the product. The following list outlines how to get answers to those questions.

FOR QUESTIONS RELATED TO	CONTACT
Problems with listing or publishing apps to AppSource due to Commercial Marketplace issues	The Partner Center - choose the **Support** section, start a partner request, search for *Business Central*, and then choose the relevant category for *Commercial Marketplace* and submit your support request.
Licensing or PSBC agreements	Email MBS Orders or MBS Agreements, respectively
Microsoft Partner Network, Partner Center, Cloud Solution Provider program	The Partner Center Support site
Payments, credit terms, checks, wire, or similar	Email MBS Accounting
Technical issues with PSBC, PartnerSource, or Order Central	Email IT MBS Support
Incentives	Email CSA Team
Cloud Solution Provider incentives	Email Online Channel Incentives Support
CSA/Ocina escalations	Email NAOC Channel Incentives Escalations
Volume licensing	The Call Logging Tool site or email Online Licensing

See Also

Inspecting and Troubleshooting Pages
The Business Central Administration Center
Managing Technical Support
Deployment Overview
Administration of Business Central Online
Administration of Business Central On-Premises
Provide technical support (Microsoft Partner Center)
Providing support to your customers (Microsoft Partner Center)

Migrating On-Premises Data to Business Central Online

2/17/2021 • 3 minutes to read • Edit Online

Organizations that run their workloads on-premises but are looking to move to the cloud can easily migrate to Business Central online. By moving to the cloud, users get the benefits of cloud scenarios such as Machine Learning, Power BI, Power Automate, and others to drive suggested actions.

First, get started with Business Central online tenant here.

The cloud migration tool in Business Central online supports migration from specific versions of specific software. For more information, see the following articles:

- Migrate to Business Central Online from Business Central On-premises
- Migrate to Business Central Online from Dynamics GP
- Upgrading from Dynamics NAV to Business Central Online

Data migration

Data migration is the process of securely migrating data from your on-premises SQL Server instance to your Business Central online tenant. The process uses the Azure Data Factory (ADF) to migrate the data between databases directly, meaning it does not look at any permissions within the applications you are transferring data between, only SQL permissions.

> **TIP**
> Before you start the migration process, please update statistics and reorganize indexes on all tables on the source database. This will ensure that the migration runs as fast as possible. For more information, see the documentation for sp_updatestats (Transact-SQL) and Resolve index fragmentation by reorganizing or rebuilding indexes.

To migrate data, in the target company in your Business Central online tenant, run the Cloud Migration Setup assisted setup wizard.

> **IMPORTANT**
> You must be signed in as an administrator of the Business Central online tenant and the Microsoft 365 tenant.

Once the wizard is complete and data migration is activated, an initial data migration will happen at the scheduled time. Alternatively, you can trigger the data migration process manually.

> **TIP**
> You can migrate the data that you want to take with you to the cloud. But if your Business Central online storage exceeds 80 GB, some administrative tasks are disabled. We recommend that you consider reducing the amount of data that you migrate so that it is less than 30 GB in each migration run. For example, reduce the number of companies that you are migrating data for, or delete outdated data in tables that contain log entries and archived records. Also, review how you can manage database capacity in a Business Central online environment.

Data is migrated table by table, and success and failures are tracked for each table. If a table fails to migrate, the

error will be captured, and the migration moves on to the next table until completed. Tables will fail to migrate if they cannot be found, or if the schema does not match between the cloud and the on-premises tables.

The initial data migration time can vary depending factors such as the amount of data to migrate, your SQL Server configuration, and your connection speeds. The initial migration will take the longest amount of time to complete because all data is migrating. After the initial migration, only changes in data will be migrated so they should run more quickly. You do not need to run the migration process more than once if you don't want to, but if you are running the migration while users are still using the on-premises system you will need to run at least one more migration in order to ensure all data was moved to the cloud before you start transacting in the cloud tenant.

See also

Running the Cloud Migration Tool
Managing the Migration to the Cloud
FAQ about Connecting to the Intelligent Cloud from On-Premises Solutions
Migrate to Business Central Online from Business Central On-premises
Migrate to Business Central Online from Dynamics GP
Upgrading from Dynamics NAV to Business Central Online
Managing your intelligent cloud environment
Managing Capacity

Running the Cloud Migration Tool

2/17/2021 • 11 minutes to read • Edit Online

The **Set up Cloud Migration** assisted setup guide helps administrators migrate data from supported on-premises solutions to their Business Central online tenant as part of the migration to the cloud.

The cloud migration tool supports migration from specific versions of specific software. For more information, see the following articles:

- Migrate to Business Central Online from Business Central On-premises
- Migrate to Business Central Online from Dynamics GP
- Upgrading from Dynamics NAV to Business Central Online

Use the same assisted setup to set up a connection to the intelligent cloud but still remain on-premises. For the list of on-premises solutions that currently supported for connecting to the intelligent cloud, see Which products and versions are supported for connecting to the intelligent cloud? in the FAQ.

In the following sections, you're working in your Business Central online tenant and connecting it to your on-premises database. Either because you are migrating from on-premises to online, or because you are connecting to the intelligent cloud.

> **TIP**
> In migration scenarios, we recommend that you start the migration by running the assisted setup from a company other than the company that you are migrating data to. For example, you can log into the demonstration company, CRONUS, and start the process there. This way, you can make sure that all users are logged out of the original company and the target company.

Best practices

This section provides best practices and recommendations for migrating to the cloud.

> **IMPORTANT**
> You must be signed in as an administrator of the Business Central online tenant and the Microsoft 365 tenant.

- As a best practice, test this configuration in your sandbox environment before making changes to a production tenant.

  For more information, see Managing Environments.

- Any existing data in your Business Central tenant will be overwritten with data from your on-premises solution, or source, once the data migration process is run.

  If you do not want data in your Business Central online tenant to be overwritten, do not configure the connection.

- All users that do not have *SUPER* permissions will be automatically reassigned to the intelligent cloud user group. This will limit them to read-only access within the Business Central tenant. See more below.

- If your data source is Business Central on-premises, several stored procedures will be added to the SQL Server instance that you define. These stored procedures are required to migrate data from your SQL Server database to the Azure SQL server associated with your Business Central tenant.

- Consider reducing the amount of data that you migrate.

  You can migrate the data that you want to take with you to the cloud. But if your Business Central online

storage exceeds 80 GB, some administrative tasks are disabled. We recommend that you consider reducing the amount of data that you migrate so that it is less than 30 GB in each migration run. For example, reduce the number of companies that you are migrating data for, or delete outdated data in tables that contain log entries and archived records. Also, review how you can manage database capacity in a Business Central online environment.

You can specify which companies to include in the migration in the assisted setup wizard, and you can view the migration status of each company in the Cloud Migration Management page.

If you want to add more companies after the first selection of companies, you can add additional companies in the **Cloud Migration Management** page in the Business Central online tenant. For more information, see Adding a tenant to an existing runtime service, or updating companies. But use the Capacity section of the Business Central administration center to keep track of how much data you migrate.

In certain cases, the customer wants to migrate very large amounts of data. In those cases, you must first run the assisted setup once to create a pipeline, and then contact Support to increase the limitations on your Business Central online tenant. For more information, see Escalating support issues to Microsoft. We are continually working on improving and optimizing the migration tool for larger database sizes, and since 2020 release wave 2, customers can buy additional environments, for example.

- Before you set up the cloud migration, make sure that at least one user in the system has *SUPER* permissions. This is the only user that will be allowed to make changes in the Business Central tenant.
- Configuring the cloud environment will have no impact on any users or data in your on-premises solution.

To begin configuring the connection, navigate to the assisted setup page and launch the **Set up Cloud Migration** assisted setup guide.

> **TIP**
> If you are using Business Central on-premises, the same setup guide is also available in your on-premises solution. You will automatically be redirected to your Business Central online tenant to continue the configuration process.

The Set up Cloud Migration assisted setup guide

When you choose the **Set up Cloud Migration** assisted setup, it launches the **Data Migration Setup** guide, which consists of up to six pages that take you through the process of connecting your solution to the Business Central online tenant.

1. Welcome and Consent page

   This page provides an overview of what the wizard will do. You must agree to the displayed warning message before you can continue to the next step.

2. Product selection

   On this page, specify the on-premises solution that you want to replicate data from. All supported sources will appear in the list. If you don't see your product, navigate to the **Manage Extensions** page, and then verify that the intelligent cloud extension for your on-premises solution is installed.

   To set up migration from the previous version of Business Central, in the **Data Migration Setup** dialog, choose *Dynamics 365 Business Central (Previous Version)* as the product.

   > **TIP**
   > Use the migration tool to migrate from the latest version of Business Central or the previous version. If your current version is older than the previous version, then you must upgrade your on-premises solution. For more information, see Supported Upgrade Paths to Dynamics 365 Business Central Releases.

3. SQL Connection

   If the product you selected requires a SQL connection, this page will be presented. Other source applications may require different information to connect to them. This page will display the connection information based on the product that you specified in the previous page. This is defined from the installed extensions for the product you have selected.

FIELD	DESCRIPTION
SQL Connection	**SQL Server**, which is your locally installed SQL Server instance, or **Azure SQL**.

SQL Connection string	You must specify the connection string to your SQL Server, including the name of the server that SQL Server is running on, and the name of the instance, the database, and the relevant user account. For example, `Server=MyServer\BCDEMO;Database=BC170;UID=MySQLAccount;PWD=MyPassW`, if you're migrating from Business Central on-premises, version 17. For more information, see the SQL Server blog. The following snippets illustrate a couple of connection strings with different formats:
	`Server={Server Name\Instance Name};Initial Catalog={Database Name};UserID={SQL Authenticated UserName};Password={SQL Authenticated Password};`
	`Server={Server Name\Instance Name};Database={Database Name};User Id={SQL Server Authenticated UserName};Password={SQL Server Authenticated Password};`
	The SQL connection string is passed to Azure Data Factory (ADF), where it is encrypted and delivered to your Self-Hosted Integration Runtime and used to communicate with your SQL Server instance during the data migration process.

Integration runtime name	If your SQL connection is **SQL Server**, you must specify the runtime service that will be used to replicate the data from the defined source to your Business Central online tenant. The integration runtime must be running on the machine that holds the SQL Server database. If you don't already have a runtime service, leave the field empty, and then choose the **Next** button. Once you choose **Next**, a new pipeline will be created in the Azure service. This takes less than a minute to complete, in most cases. If you want to test your SQL string, open the **Microsoft Integration Runtime Configuration Manager**, and then choose the **Diagnostics** menu option. From there, you can test to see if the connection is good. If you are a hosting partner, you may have multiple tenants running on the same integration runtime service. Each tenant will be isolated in their own data pipeline. To add tenants to an existing integration runtime service, enter the name of the existing integration runtime service into this field. The integration runtime name can be found in the Microsoft Integration Runtime Manager. For more information, see Create and configure a self-hosted integration runtime.

   If you left the *Integration runtime name* field empty, a new page appears from where you can download the self-hosted integration runtime that you must install. Follow the instructions on the page.

4. Company Selection

   From the list of companies from your on-premises solution, the source of the migration, select the

companies you want to migrate data for. If the company does not exist in your Business Central online tenant, it will be automatically created for you.

> **NOTE**
> This process may take several minutes depending on the number of companies that need to be created.

5. Enable & Scheduling Migration

   The final page in the wizard allows you to enable the migration process and create a schedule for when the data migration should occur. These settings are also available within your Business Central tenant on the **Cloud Migration Management** page. You have the option to schedule migrations daily or weekly.

> **TIP**
> We recommend that you schedule your data migration for off-peak business hours since it can take many hours to run, depending on the amount of data.
>
> We also recommend that you make sure that all users are logged out of both the source company and the target company.

Once you have migrated the data that you want to migrate to Business Central online, you end the migration by disabling cloud migration in the **Cloud Migration Setup** page. This is an important step, because each time someone runs the migration, outstanding documents for vendors and customers, general ledger account numbers, inventory items, and any other changes made in the target company in Business Central online are overwritten. If you are not migrating but using the assisted setup guide to connect to the intelligent cloud, you can adjust the migration schedule.

> **NOTE**
> The amount of time the migration will take to complete depends on the amount of data, your SQL configuration, and your connection speed. Subsequent migrations will complete more quickly because only changed data is migrating.

Adding a tenant to an existing runtime service, or updating companies

There are some scenarios where it will be necessary for you to run the cloud migration setup wizard more than once.

One example is if you want to change the companies you replicate data for. If the companies in your on-premises solution have changed, either added or deleted, or you want to change the companies to migrate, run the assisted setup wizard again. Alternatively, choose the additional companies in the **Cloud Migration Management** page.

Another example is that if you are a hosting partner and want to add tenants to your existing runtime service.

In both examples, you will be making updates to an existing runtime service. When you get to the point of the wizard where you can specify an existing runtime services name, open the Microsoft Integration Runtime Service Manager and enter the runtime name in the field in the wizard; you will not be allowed to copy/paste. The runtime service will identify that you are making updates to an existing service and will not create a new one.

Complete the steps in the wizard to update the runtime service. If the change was related to adding tenants to an existing service, a new data pipeline will be created for that tenant. Changing your migration schedule or regenerating an Azure Data Factory (ADF) key may be done using the **Cloud Migration Management** page in your Business Central cloud tenant. For more information, see Managing the Migration to the Cloud.

User groups and permission sets

When running as connected with an on-premises solution, the Business Central online tenant will be read-only with few exceptions. Because the on-premises solution is your primary application for running your business, including activities such as data entry, tax reporting, and sending invoices, these tasks must be completed in the on-premises solution. We limit the amount of data that you can enter in your Business Central tenant to data that is not migrated. Otherwise any data that was written to the tenant database would be continuously overwritten during the migration process.

To make setting up this read-only tenant more efficient, we created a new *Intelligent Cloud* user group and an *Intelligent Cloud* permission set. Once the cloud migration environment is configured, all users without SUPER permissions will be automatically assigned to the *Intelligent Cloud* user group. Only users with SUPER permissions will be allowed to make modifications to the system at this point.

> **NOTE**
> Before you configure the a connection from on-premises to Business Central, make sure that at least one user in each company is assigned SUPER permissions.

Users that are reassigned to the *Intelligent Cloud* user group will have access to read ALL data by default. If you need to further restrict what data a user should be able to read, the SUPER user may create new user groups and permissions sets and assign users accordingly. It is highly recommended to create any new permissions sets from a copy of the *Intelligent Cloud* permission set and then take away permissions you do not want users to have.

> **WARNING**
> If you grant insert, modify or delete permissions to any resource in the application that was set to read-only, it could have a negative impact on the data in the Business Central cloud tenant. If this occurs, you may have to clear all your data and rerun a full migration to correct this.

Extensions

It is highly recommended that you test the impact of any extension in a sandbox environment before having it installed in your production Business Central tenant to help avoid any data failures or untended consequences.

See Also

Managing the Migration to the Cloud
Migrating On-Premises Data to Business Central Online
Migrate to Business Central Online from Business Central On-premises
Migrate to Business Central Online from Dynamics GP
Upgrading from Dynamics NAV to Business Central Online
FAQ about Connecting to Business Central Online from On-Premises Solutions
Intelligent Insights with Business Central Online

Managing the Migration to the Cloud

2/17/2021 • 5 minutes to read • Edit Online

You can connect your on-premises solution to your Business Central online tenant for the purpose of migrating your data to the cloud. Once you have set up this configuration, you can manage your cloud environment and data migration from the **Cloud Migration Management** page in Business Central online.

> **NOTE**
>
> To manage migrations and run the migration tool, you must have the SUPER permission set in Business Central and be an administrator of the Microsoft 365 tenant. .

Cloud Migration Management

The **Cloud Migration Management** page provides information about your data migration runs as well as the ability to manage your migration services, for example.

The page provides a view of the status of all migration runs. You can view the time the migration ran and the status of each migration. If you have set up a schedule, you can also see when the next migration is scheduled to run. The **Migration Information** tiles show the number of migrated tables and the number of tables that did not migrate due to warnings or errors. Choose a tile to drill into additional details and guidance to correct any errors.

There is also a tile that shows tables that are not migrated due to problems with the data. For example, tables with permissions are not migrated from on-premises solutions because permissions work differently between online and on-premises.

The following table describes the actions that you can run from the page:

ACTION	DESCRIPTION
Manage Schedule	Opens a page where you can set the migration schedule without having to run the assisted setup wizard again.
Run Migration Now	Choose this to start the data migration manually. This can be helpful if you received errors in the scheduled data migration, you corrected the errors, and you now want to push updated data to the cloud outside of a normally scheduled run. The migration can also be used for subsequent runs after the initial migration. On subsequent runs, the migration tool will only migrate changes that have happened since the previous migration was run. Change tracking is used to identify what data should be moved in those subsequent runs.
Refresh Status	If a migration run is in progress, you can choose to refresh status to update the page. If the run is complete, the status will update using the refresh status action without having to close the window and reopen it.

ACTION	DESCRIPTION
Reset Cloud Data	You may run into instances where you need to reset your cloud data. This option will clear all data in your cloud tenant and enable you to start over with data migration. Only run this process if you want to start the migration process all over from the beginning. If you need to clear data in your cloud tenant, and you have connectivity issues that persist for more than 7 days, you must contact customer support. They will create a ticket to have your tenant data cleared. This process should *only* be done if you want to start the data migration all over and bring all data from on-premises to your cloud tenant.
Get Runtime Service Key	Returns the existing runtime key.
Reset Runtime Service Key	If at any time you suspect that your Self-Hosted Integration Runtime key is no longer secure, you can choose this option to regenerate a new key. A new key will be generated for you and automatically be updated in the Self Host Integration Runtime service.
Disable Cloud Migration	Opens a guide that helps you through a checklist of instructions to disable the cloud migration configuration. Once the steps in this process are complete, you can use your Business Central online tenant as your primary solution.
Check for Update	If there have been changes to the migration service, we will publish the new service. This action will check to see if a new service has been published. The check will display the version of the service you are currently running and then also display the latest service published. You will then have the option to update your solution. We recommend that you update the solution if a newer version has been published.
Select Companies to Migrate	If your database contains more than one company, you this action to specify which company or companies to schedule a migration run for. For example, you're migrating a very large database with multiple companies, so you break down the migration in several runs by including one or a few companies in each migration run. You can see the estimated size of each company
Define User Mappings	This option is available when you log in to a particular company that has been migrated. This action should be done in one of the companies you have migrated. This action gives you a list of the users that were in your on-premises environment, and then gives you a list of your Microsoft 365 users, so that you can map the two together. This process renames the **Name** field on the **User Card** to match the user name in your on-premises solution. It is not a required step, but if you use some of the processes in Business Central that work in conjunction with the user name, such as timesheets, you may want to map users. Timesheets are visible based on the user name you are logged in as in Business Central.

ACTION	DESCRIPTION
Setup Checklist	When you are ready to use your Business Central online tenant as your main system, the tables that were not migrated must be set up or defined as needed. The checklist page shows recommended steps to complete your migration to the cloud.
Azure Data Lake	This option is available if the Business Central online tenant is connected to Dynamics GP. For more information, see Migrate Dynamics GP to Azure Data Lake.

Company initialization

When a company is created in Business Central, no one can access it until it has been initialized. If you are familiar with Dynamics NAV, then you are used to this step happening automatically during the upgrade process, for example. But it's not quite the same with Business Central online. Starting with 2020 release wave 2, when a migration run completes, you are prompted to view a list of companies that were not initialized yet so that you can start the initialization. You can choose to mark a company as already initialized, such as if it was initialized in an earlier migration run. Technically, the initialization runs as a scheduled task in the job queue, and the status is automatically updated in the list of companies when a task completes.

> **NOTE**
> When you schedule an initialization in the **Hybrid Companies** list, then you cannot make any modifications to the company until the initialization task completes.

See also

Running the Cloud Migration Tool
Migrate to Business Central Online from Business Central On-premises
Migrate to Business Central Online from Dynamics GP
Upgrading from Dynamics NAV to Business Central Online
Migrating On-Premises Data to Business Central Online
FAQ about Connecting to Business Central Online from On-Premises Solutions

Migrate to Business Central Online from Business Central On-premises

2/17/2021 • 3 minutes to read • Edit Online

Your Business Central on-premises solution can have an identical twin in a Business Central online tenant. Use this twin to migrate to the cloud, or use it to connect to intelligent cloud scenarios. The migration can be started quite easily from the assisted setup wizard in your on-premises solution.

> **NOTE**
>
> Currently, you can migrate to Business Central online from versions 14, 15, 16, and 17. Alternatively, you can upgrade to the current version and then migrate to the cloud. For more information, see Supported Upgrade Paths to Business Central Releases.

To verify that you are running on a version that supports this migration, in the Business Central administration center, open the environment that you intend to migrate your data to, and then choose the **Apps** action. Make sure that these apps have the latest updates installed:

- Intelligent Cloud Base
- Business Central Intelligent Cloud

If you are migrating from an earlier supported version, you must also make sure that the following apps are updated:

- Business Central Cloud Migration – Previous Release
- Business Central Cloud Migration – Previous Release [code for your country-specific version]

For more information, see Managing Apps.

Migrating data from extensions

When your on-premises solution is connected to the cloud, it is highly recommended that you test the impact of any extension in a sandbox environment before you install the extensions in your Business Central production tenant to help avoid any data failures or unintended consequences.

In order to support data migration, tables and table extensions must specify if data from that table must be migrated or not. By default, the **ReplicateData** property is set to *Yes* so that, by default, any extension that is installed in the Business Central cloud tenant will have all its tables migrated.

In certain circumstances, you may want to not migrate all data. Here are a few examples:

- The extension is installed in the Business Central online tenant but not in the Business Central on-premises solution

  In this case, Business Central will attempt to migrate the data but show a warning. Since the extension is not installed on-premises, any table related to that extension table will not migrate, and warning notifications will appear in the cloud migration status page.

  If you own the extension, we recommend that you set the **ReplicateData** property to *No* on the extension tables. If you do not, and if you want data to migrate, install the extension in both your Business Central cloud tenant and your on-premises solution. If you do not want data to migrate, uninstall the

extension from your Business Central cloud tenant.

- The extension references a base table

  This can cause your base table to appear empty when you view data in your Business Central cloud tenant. If that happens, uninstall the extension from your Business Central cloud tenant, and then run the cloud migration process again.

Data that is not migrated

During the data migration process, Business Central does not migrate most system tables, users, and permissions.

Upgrading to a new version of Business Central

If you upgrade to a new version of Business Central on-premises, including a cumulative update, then you must update the extensions as well. Depending on your on-premises solution, your Business Central tenant contains different extensions for the cloud migration. For more information, see Business Central Intelligent Cloud Extensions.

> **IMPORTANT**
> In your Business Central online tenant, install, publish, or upgrade the **Intelligent Cloud Base** extension first, and then the product-specific extension or extensions.

Also, at the end of the upgrade, you must make sure that the `applicationVersion` field in the `ndo$tenantdatabaseproperty` table is set to the right version. If the field is blank, or if it is set to an older version than the migration tool supports, the migration cannot run. For more information, see Post-upgrade tasks.

See also

Migration On-premises Data to Business Central online
Connect to the Intelligent Cloud from On-Premises
Managing your Intelligent Cloud Environment
ReplicateData Property
Intelligent Insights with Business Central
Migrate Legacy Help to the Dynamics 365 Business Central Format
Upgrading from Dynamics NAV to Business Central Online
Important Information and Considerations for Before Upgrading to Dynamics 365 Business Central Spring 2019

Migrate to Business Central Online from Dynamics GP

2/17/2021 • 9 minutes to read • Edit Online

An assisted setup guide in Business Central can help you migrate data from Dynamics GP. You can migrate data from Dynamics GP 2015 on SQL Server 2012 and later versions of Dynamics GP and SQL Server.

> **IMPORTANT**
>
> Migrating from Dynamics GP using the **Cloud Migration Setup** assisted setup guide is currently only supported for the following markets: United States, Canada, United Kingdom, Australia.

Migrated data

When you migrate from Dynamics GP, the following information is migrated from Dynamics GP to Business Central online:

1. Chart of Accounts master records

   The account number in Business Central will be mapped from the main account segment from Dynamics GP. Remaining account segments are then defined as dimensions in Business Central. The assisted setup guide asks the user to enter a segment for *Global Dimension 1* and *Global Dimension 2*. If your chart of accounts in Dynamics GP has more than 2 segments outside of the main segment, the other segments are automatically set up as shortcut dimensions (3-8). You can verify the setup in the **General Ledger Setup** page in Business Central.

   Account Summary transactions are generated and posted for open and history years that were set up in Dynamics GP. The summary amounts are created based on the fiscal periods set up in Dynamics GP.

   Let us look at an example of an account from Dynamics GP:

ACCOUNT IN DYNAMICS GP	YEAR	NAME	AMOUNT
000-1100-00 Cash	Year 2019		
		Period 1	250.00
		Period 2	117.00
		Period 3	340.00
000-1100-01 Cash	Year 2019		
		Period 1	240.00
		Period 2	102.00
		Period 3	501.00

ACCOUNT IN DYNAMICS GP	YEAR	NAME	AMOUNT
000-4000-00 Sales	Year 2019		
		Period 1	490.00
		Period 2	219.00
		Period 3	841.00

The migration creates two accounts in Business Central, number 1100 and number 4000. New dimensions are also added with the names 000, 00, and 01. General journal transactions are created as follows:

TRANSACTION DATE	ACCOUNT NO.	AMOUNT	DIMENSIONS
1/31/2019	1100	250.00	Dimension 000, 00
1/31/2019	1100	240.00	Dimension 000, 01
1/31/2019	4000	490.00	Dimension 000, 00
2/28/2019	1100	117.00	Dimension 000, 00
2/28/2019	1100	102.00	Dimension 000, 01
2/28/2019	4000	219.00	Dimension 000, 00
3/31/2019	1100	340.00	Dimension 000, 00
3/31/2019	1100	501.00	Dimension 000, 01
3/31/2019	4000	841.00	Dimension 000, 00

The data migration generates dimensions on that account based on the different segments. User will see a *Department* dimension with the values *000*, *100*, and *200*, respectively. A second dimension, *Division*, will show the values *00*, *01*, and *02*, respectively.

2. Customer master records and outstanding transactions from the Receivables module

   In the setup wizard, you can choose to migrate all customers from Dynamics GP or only active customers. This allows you to not migrate over customers that have been marked as inactive. We also have added bringing all addresses from the customer over into Business Central. All of the addresses on the customer will be setup as shipping addresses in Business Central. That will allow the end user to choose the address needed when entering transactions after the migration.

   We also bring over outstanding receivables transactions. These transactions will be brought in with the amount remaining in Dynamics GP. For example, if an invoice for $1000 was entered into Dynamics GP, and it has been partially paid and has a remaining balance of $400, the new invoice created in Business Central will be for $600 as that is the amount remaining to be paid. We bring over all transaction types from Receivables Management.

3. Vendor master records and outstanding transactions from the Payables module

In the setup wizard, you can choose to migrate all vendors from Dynamics GP or only active vendors. This allows you to not migrate over vendors that have been marked as inactive. We also have added bringing all addresses from the vendor over into Business Central. All of the addresses on the vendor will be setup as order addresses in Business Central. That will allow the end user to choose the address needed when entering transactions after the migration.

We also bring over outstanding Payables transactions. These transactions will be brought in with the amount remaining in Dynamics GP. For example, if an invoice for $1000 was entered into Dynamics GP, and it has been partially paid and has a remaining balance of $400, the new invoice created in Business Central will be for $600 as that is the amount remaining to be paid. We bring over all transaction types from Payables Management.

You can also bring over Open Purchase Orders. When we migrate purchase orders, we are looking at the items and the quantities remaining on those items to determine what we will bring over as an open purchase order. If an item is fully received and invoiced that item will not migrate. By bringing over open purchase orders, you do not have to enter outstanding transactions from the purchase order aspect.

4. Inventory items

   Inventory is imported with the cost valuation method that was selected when the company setup wizard was run. Currently, the data migration brings in the quantity on hand for the items at the time of migration. This quantity is brought into the blank location.

5. Historical data from Receivables, Payables, Sales Order Processing, Purchase Order Processing, and Inventory

   This data can be used in Power BI reports and Power Apps. In Business Central online, the data is included in the SmartList views in the Customers, Vendors, and Items lists. Technically, the data is stored in table extensions.

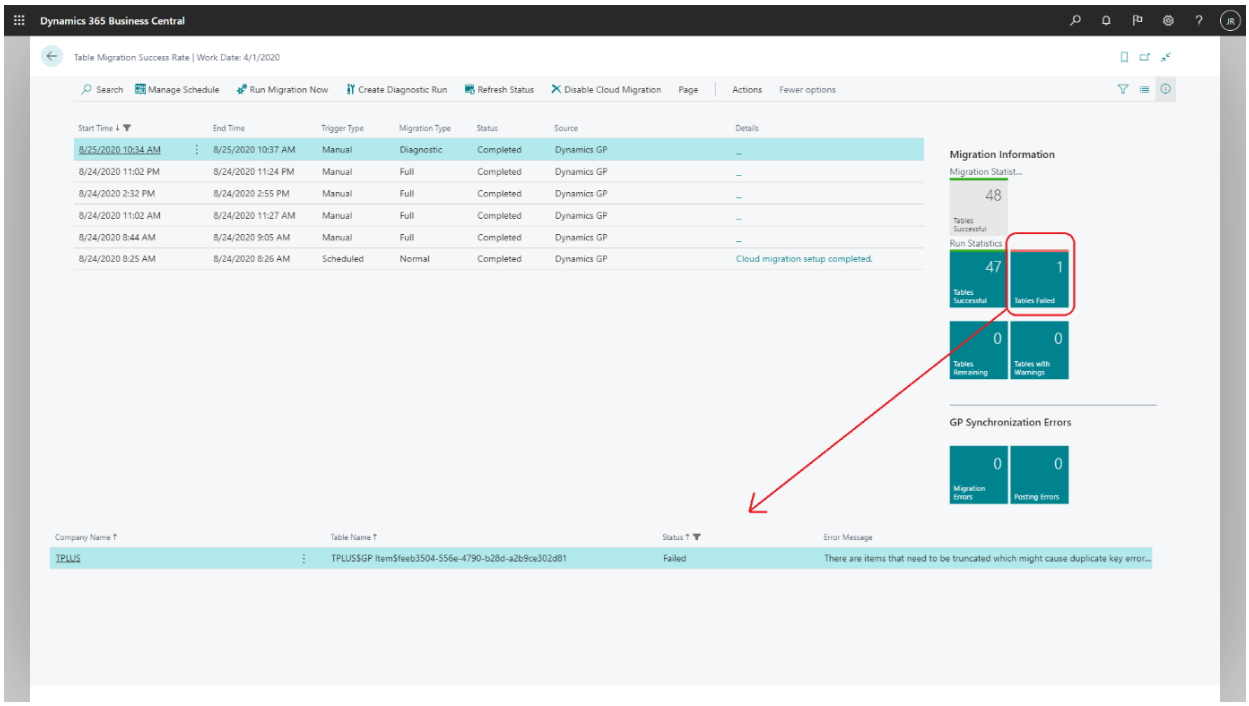
6. Checkbooks and outstanding transactions in Bank Reconciliation. We strongly recommend that you reconcile your checkbooks before you run the migration process to Business Central as we will bring over transactions that have not been reconciled during the migration process.

Diagnostics run

In the **Cloud Migration Management** page, you can create a diagnostics run to do more data validation/verification before the migration is run so that you can decrease the risk of a failed migration.

The maximum field length is different in Dynamics GP (30) and Business Central (20), and the diagnostics run checks for issues and shows warnings. The tool also checks item numbers to look for duplicates based on the character limit, and it checks to make sure there are no blank posting accounts that are needed for posting of transactions with the migration.

Here is an example of what you might see when you run a diagnostic run:



Migration tool support for Australia

Starting with 2020 release wave 2, organizations in Australia can migrate from Dynamics GP to an Australian version of Business Central online.

The process to run the migration in an Australian Business Central online is the same as running a migration in the other supported countries. However, there is some setup that needs to be done before the migration can be run. To make the changes to the setup, you must run through the cloud migration wizard. When the wizard setup is complete, and companies are fully set up, you must log into the companies that you are planning to migrate and make the following changes

- With transactions that are being migrated, we bring over totals on the invoices for the customers so GST information is already included in the transactions. You must turn off the GST and Adjustment Mandatory features in the **General Ledger Setup** page to allow the transactions to post during the migration. After the migration is complete, you can turn both GSP and Adjustment Mandatory back on so that new transactions that are entered in Business Central online will use this functionality.
- Within the posting process there is validation to look for a blank VAT business posting group and VAT product posting group. This combination isn't setup by default and is needed for the migration. So add a VAT posting configuration in the **VAT Posting Setup** page with blank values for the **VAT Bus Posting Group** and **VAT Prod. Posting Group** fields.

Move your Dynamics GP database to Azure Data Lake

Starting with 2020 release wave 2, you can create a copy of the Dynamics GP database in Azure Data Lake so that you have it for future reference after the migration to Business Central online. To take advantage of this functionality, there are some pieces that must be set up before the migration process. A customer is not required to copy their database to Azure Data Lake, but there are several benefits to being able to do so, including having access to the historical data that is not migrated by the migration tool. For an introduction to Azure Data Lake, see azure.microsoft.com.

To create an Azure Data Lake storage account, you log in to your Azure subscription (or sign up for an Azure subscription). Once logged into the Azure portal, you can create a storage account. Under the **Access Keys** section of the information about your storage account in the Azure portal, you can see the keys that you must provide in the cloud migration wizard in Business Central. For more information about Azure Data Lake and

setting up a storage account, see Azure documentation.

To move to Azure Data Lake

1. In your Business Central online tenant, choose the icon, enter **Cloud Migration Management**, and then choose the related link.

2. Choose the **Azure Data Lake** action.

   The **Azure Data Lake Migration Setup** guide takes you through the steps to connect your on-premises Dynamics GP database and your storage in Azure Data Lake. You must have created an Azure Data Lake storage account before launching the assisted setup, and you must have access to the storage account name and storage account key.

3. When the setup completes, you can monitor the progress of moving your data from on-premises to Azure Data Lake in the **Cloud Migration Management** page in Business Central.

If you go back to the Azure portal, you can see the data that was moved from your Dynamics GP database to Azure Data Lake. Under the containers section of the Data Lake storage, a new *gp-database* folder contains the company/system databases, and sub-folders contain the series folder and data files.

The database folder in Azure Data Lake will always be set as *gp-database*. If you want to rename it, you can do so after the migration has competed. The same applies to the various sub-folders. The folders created follow the standard naming convention of the *series* folders in Dynamics GP. An additional folder, *3rd Party*, will be created, and any files that haven't be mapped to a folder will reside in there. You can modify the content in the folders after the migration has completed.

If an Azure Data Lake migration has already been run in the Business Central company and you launch a second run, the migration tool will check to see if a *gp-database* folder exists. If you still want to run the migration again, you must delete the *gp-database* folder and its contents before you can run the wizard again.

At the end of the migration, you have a completely copy of your Dynamics GP database in the cloud with all the advantages of having access to historical data without having to maintain a server on-premises.

See also

Migrating On-Premises Data to Business Central Online

Important Information and Considerations for Before Upgrading to Dynamics 365 Business Central Spring 2019 and Later Versions

2/17/2021 • 5 minutes to read • Edit Online

Depending on which version you are upgrading from, and the degree to which your solution differs from the standard version of Business Central, you may want to prepare your solution for the upgrade. This topic provides important information and tips for things to consider when you prepare to upgrade to Business Central.

Migrate from Dynamics NAV to Business Central online

You can upgrade to Business Central online from supported versions of Dynamics NAV on-premises, provided that your application customization is handled by extensions. Any data from tables with code customizations cannot be carried forward from Dynamics NAV.

> **NOTE**
> Upgrade your solution to Business Central Spring 2019 (version 14) or later, and then migrate to Business Central online.

The process consists of two parts:

- Upgrade from Dynamics NAV to Business Central using the tools described in Upgrading to Business Central on-premises (version 14). For more information, see Supported Upgrade Paths to Dynamics 365 Business Central Releases.
- Convert non-standard functionality and customizations to apps and per-tenant extensions. For more information, see Deploying a Tenant Customization.
- Run the cloud migration tool as described in Running the Cloud Migration Tool, and then switch to use Business Central online going forward.

Upgrading from Dynamics NAV

This section lists specific changes between Dynamics NAV and Business Central.

Codeunit 1 has been deprecated and replaced

Dynamics NAV included codeunit **1 ApplicationManagement**. In Business Central, this codeunit has been retired, and new 'system' codeunits have been introduced in the 2 billion range.

For information, see Transitioning from Codeunit 1 to System Codeunits.

V1 Extensions have been discontinued

With Business Central, extensions V1 are no longer supported for on-premise installations. As a result, any custom extensions V1 must be converted to extensions V2 in the old environment before upgrading to Business Central.

For information about how to convert to extensions V2, see Converting Extensions V1 to Extensions V2.

MenuSuite not used for page and report search

With Business Central, the MenuSuite is no longer used to control whether a page or report can be found in the

search feature of the Web client. This is now determined by specific properties on the page and report objects. As part of the application code upgrade process, you change these properties on existing pages and reports used by the MenuSuite to ensure that they are still searchable from the Web client. For more information, see Making Pages and Reports Searchable After an Upgrade.

Dynamics 365 Sales integration

Because of changes in Dynamics 365 Sales and the integration since pervious releases, if your application is integrating with Dynamics 365 Sales, then you must perform a full upgrade instead of just a technical upgrade.

New and changed application features

There a several new and changed application features available in Business Central April 2019 for users, administrators, and developers. For an overview of these features, see Overview of Dynamics 365 Business Central April '19 release.

To take advantage of these all these features, you will have to perform an application code upgrade, not just a technical (platform) upgrade.

Changes to profiles in the CRONUS International Ltd. demonstration database and promoted actions

With the Business Central April 2019 release, profiles that are part of the CRONUS International Ltd. demonstration database, such as the **Sales Order Processor** profile, customize fewer pages compared to earlier releases. For customers that rely on these profiles, their users might experience slight differences in the layout of actions in the action bar on pages. Additionally, the layout of promoted actions on over 380 core application pages has been fine-tuned.

To ensure that users are not disrupted by these changes, we recommend that administrators and partners who are upgrading a customer to Business Central April 2019, review the layout of promoted actions when combined with their own code and profile customization.

Names of variables

Business Central introduces new methods and statements. If your solution includes variables where the name is now used by a standard AL method or statement such as REGISTERTABLECONNECTION or FOREACH, you must change the variables before you upgrade to Business Central. Alternatively, you can enclose the variable names in quotation marks. If you do not, and you import an object that has this code in text format, you cannot compile the object.

Deprecated or redesigned functionality

If you are upgrading a solution that depends on functionality that is deprecated or changed in the default version of Business Central, you must verify that the upgrade codeunits migrate data correctly. See the See Also section for links to descriptions of deprecated functionality.

Deprecated fields and fields marked as obsolete

Sometimes Microsoft will refactor code so that fields are no longer used, or the functionality is moved from the base application to an extension, for example. Typically, the upgrade toolkit will manage the upgrade impact, but for transparency, you can find a list of fields that are deprecated in the current release or marked to be obsolete in a later release. For more information, see Deprecated Fields, and Fields Marked as Obsolete.

Upgrade codeunits

When you introduce changes to the database schema, Business Central will check if these changes are

destructive or not. If the database check indicates that the change may lead to data deletion, such as if you are dropping a table column so that the contents of that column will be deleted, this is considered a destructive change. You will be prompted to handle the situation using upgrade codeunits.

Company names

If a company name includes a special character, an error may display during the upgrade. In this context, special characters include the following:
[~ @ # $ % & * ( ) . ! % - + / = ?]

If you are going to upgrade a database where one or more company name includes a special character, we recommend that you rename the company before you start the upgrade process. After the upgrade is successfully finished, you can rename the company again.

System tables with non-English names

In older versions of Dynamics NAV, you could translate the columns in system tables to a language other than English. Starting with version 3.0, we advised heavily against this, and versions later than Microsoft Dynamics NAV 2013 R2 require that all columns in all system tables are in English. As a result, if you try to open a database with non-English system tables in Microsoft Dynamics NAV 2013 R2 or later, an error displays, saying that one or more columns do not exist.

Make sure that all objects where compiled in a development environment with the right .ETX and .STX files. You can verify that you are running in the correct environment with English (US) as the base language by opening the **ndo$dbproperty** table in SQL Server Management Studio. In the **Identifiers** column, the word `Object` must be written exactly as shown here.

See Also

Upgrading the Application Code
Upgrading the Data
Deprecated Fields, and Fields Marked as Obsolete

FAQ about Connecting to Business Central Online from On-Premises Solutions

2/17/2021 • 7 minutes to read • Edit Online

This section contains answers to frequently asked questions about connecting on-premises solutions to Business Central online.

> **TIP**
> Check the tips in this article if your organization is not yet ready to migrate to Business Central online but still wants to enjoy some of the benefits of the cloud. The same tips apply to when you are migrating to the cloud and are running the migration tool. For more information, see Running the Cloud Migration Tool.

Which products and versions are supported for this connection?

The current version of Business Central can connect the following products in order to provide intelligent insights:

- Dynamics GP (supported major versions)
- Business Central on-premises

  Currently, you can migrate to Business Central online from versions 14, 15, 16, and 17.

If you are currently on a version of Dynamics NAV, you must upgrade to Business Central on-premises, and then switch to Business Central online. For more information, see Upgrading from Dynamics NAV to Business Central online.

System requirements

To connect to the cloud through Business Central, the on-premises solution must use SQL Server 2016 or a later version, and the database must have compatibility level 130 or higher. The on-premises solution must also be one of the supported versions.

How is my on-premises data replicated to my Business Central online tenant?

Data is replicated using an Azure service called Azure Data Factory (ADF). ADF is a service that is always running within the Business Central online service manager. When you have connected to the intelligent cloud, a data pipeline is created in the ADF service so that data can flow from your on-premises solution to your Business Central online tenant. If your data source is a local SQL Server instance, you will also be asked to configure a self-hosted integration runtime (SHIR). The runtime is installed locally and manages the communication between the cloud services and your on-premises data without opening any ports or firewalls.

Are there any limits on the amount or type of data will replicate?

There are no restrictions on the type of data that can be replicated. In the current version of Business Central, the migration tool is by default limited to migrate databases up to 80 GB. If your database is larger than 80 GB, we recommend that you reduce the number of companies that you are migrating data for. You can specify which companies to include in the migration in the assisted setup wizard.

If you want to add more companies after the first selection of companies, you can add additional companies in the **Cloud Migration Management** page in Business Central online. For more information, see Adding a tenant to an existing runtime service, or updating companies.

If you are looking at migrating databases larger than 80 GB, we recommend that you contact the support team and work with them to make sure that the migration is successful.

Is my SQL connection string required to set up the connection?

Yes. The SQL connection string is passed to Azure Data Factory, where it is encrypted and delivered to your Self-Hosted Integration Runtime. The connection string is used to communicate with your SQL Server instance during the data replication process. For more information, see How do I find my SQL connection string?.

How do I find my SQL connection string?

Find the connection string to your SQL database in SQL Management Studio or Visual Studio. The user name and password defined in the connection requires a SQL Authenticated user name/password. Your connection string will look something like this:

Server=tcp:{ServerName},1433;Initial Catalog={DatabaseName};Persist Security Info=False; User ID={UserName};Password={Password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=True;Connection Timeout=30;

How do I find the Integration Runtime name?

Find the Integration Runtime name in the Microsoft Integration Runtime Manager, which you can find in your Windows system tray or by searching for the program. You must type the name. You will not be able to copy and paste the name.

I am a hosting partner - do I need to configure the Self-Hosted Runtime Service for each tenant?

No, there is no limit on the number of tenants that can be added to your Self-Hosted Integration Runtime. Each added tenant will have a dedicated pipeline created.

Will data from tables with code customizations replicate?

No, only tables that are available in both your on-premises solution and your Business Central online tenant will replicate. Any customization must be made into an extension and installed on both your on-premises solution and your Business Central online tenant to replicate.

Why are my permissions restricted in the Business Central online tenant?

When you connect your on-premises solution to Business Central online for intelligent insights, all existing users are automatically added to the *Intelligent Cloud* user group, unless they have the SUPER permission set. In this configuration, your on-premises solution is the master where all business transactions take place. The Business Central online environment is read-only, and the data is used to generate intelligent business insights based on your on-premises data for you. We restrict permissions to prevent users from accidentally entering transactions or updating master records only to have that information overwritten and lost when data replication takes place.

Can I 'turn off' my intelligent cloud?

You can switch off your connection to the Business Central online environment at any point. Once you disable

your intelligent cloud configuration, your on-premises solution and the Business Central online tenant will become independent of one another. If you switch off the connection, and you want to use your Business Central online environment as your primary solution to run and manage your business, you must reassign permissions to provide read/write access to the relevant users.

For more information, see Managing Users and Permissions.

Will my on-premises users and permissions replicate?

No. Since you are not required to configure your on-premises solution with Azure Active Directory (Azure AD), we cannot guarantee a mapping between on-premises users and users in your Business Central online tenant. Business Central online requires Azure AD accounts, and users must be manually added. All permissions must be granted in the Business Central tenant, independent from your on-premises permissions.

For more information, see Managing Users and Permissions.

Can I view insights from cloud services in my on-premises solution?

Yes, the **Intelligent Cloud Insights** page can be hosted within your on-premises solution if that is one of the currently supported solutions. Each user will need to have a Business Central license to view the data.

Can you export to Excel, modify the contents, and import the data back in?

You can export the list to Excel from the Business Central online tenant, but since the data is read-only you cannot make changes and import it again.

Is the data replication only one way?

Yes, data is only replicated from the on-premises solution to your Business Central online tenant.

Is there a cost to connect to the intelligent cloud?

Currently, the only costs associated with the intelligent cloud are your named user license costs. For more information, see the Business Central Licensing Guide (download).

Why did my Role Center change after configuring the intelligent cloud?

To keep the Role Center experience as clean as possible and avoid permission errors, we automatically hide actions that would generate a permission error for the user.

Should I uninstall all my Business Central extensions?

Not necessarily. Most extensions will run without issues in the online environment. You may want to consider uninstalling extensions that send data to an external service to avoid potential duplicated calls to that service. It is a best practice to test any extension in a sandbox tenant configured for the Business Central online environment that you are connecting to.

How do I build an extension that enables data replication?

The extension must be created in the same manner as any other extension. For data to replicate, you must add a **ReplicateData** property to your table and set the value to *True*. If your extension connects with an external service and you want to restrict any service calls from your Business Central online tenant, a good practice

would be to store the connection information in a separate table and set the **ReplicateData** property to *False*. This would enable you to keep the extension installed but prevent it from making any type of service calls from the read-only Business Central tenant. Once the extension is installed in Business Central online and on-premises, the data will begin to replicate.

See also

Running the Cloud Migration Tool
Migrating On-Premises Data to Business Central Online

Troubleshooting Cloud Migration

2/17/2021 • 6 minutes to read • Edit Online

In this article, you learn how to troubleshoot problems that you may experience with the cloud migration of Business Central. For the cloud migration to work properly, there are certain requirements that must be met on the online and on-premises databases. The following sections talk about these requirements, how you can check them, and correct them as needed.

SQL Server compatibility level

> Database: on-premises

The SQL Server compatibility level must be 130 or higher.

To check the compatibility level run following query:

```
SELECT compatibility_level FROM sys.databases WHERE name = 'YourDatabaseName';
```

To change the compatibility level, run this query:

```
ALTER DATABASE YourDatabaseName SET COMPATIBILITY_LEVEL = 130;
```

> **NOTE**
> You may also get the following error when compatibility level isn't set to the expected value: "A database operation failed with the following error: Invalid length parameter passed to the LEFT or SUBSTRING function."

Migration user

> Database: on-premises

Make sure that the database user that the Integration Runtime uses to connect to your database has access to the database.

You can verify the database user, for example, by running the following command using Microsoft PowerShell ISE.

```
sqlcmd -S 'ServerName\ServerInstance' -d 'DatabaseName' -U 'UserID' -P 'Password' -Q 'select * from [dbo].
[CRONUS USA, Inc_$Accounting Period$437dbf0e-84ff-417a-965d-ed2bb9650972]'
```

Change tracking

> Database: on-premises

Change tracking must be enabled on the database. It should be enabled automatically. However, if you see an error like **Change tracking is not enabled on table <number>** during migration, you'll have to enable it

manually.

To enable change tracking on your database, run the following query:

```
ALTER DATABASE [YOUR DATABASE] SET CHANGE_TRACKING = ON (CHANGE_RETENTION = 2 DAYS, AUTO_CLEANUP = ON)
```

To enable change tracking on specific tables, run the following query:

```
ALTER TABLE [YOUR TABLE] ENABLE CHANGE_TRACKING
```

User permissions

> Database: online

If a user has problems managing a cloud migration, like starting migration, initializing companies, or migrating data from earlier versions, check that:

- The user has a Business Central license (Essentials or Premium, depending on their solution). We recommend using free Dynamics 365 Business Central Premium Trial subscription for this user.
- The user is assigned the SUPER permission set.

Users without a license, such as internal administrators or delegated administrators, aren't allowed to run the migration.

Self-Hosted Integration Runtime

> Database: on-premises

- Ensure that you're running the latest, compatible version of Microsoft Integration Runtime (IR).

  You can check for and download the latest version from the Microsoft Download Center.

  When downloading and installing Integration Runtime, choose version 5 (IntegrationRuntime_5.x.x.x.msi) only if your machine runs .NET Framework Runtime 4.7.2. Otherwise, or if in doubt, choose version 4 (IntegrationRuntime_4.x.x.x.msi).

  > **IMPORTANT**
  > Running Integration Runtime v5 on a machine that doesn't have .NET Framework Runtime 4.7.2 can cause timeouts when connecting to the on-premise SQL Server, which will break the cloud migration setup.

  Before you install a new Integration Runtime version, uninstall the old version. When you uninstall the old version, choose to delete the user data (such as authentication key and data source credentials) when prompted. Then, install the Integration Runtime again and connect it to the online environment using the new authentication key.

- If you get a "Failed to enable your replication." error when running the**Data Migration Setup** assisted setup, check the IR logs.

  You view the logs by using the **Microsoft Integration Runtime Configuration Manager**, which was installed on-premises as part of the IR installation. Launch **Microsoft Integration Runtime Configuration Manager**, select the **Diagnostics** tab, then select **View Logs** action. You can also use the **Test Connection** feature on the page to verify that your user can connect to the database.

- Synchronization errors can sometimes occur because IR is installed on a laptop or desktop computer where the hibernate feature is turned on.

  The computer where IR is installed ideally shouldn't be switched off, go to sleep, or hibernate. If these conditions happen, the IR may get into an error state. In this case, we recommend that you reinstall the IR and turn off sleep hibernate on the computer.

- Make sure the machine, which you use for hosting IR has plenty of memory (RAM) available. Migration can be interrupted by your machine running out of memory, and you can find this issue described in the IR log. To prevent this situation, avoid running too many migrations simultaneously using the same IR. Every additional parallel migration slows down the overall progress considerably.

If you experience problems with Microsoft Integration Runtime, also see Troubleshoot self-hosted integration runtime.

Migrating between multiple source and destination databases

> Database: online and on-premises

- If you migrate several on-prem databases to several online environments, it's possible to reuse the same IR for these migrations.

  Once you've successfully connected and migrated data into one online environment, you can reuse the IR for another environment. To reuse an IR, enter its name in **Integration Runtime Name** field of **Data Migration Setup** assisted setup, instead of leaving the field blank.

- Use a restored backup when migrating the same on-premise database to different online environments.

  Cloud migration stores some data in the on-premise database. So using the same on-premise database to migrate into another online environment can affect the next synchronization run. If you need to migrate to several online environments, we recommend you make a backup of the on-premise database before enabling the data migration. Then, restore the on-premise database to this backup before setting up migration to another online environment.

- Avoid running several migrations of the same on-prem database to different online environment at the same time.

  If you need to do this type of migration, then migrate data sequentially. First, migrate data into the online environment and disable the migration. Then restore the on-prem database from backup and enable the migration again by providing a connection string to this database. You can use the same Integration Runtime and Authorization key.

- Don't try to migrate data from several on-premise databases into the same online environment at the same time.

  For example, you may have two companies, where each company is in its own on-premise database. If you need to do this type of migration, the migrate data sequentially. First, migrate data from one database into the online environment and disable the migration. Then set up the migration in the same online environment, provide a new connection string to the next on-prem database. You can use the same Integration Runtime and Authorization key.

Product version

> Database: online

- When running the**Data Migration Setup** assisted setup, make sure to select the right product that you

want to migrate from. Depending on which Cloud Migration apps you've installed, the assisted setup will let you choose from three options:

OPTION	WHEN TO USE
Dynamics 365 Business Central	Select this option if you're migrating from the Business Central latest version, currently version 17
Dynamics 365 Business Central - Previous Version	Select this option if you're migrating from the an earlier supported version. Currently, you can migrate to Business Central online from versions 14, 15, 16, and 17.
Dynamics GP	Select this option if you're migrating from the Dynamics GP product.

- When migrating data from Business Central, check the `applicationVersion` field in the `$ndo$tenantdatabaseproperty` table. Set this field to the correct version in the SQL if it's blank or not up to date. The migration code uses the field's value for the following reasons:
  - Verifies that you're migrating from a supported version
  - Verifies that you've selected the right product version in the **Data Migration Setup** assisted setup, like Dynamics 365 Business Central or Dynamics 365 Business Central - Previous Version.
  - Determines which upgrade code will be executed.

    If that field is blank, the migration can't run.

Disabling the Cloud Migration

> Database: online

When you've completed the migration, disable cloud migration by using the **Disable Cloud Migration** action on the **Cloud Migration Management** page. This action properly disengages the synchronization and cleans up the Azure Data Factory resources deployed for this migration.

> **IMPORTANT**
> Just uninstalling the Cloud Migration apps, even with the option to remove the data, won't disable the migration in the same way. If you don't disable **Cloud Migration**, users will experience permission-related errors when they try to modify records in the migrated companies.

See also

Running the Cloud Migration Tool
Migrate to Business Central Online from Business Central On-premises
Migrate to Business Central Online from Dynamics GP
Upgrading from Dynamics NAV to Business Central Online
Migrating On-Premises Data to Business Central Online
FAQ about Connecting to Business Central Online from On-Premises Solutions

Update 17.4 for Microsoft Dynamics 365 Business Central online 2020 release wave 2

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 17.4? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Find an overview of hotfixes in this article.

Feature changes

- Use recurring journals to allocate balances by dimension values
- The Busines Central app for Microsoft Teams reaches General Availability

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 2 (release from October 2020 through March 2021), find the link to the release plan here.

Upgrade to 17.4

Please note that new customers will automatically get the latest builds of Business Central (17.4). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

OData V3 will be removed with 2021 Wave 1
As previously announced, OData Version 3.0 (V3) has been deprecated since April 2020 and support for it will be removed in Dynamics 365 Business Central 2021 release wave 1. OData V4 is the current recommended version of OData and current integrations needs to move to OData V4. To identify incoming OData V3 requests, enable and use the Web Service Request telemetry that's available to partners. Notifications has also been sent via M365 Message Center to tenants with identified usage.

Release Plan for wave 1 2021 is out
The release plan for Dynamics 365 Busines Central wave 1 2021 is now live! You can find an overview of all the new and planned features here.

Basic Authentication
Basic Authentication (Web Service Access Key) removal for Business Central online has been postponed until April 2022. We see that some integrations needs more time to move from Web Service Access Key usage to OAuth. PowerShell samples on how to connect to Business Central is published on GitHub.

Business Central Office Hours Calls in February
Make sure to join the office hours calls around 'Power Platform Integration, Power Apps, and Dataverse' on February 9 and 'Performance Toolkit and Telemetry; How to deploy for performance' on February 23. Register

and stay tuned for the upcoming calls: https://aka.ms/BCOfficeHours.

It's time to switch your Dynamics 365 Business Central browser to Microsoft Edge

On April 2, 2021, Microsoft will remove Internet Explorer 11 and Microsoft Edge Legacy browsers from the list of supported browsers for the Business Central modern clients. Read the details, how it applies to you and actions to take here.

Update 17.3 for Microsoft Dynamics 365 Business Central online 2020 release wave 2

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 17.3? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Find an overview of hotfixes in this article.

Feature changes

- Restoring environments in Business Central admin center
- Use Shortcut dimensions in G/L Entries for Financial reporting
- Signal from web service key authentication added to Application Insights telemetry for partners
- Handle Price List Exceptions with Allow Updating Defaults

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 2 (release from October 2020 through March 2021), find the link to the release plan here.

Upgrade to 17.3

Please note that new customers will automatically get the latest builds of Business Central (17.3). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

Basic Authentication

Basic Authentication (Web Service Access Key) removal for Business Central online has been postponed until April 2022. We see that some integrations needs more time to move from Web Service Access Key usage to OAuth. PowerShell samples on how to connect to Business Central is published on GitHub.

Snapshot debugging is now enabled in production environments

From version 17.2 and forward it is now possible to use snapshot debugging to investigate production environments. For more information, see the help page or view the Virtual Event for 2020 Wave 2 release recording on the What's New in Visual Studio and AL, covering snapshot debugging in details.

Business Central Office Hours Calls in January

Make sure to join the office hours calls around 'Customer Migration Tooling' on January 12 and 'Power Platform Integration – Power BI' on January 26. Register and stay tuned for the upcoming calls: https://aka.ms/BCOfficeHours.

Want to improve the performance of Business Central?

Visit https://aka.ms/bcperformance and learn about best practices, dos and don'ts and different ways to make

changes with a performance impact. The Performance Tuning Guide will help you understand and improve the performance of Business Central whether you are a functional consultant, a developer, or an administrator.

It's time to switch your Dynamics 365 Business Central browser to Microsoft Edge

On April 2, 2021, Microsoft will remove Internet Explorer 11 and Microsoft Edge Legacy browsers from the list of supported browsers for the Business Central modern clients. Read the details, how it applies to you and actions to take here.

Update 17.2 for Microsoft Dynamics 365 Business Central online 2020 release wave 2

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 17.2? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Find an overview of hotfixes in this article.

Feature changes

- Application Performance Improvements
- Signal from job queue execution added to Application Insights telemetry for partners
- Signal from permission changes to added to Application Insights telemetry for partners
- Signal from the retention policy feature added to Application Insights telemetry for partners
- Signal from the sensitive field audit feature added to Application Insights telemetry for partners
- Signal from the email feature added to Application Insights telemetry for partners
- Signal from application packages lifecycle added to Application Insights Telemetry for partners
- Snapshot Debugging

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 2 (release from October 2020 through March 2021), find the link to the release plan here.

Upgrade to 17.2

Please note that new customers will automatically get the latest builds of Business Central (17.2). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

Business Central Office Hours

The Office Hours are back! Find an overview of the topics and register here: https://aka.ms/BCOfficeHours.

Small update for Business Central mobile app

A small update for the Business Central mobile app (with added languages and some fixes) is being released to the app stores on all three platforms (Windows, iOS, and Android). Please remember to update the Business central app on your device (version 3.1.12032 or later). Direct links: Windows (Microsoft Store), iOS (Apple's App Store), Android (Google's Play Store).

Watch what's new sessions on demand

You can still watch the sessions from the Business Central Launch Event in October. Register and get access to 30+ sessions here.

Migrate directly from version 14, 15, and 16 to Business Central online

Business Central includes a cloud migration tool that administrators can use to migrate customer data from on-premises databases to Business Central online. Starting now, the Cloud Migration app also supports migration from the earlier on-premises versions of Business Central, with the latest cumulative update applied: Migrate directly from versions 14.x, 15.x, and 16.x to version 17.x. This helps customers reduce their costs of migrating to the Business Central cloud significantly, as they can skip upgrading their on-premises environments to the latest version of Business Central and just migrate the data, including all historical transactions. The Cloud Migration app now includes all the necessary data upgrade logic to convert the data from the previous versions to the data structure of 2020 release wave 2 (version 17.0). Learn more here.

Update 17.1 for Microsoft Dynamics 365 Business Central online 2020 release wave 2

2/17/2021 • 3 minutes to read • Edit Online

Would you like to know what has changed in update 17.1? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Find an overview of hotfixes in this article.

Feature changes

- Enhanced email capabilities
- Notify users of high-risk changes in selected setup fields
- Use new sales pricing experience
- Control how Account Schedules for core financial reports are generated
- Improved VAT Registration no. lookup
- Extension lifecycle telemetry in Application Insights for ISVs
- Use environmentType and environmentName launch.json properties instead of obsoleted sandboxName
- Symbols can be downloaded with a snapshot initialize launch.json configuration, using above environment properties

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 2 (release from October 2020 through March 2021), find the link to the release plan here.

Upgrade to 17.1

Please note that new customers will automatically get the latest builds of Business Central (17.1). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

NEW! Migrate directly from version 14, 15, and 16 to Business Central online
Business Central includes a cloud migration tool that administrators can use to migrate customer data from on-premises databases to Business Central online. Starting now, the Cloud Migration app also supports migration from the earlier on-premises versions of Business Central, with the latest cumulative update applied: Migrate directly from versions 14.x, 15.x, and 16.x to version 17.x. This helps customers reduce their costs of migrating to the Business Central cloud significantly, as they can skip upgrading their on-premises environments to the latest version of Business Central and just migrate the data, including all historical transactions. The Cloud Migration app now includes all the necessary data upgrade logic to convert the data from the previous versions to the data structure of 2020 release wave 2 (version 17.0). Learn more here.

Database size when migrating from Business Central on-prem to online

We have increased the limit we apply to the database size from 30 Gb to 80 Gb when migrating from Business Central on-prem to Business Central online using the Cloud Migration tool. If you are looking at migrating a larger database, we recommend that you contact the support team and work with them to make sure that the migration is successful. Learn more here and read about running the cloud migration tool here.

Watch what's new sessions on demand

Watch the sessions from the Business Central Launch Event in October. Register and get access to 30+ sessions here.

Snapshot debugging

Snapshot debugging has now been enabled for sandbox environments. It will be enabled in production in a later minor. Learn more here.

Apps are moving to office.com

The home for all your business applications across Dynamics 365 and Microsoft Power Platform is moving to office.com, where you'll find the Business Central tiles for production and sandbox environments. Learn more here.

Easy access to production or sandbox environments from the mobile app

Users of mobile devices can now choose between their sandbox and production environments without the need to use the pre-crafted URL as before. Partners running their own apps based on Business Central can also let their users explore it from mobile devices. Learn more here.

Unblock multifactor authentication for mobile apps

We have updated core authentication components of the mobile app to support multifactor authentication. Enabling such flow in the user setting on your Microsoft 365 account and then using it on a mobile device (via an SMS code, authenticator app, or more) is now possible after updating the Business Central mobile app to version 3.0 or higher. Learn more here.

Update 16.5 for Microsoft Dynamics 365 Business Central online 2020 release wave 1

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 16.5? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Find an overview of hotfixes in this article.

Feature changes

- Delete extension data

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 1 (release from April 2020 through September 2020), find the link to the release plan here.

For a list of next wave release plans, see the 2020 release wave 2 release plan.

Upgrade to 16.5

Please note that new customers will automatically get the latest builds of Business Central (16.5). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

It's time to switch your Dynamics 365 Business Central browser to Microsoft Edge
On April 2, 2021, Microsoft will remove Internet Explorer 11 and Microsoft Edge Legacy browsers from the list of supported browsers for the Business Central modern clients. Read the details, how it applies to you and actions to take here.

Leveraging IoT data in Dynamics 365 Business Central
Make sure to read the blog about getting started with leveraging the Internet of Things (IoT) data in Dynamics 365 Business Central using services from Microsoft Azure is easier than you may think.

Support for an unlimited number of production and sandbox environments
With 2020 release wave 2 (October), we're introducing the ability for customers to buy additional Business Central production environments to expand their business. More environments will open up new opportunities like creating more business branches, moving into more countries, or expanding within their current country, read more here.

Running a Container-Based Development Environment
From 16.4 and onwards Microsoft will stop producing docker images and instead publish builds as artifacts, which can be used together with the generic Docker image to run the image you want. Read more here.

Update 16.4 for Microsoft Dynamics 365 Business Central online 2020 release wave 1

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 16.4? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links that you might find interesting.

Hotfixes

Find an overview of hotfixes in this article.

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 1 (release from April 2020 through September 2020), find the link to the release plan here.

For a list of next wave release plans, see the 2020 release wave 2 release plan.

Upgrade to 16.4

Please note that new customers will automatically get the latest builds of Business Central (16.4). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

Running a Container-Based Development Environment
From 16.4 and onwards Microsoft will stop producing docker images and instead publish builds as artifacts, which can be used together with the generic Docker image to run the image you want. Read more here.

Want to improve the performance of Business Central?
Visit aka.ms/bcperformance and learn about best practices, dos and don'ts and different ways to make changes with a performance impact. The Performance Tuning Guide will help you understand and improve the performance of Business Central whether you are a functional consultant, a developer, or an administrator.

Virtual sessions of what is new in 2020 release wave 1?
Did you attend the #MSDyn365BCVirtualEvent on June 3rd? Don't forget to watch the 15+ virtual on demand sessions featuring new innovations, tools, and in-depth product demonstrations from the experts. You just need to register to view the sessions, but it's free.

Update 16.3 for Microsoft Dynamics 365 Business Central online 2020 release wave 1

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 16.3? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Link to Hotfixes.

Feature changes

- Bank reconciliation improvements
- Extension lifecycle telemetry in Application Insights for partners
- Client page view telemetry in Application Insights for partners
- Installing AppSource apps updates in the Business Central administration center – rollout has been completed. Customers can now discover and install all AppSource apps updates in the Business Central Admin Center.
- Feature Update: Page layout is now cached to browser storage

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 1 (release from April 2020 through September 2020), find the link to the release plan here.

Upgrade to 16.3

Please note that new customers will automatically get the latest builds of Business Central (16.3). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

Keep track on deployed builds
In between the Microsoft Dynamics 365 Business Central minor updates we deploy new builds as they become available. Visit the Update History to see the latest important builds that have been pushed out automatically since last minor update.

Want to improve the performance of Business Central?
Visit aka.ms/bcperformance and learn about best practices, dos and don'ts and different ways to make changes with a performance impact. The Performance Tuning Guide will help you understand and improve the performance of Business Central whether you are a functional consultant, a developer, or an administrator.

Are you a Business Central Wave 'Champ'?
Did you attend the #MSDyn365BCVirtualEvent on June 3rd? Don't forget to watch the 15+ virtual on demand sessions featuring new innovations, tools, and in-depth product demonstrations from the experts. The best part

of it - it's all for free! You just need to register to become a Business Central 'Wave Champ' and boost your knowledge.

Update 16.2 for Microsoft Dynamics 365 Business Central online 2020 release wave 1

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 16.2?
Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Link to Hotfixes

Feature changes

- Migrate data from Business Central 14.x on-premises to Business Central 15.x online
- Update error telemetry in Application Insights for partners
- New URL parameter hides web client header
- Improved user experience to keep things from going wrong
- Use modern authentication to connect to Microsoft Dataverse and Dynamics 365 Sales
- Installing AppSource apps updates in the Business Central administration center – please note gradual availability! Until the feature is generally available, some AppSource apps can't be updated using this feature. For information about the timeline, see here

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire 2020 release wave 1 (release from April 2020 through September 2020), find the link to the release plan here.

Upgrade to 16.2

Please note that new customers will automatically get the latest builds of Business Central (16.2). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

A change to the Web client header
The top-most header in the Business Central Web client and desktop app have been updated to align with other Microsoft 365 and Dynamics 365 apps. The most notable change is to the height of the header that has now been reduced to 48 pixels.

Warning of performance impact when using configuration packages to import data
Configuration packages were designed to speed up the implementation of new companies. In case a customer wants to use Configuration Packages as a quick way to import data -sometimes big data- from other systems to Business Central, this small feature analyzes if the import will bring data to a new company or not and, depending of that and the size of the package, alerts the user that the performance of its tenant could be affected suggesting other ways to import or request partner help.

Want to improve the performance of Business Central?

Visit aka.ms/bcperformance and learn about best practices, dos and don'ts and different ways to make changes with a performance impact. The Performance Tuning Guide will help you understand and improve the performance of Business Central whether you are a functional consultant, a developer, or an administrator.

Are you a Business Central Wave 'Champ'?

Did you attend the #MSDyn365BCVirtualEvent on June 3rd? Don't forget to watch the 15+ virtual on demand sessions featuring new innovations, tools, and in-depth product demonstrations from the experts. The best part of it - it's all for free! You just need to register to become a Business Central 'Wave Champ' and boost your knowledge.

Update 16.1 for Microsoft Dynamics 365 Business Central 2020 online release wave 1

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 16.1?
Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update. In addition, we have gathered some good to know information and links, you might find interesting.

Hotfixes

Link to Hotfixes

Feature changes

- Company lifecycle telemetry in Application Insights for partners

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire wave 1 (release from April 2020 through September 2020), find the link to the release plan here.

Upgrade to 16.1

Please note that new customers will automatically get the latest builds of Business Central (16.1). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

- **Minimum requirements for Business Central Administration Center**
  We have updated system requirements with the list of browsers supported by the Administration Center. When signing in from an unsupported browser that delivers a significantly degraded experience, administrators will immediately see a message that prevents them from continuing to the Administration Center. For example, it is not possible to access the Business Central Administration Center from Internet Explorer.
  See minimum requirements
  Learn more about the Business Central Administration Center

- **Sandbox environment picker**
  When you navigate from the Business Central Sandbox tile in the Dynamics 365 Home portal or in the App Launcher, a dialog will be displayed for you to choose exactly which sandbox environment from your organization you would like to navigate to.
  Read more about sandbox environments

- **Message to Global Admins about the capabilities**
  On login, Global Admins will get a message about the capabilities in Business Central according to their license or suggesting one if they don't have any assigned.

- **Coming up: Virtual partner readiness event - register now!**
  We're excited to share the latest innovations for Business Central. Join us at the upcoming Business

Central All Access Virtual Event June 3rd with more than 15 deep dive sessions on what's new in 2020 release wave 1 (April release), also available on-demand.
Registration open now at aka.ms/virtual/businesscentral/2020RW1
Pssst...did we mention it's for free...

- **Major updates**
  Get an overview of what you need to know about how a major Business Central update rolls out. It includes key dates, actions you need to take, and answers some common questions.
  See Major Updates of Business Central Online

Update 15.4 for Microsoft Dynamics 365 Business Central 2019 online release wave 2

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 15.4? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update.

Hotfixes

Link to Hotfixes

Feature changes

- Bookmarking a report - rollout is complete. With Update 15.4, all environments now include this feature.

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire wave 2 (release from October 2019 through March 2020), find the link to the release plan here.

Upgrade to 15.4

Please note that new customers will automatically get the latest builds of Business Central (15.4). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

- Major updates
  Get an overview of what you need to know about how a major Business Central update rolls out. It includes key dates, actions you need take, and answers some common questions. See Major Updates of Business Central Online.
- Preview environments of next major update
  Partners should start to prepare for the next major update of Business Central in April. Partners can try out new functionality in preview environments, give Microsoft feedback and validate your extensions. See more details in Prepare for major updates with preview environments
- Visit our new aka.ms/bcperformance page and start your learning about Business Central performance.

Update 15.3 for Microsoft Dynamics 365 Business Central 2019 online release wave 2

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 15.3? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update.

Hotfixes

Link to Hotfixes

Feature changes

- View Your Reference and External Document No. on sales documents
- Skip empty lines in the Account Schedule report
- MICR fonts available in Business Central online
- New events to unblock printer extensions in Business Central online
- Bookmarking a report
- Sign-in attempt telemetry in Application Insights for partners

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire wave 2 (release from October 2019 through March 2020), find the link to the release plan here

Upgrade to 15.3

Please note that new customers will automatically get the latest builds of Business Central (15.3). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

- Changes to Browser Cookie Policy
  Avoid disruption by following these steps to prepare for tighter security policies starting February 2020 in Google Chrome and Microsoft Edge. For more information, see Preparing Dynamics NAV or Dynamics 365 Business Central for Upcoming Changes to Browser Cookie Policy.

- Using Internet Explorer
  From Update 15.3 onwards, users who access Business Central from Internet Explorer will receive a notification suggesting they consider a different browser. Users who switch to a modern browser, such as Microsoft Edge, typically observe improved performance and a generally smoother experience. Learn more about our supported and recommended browsers for Business Central on premises and online.

- To support partners who are interested in monetizing the app they have published to AppSource, update 15.3 comes with new options. These options allow partners to understand number, type and state of Business Central licenses in the current tenant as well as number of allocated seats. That data enables the partner to implement user-based billing into their apps. To learn more, see Azure AD Licensing.

- Visit our new aka.ms/bcperformance page and start your learning about Business Central performance.

Update 15.2 for Microsoft Dynamics 365 Business Central 2019 online release wave 2

2/17/2021 • 2 minutes to read • Edit Online

Would you like to know what has changed in update 15.2? Below you'll find an overview and relevant links to what has been done on hotfixes and regulatory features in this update.

Hotfixes

Link to Hotfixes

Feature changes

- Add the Name 2 field to customer and vendor cards
- Include job information in archived sales and purchase documents
- Use VAT clauses on different document types
- View item availability by unit of measure
- View the number of general journal lines

Release Plan

If you want to get a comprehensive overview of what's new and planned for Business Central online for the entire wave 2 (release from October 2019 through March 2020), find the link to the release plan here

Upgrade to 15.2

Please note that new customers will automatically get the latest builds of Business Central (15.2). If you are an existing partner/customer, you will receive an email notification as soon as your environment has been upgraded.

Good to know

Please find an overview of new "How to videos" recently published on Dynamics 365 YouTube channel for the Business users and the Functional Consultants. These videos complement the extensive MS Learn and Documentation content already available for Business Central.

- How to set up a customer
- How to adjust exchange rates
- How to set up a vendor
- How to set up purchasers
- How to prioritize vendors
- How to set up schedule reports
- How to set up sales people
- How to apply a payment to multiple customer ledger entries
- How to reconcile customer payments manually
- How to create credit notes and applying across single and multiple ledgers
- How to offer discounts and special prices

- How to link purchase orders to a sales order
- How to set up a new bank account
- How to set up an item to sell
- How to set up payment methods
- How to set up chart of accounts
- How to submit a support request
- How to report a production outage
- How to correct or cancel purchase invoices
- How to adjust physical inventory levels
- How to work with inventory costing
- How to set up an inventory location
- How to create purchase order
- How to email documents
- How to set up documents sending profiles
- How to setup categories
- How to set up customer approval workflow
- How to open a new fiscal year (part of payment methods)

Building Your First Sample Extension With Extension Objects, Install Code, and Upgrade Code

2/17/2021 • 16 minutes to read • Edit Online

This walkthrough will guide you through all the steps that you must follow to create a sample extension in AL. New objects and extension objects will be added to the base application for a simple reward feature for customers. Every section of this exercise includes code that serves for installing, customizing, or upgrading this sample extension. The final result can be published and installed on your tenants.

About this walkthrough

This walkthrough illustrates the following tasks:

- Developing a sample extension with a table, a card page, and a list page.
- Deploying the sample extension to your development sandbox environment.
- Using the Dynamics 365 Business Central Designer to modify visual aspects of the extension.
- Creating extension objects that can be used to modify page and table objects.
- Initializing the database during the installation of the extension.
- Upgrading and preserving data during the upgrade of the extension.

Prerequisites

To complete this walkthrough, you will need:

- The Dynamics 365 Business Central tenant.
- Visual Studio Code.
- The AL Language extension for Visual Studio Code.

For more information on how to get started with your first extension for Dynamics 365 Business Central, see Getting Started.

Rewards extension overview

The extension enables the ability to assign one of three reward levels to customers: GOLD, SILVER, and BRONZE. Each reward level can be assigned a discount percentage. Different types of objects available within the AL development environment will build the foundation of the user interface, allowing the user to edit the information. If you look for another option to update the layout of a page, you can use the Designer drag-and-drop interface. Additionally, this exercise contains the install code that will create the base for the reward levels. The upgrade code is run to upgrade the extension to a newer version and it will change the BRONZE level to ALUMINUM. Following all the steps of this walkthrough allows you to publish the extension on your tenant and create a possible new feature for your customers.

Reward table object

The following code adds a new table **50100 Reward** for storing the reward levels for customers. The table consists of three fields: **Reward ID**, **Description**, and **Discount Percentage**. For example, the **Description**

field must contain a value of type text and it cannot exceed the limit of 250 characters. The second field contains three properties that are used to set the range of the discount percentage assigned to every customer. Properties can be created for every field, depending on the scope.

> **TIP**
>
> Type `ttable` followed by the Tab key. This snippet will create a basic layout for a table object.

```
table 50100 Reward
{
    DataClassification = ToBeClassified;

    fields
    {
        // The "Reward ID" field represents the unique identifier
        // of the reward and can contain up to 30 Code characters.
        field(1;"Reward ID";Code[30])
        {
            DataClassification = ToBeClassified;
        }

        // The "Description" field can contain a string
        // with up to 250 characters.
        field(2;Description;Text[250])
        {
            // This property specified that
            // this field cannot be left empty.
            NotBlank = true;
        }

        // The "Discount Percentage" field is a Decimal numeric value
        // that represents the discount that will
        // be applied for this reward.
        field(3;"Discount Percentage";Decimal)
        {
            // The "MinValue" property sets the minimum value for the "Discount Percentage"
            // field.
            MinValue = 0;

            // The "MaxValue" property sets the maximum value for the "Discount Percentage"
            // field.
            MaxValue = 100;

            // The "DecimalPlaces" property is set to 2 to display discount values with
            // exactly 2 decimals.
            DecimalPlaces = 2;
        }
    }

    keys
    {
        // The field "Reward ID" is used as the primary key of this table.
        key(PK;"Reward ID")
        {
            // Create a clustered index from this key.
            Clustered = true;
        }
    }
}
```

For more information about table properties, see Table Properties.

Reward card page object

The following code adds a new page **50101 Reward Card** for viewing and editing the different reward levels that are stored in the new **Reward** table. Pages are the primary object that a user will interact with and have a different behavior based on the type of page that you choose. The **Reward Card** page is of type Card and it is used to view and edit one record or entity from the **Reward** table.

> **TIP**
> Use the snippet `tpage, Page` to create the basic structure for the page object.

```
page 50101 "Reward Card"
{
    // The page will be of type "Card" and will render as a card.
    PageType = Card;

    // The page will be part of the "Tasks" group of search results.
    UsageCategory = Tasks;

    // The source table shows data from the "Reward" table.
    SourceTable = Reward;

    // The layout describes the visual parts on the page.
    layout
    {
        area(content)
        {
            group(Reward)
            {
                field("Reward Id";"Reward ID")
                {
                    // ApplicationArea sets the application area that
                    // applies to the page field and action controls.
                    // Setting the property to All means that the control
                    // will always appear in the user interface.
                    ApplicationArea = All;
                }

                field(Description;Description)
                {
                    ApplicationArea = All;
                }

                field("Discount Percentage";"Discount Percentage")
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}
```

For more information about the types of pages in AL, see Pages Overview.

Reward list page object

The following code adds the **50102 Reward List** page that enables users to view the contents of the **Reward** table and edit specific records by selecting them and viewing them in the **Reward Card** page.

TIP

Use the snippet `tpage, Page of type list` to create the basic structure for the page object.

```
page 50102 "Reward List"
{
    // Specify that this page will be a list page.
    PageType = List;

    // The page will be part of the "Lists" group of search results.
    UsageCategory = Lists;

    // The data of this page is taken from the "Reward" table.
    SourceTable = Reward;

    // The "CardPageId" is set to the Reward Card previously created.
    // This will allow users to open records from the list in the "Reward Card" page.
    CardPageId = "Reward Card";

    layout
    {
        area(content)
        {
            repeater(Rewards)
            {
                field("Reward ID";"Reward ID")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the level of reward that the customer has at this point.';
                }

                field(Description;Description)
                {
                    ApplicationArea = All;
                }

                field("Discount Percentage";"Discount Percentage")
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}
```

After you have created the objects, update the **startupObjectId** in the `launch.json` file to 50102, the ID of the **Reward List** page and select the Ctrl+F5 shortcut to see the new page in your sandbox environment. You will be asked to sign in to your Business Central, if you have not already done so.

TIP

Information about your sandbox environment and other environments is stored as configurations in the `launch.json` file. For more information, see JSON Files.

Designer

Dynamics 365 Business Central Designer works in the browser and allows modifying the current page. It enables users to add existing table fields, move fields around, or remove fields from the page. Users can make changes to display the information they need, where they need it by using drag-and-drop components.
To show how the Designer changes the design of a page, you begin by adding two new fields to the **Reward**

table. These fields will be used later on to exemplify the Designer's properties.

```
field(4;"Minimum Purchase";Decimal)
{
    MinValue = 0;
    DecimalPlaces = 2;
}

field(5;"Last Modified Date";Date)
{
    // The "Editable" property sets a value that indicates whether the field can be edited
    // through the UI.
    Editable = false;
}
```

The **Last Modified Date** field requires constant changes to remain accurate. To keep it updated, triggers will be used. Triggers are predefined methods that are executed when certain actions happen. They are added by default when you use the `ttable` template, but you can also use the `ttriger` snippet to add them manually. Now you can add code to the triggers.

```
    // "OnInsert" trigger executes when a new record is inserted into the table.
    trigger OnInsert();
    begin
        SetLastModifiedDate();
    end;

    // "OnModify" trigger executes when a record in the table is modified.
    trigger OnModify();
    begin
        SetLastModifiedDate();
    end;

    // "OnDelete" trigger executes when a record in the table is deleted.
    trigger OnDelete();
    begin
    end;

    // "OnRename" trigger executes when a record in a primary key field is modified.
    trigger OnRename();
    begin
        SetLastModifiedDate();
    end;

    // On the current record, the value of the "Last Modified Date" field to the current
    // date.
    local procedure SetLastModifiedDate();
    begin
        Rec."Last Modified Date" := Today();
    end;
```

From this point, changes to the **Reward Card** page can be done either manually by adding the code below in Visual Studio Code or by using the Designer's functions to add the same fields. Both ways lead to the same results, but the Designer speeds up the process.

```
field("Minimum Purchase";"Minimum Purchase")
{
    ApplicationArea = All;
}

field("Last Modified Date";"Last Modified Date")
{
    ApplicationArea = All;
}
```

Using the F6 key shortcut in Visual Studio Code launches the browser and enters the Designer. You can also use the Designer from the Business Central client, by selecting  **Designer**.

> **NOTE**
> Every time you start designing, you create a new extension and the changes you make in the Designer will apply to all users.

To add the same fields and customize the **Reward Card** page, follow the next steps:

- Navigate to the **Reward Card** page by choosing **+ new**.
- Enter the Designer mode from the UI and select **More** from the Designer bar.
- Select **Field** from the Designer bar to show the list of available fields.
- Drag the **Minimum Purchase** and **Last Modified Date** fields from the list onto the page in the **Reward group**.
- Choose the **Reward** in the group caption to enable the value to be edited. Change the caption to **Info** and press **Enter**.

After making these adjustments, finish up your design by choosing **Stop Designing**, which allows you to name the extension with an option to download code, and save the extension for the tenant. If you choose not to download the code at the end, you can still pull the changes via the **Alt+F6** key shortcut from Visual Studio Code. You can also uninstall the extension by opening the **Extension Management** page.
For more information about Designer, see Designer.

Customer table extension object

The **Customer** table, like many other tables, is part of the Dynamics 365 Business Central service and it cannot be modified directly by developers. To add additional fields or to change properties on this table, developers must create a new type of object, a table extension. The following code creates a table extension for the **Customer** table and adds the `Reward ID` field.

> **TIP**
> Use the snippet `ttableext` to create a basic structure for the table extension object.

```
tableextension 50103 "Customer Ext" extends Customer
{
    fields
    {
        field(50100;"Reward ID";Code[30])
        {
            // Set links to the "Reward ID" from the Reward table.
            TableRelation = Reward."Reward ID";

            // Set whether to validate a table relationship.
            ValidateTableRelation = true;

           // "OnValidate" trigger executes when data is entered in a field.
            trigger OnValidate();
            begin

            // If the "Reward ID" changed and the new record is blocked, an error is thrown.
                if (Rec."Reward ID" <> xRec."Reward ID") and
                    (Rec.Blocked <> Blocked::" ") then
                begin
                    Error('Cannot update the rewards status of a blocked customer.')
                end;
            end;
        }
    }
}
```

Customer card page extension object

A page extension object can be used to add new functionality to pages that are part of the Dynamics 365 Business Central service. The following page extension object extends the **Customer Card** page object by adding a field control, **Reward ID**, to the **General group** on the page. The field is added in the layout section, while in the actions section the code adds an action to open the **Reward List** page.

> **TIP**
> Use the shortcuts `tpageext` to create the basic structure for the page extension object.

```
pageextension 50104 "Customer Card Ext" extends "Customer Card"
{
    layout
    {
        // The "addlast" construct adds the field control as the last control in the General
        // group.
        addlast(General)
        {
            field("Reward ID";"Reward ID")
            {
                ApplicationArea = All;

                // Lookup property is used to provide a lookup window for
                // a text box. It is set to true, because a lookup for
                // the field is needed.
                Lookup = true;
            }
        }
    }

    actions
    {
        // The "addfirst" construct will add the action as the first action
        // in the Navigation group.
        addfirst(Navigation)
        {
            action("Rewards")
            {
                ApplicationArea = All;

                // "RunObject" sets the "Reward List" page as the object
                // that will run when the action is activated.
                RunObject = page "Reward List";
            }
        }
    }
}
```

At this point, reward levels can be created and assigned to customers. To do that, update the `startupObjectId` value in launch.json to 21 and select the Ctrl+F5 key to open the page.

Help links

This app is relatively straightforward, but we want users of your app to be able to get unblocked and learn more just like all other users of Business Central. First, configure your app to get context-sensitive links to Help, and then apply tooltips to the fields in your pages.

Configure context-sensitive links to Help

At an app level, you can specify where the Help for your functionality is published in the app.json file. Then, for each page in your app, you specify which Help file on that website is relevant for that particular page. For more information, see Configure Context-Sensitive Help.

Open the app.json file, and then change the value of the `contextSensitiveHelpUrl` property to point at the right location on your website. In this example, you publish Help for your app at *https://mysite.com/documentation*.

```
"contextSensitiveHelpUrl": "https://mysite.com/documentation/",
```

Next, you set the ContextSensitiveHelpPage property for the **Reward Card** and **Reward List** pages:

```
    // The target Help topic is hosted on the website that is specified in the app.json file.
    ContextSensitiveHelpPage = 'sales-rewards';
```

The following example illustrates the properties for the **Reward List** page after you have specified the context-sensitive Help page.

```
page 50102 "Reward List"
{
    // Specify that this page will be a list page.
    PageType = List;

    // The page will be part of the "Lists" group of search results.
    UsageCategory = Lists;

    // The target Help topic is hosted on the website that is specified in the app.json file.
    ContextSensitiveHelpPage = 'sales-rewards';

    // The data of this page is taken from the "Reward" table.
    SourceTable = Reward;

    // The "CardPageId" is set to the Reward Card previously created.
    // This will allow users to open records from the list in the "Reward Card" page.
    CardPageId = "Reward Card";

...
```

You can specify the same relative link for **Reward Card**, **Reward List**, and the customization of the **Customer** page, or you can specify different targets. For more information, see Page-level configuration.

Add tooltips

Even the best designed user interface can still be confusing to some. It can be difficult to predict specifically what users will find confusing, and that is why the base application includes tooltips for all controls and actions. For more information, see Help users get unblocked.

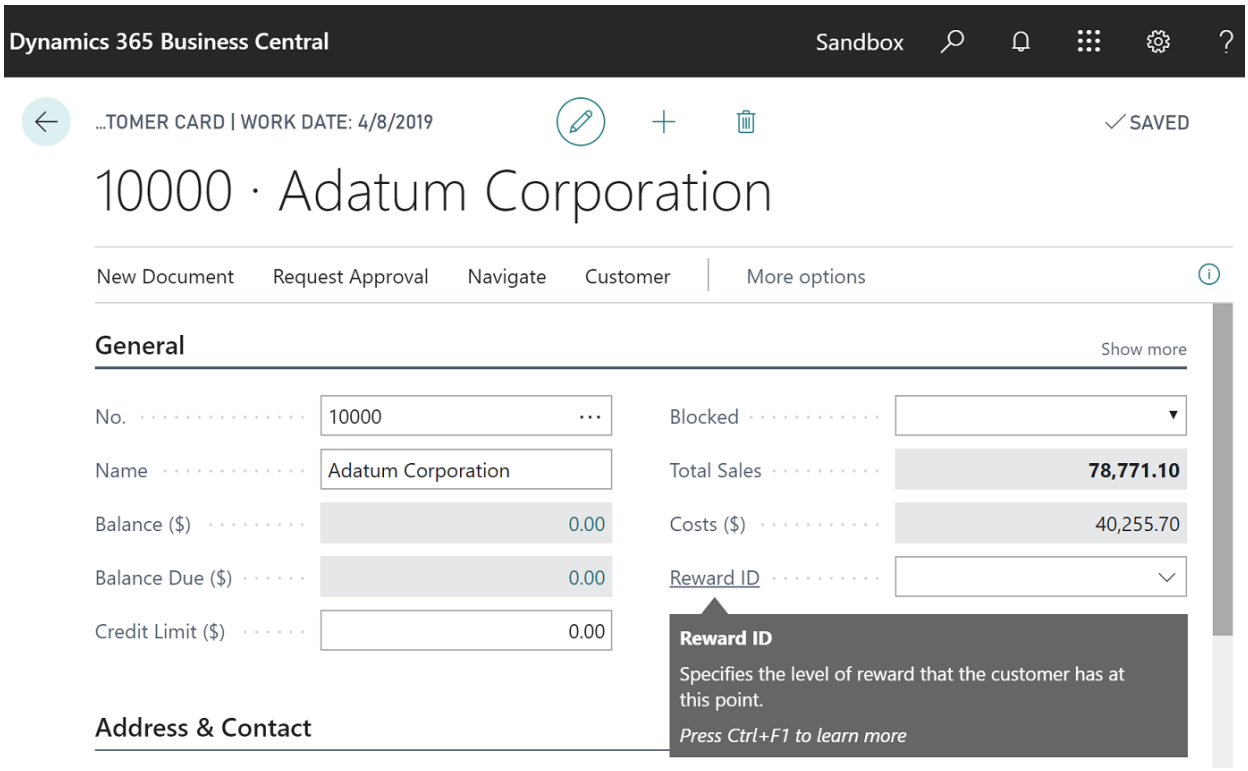
For the purposes of this walkthrough, add the following tooltip to the properties of the **Reward ID** field on all three pages:

```
ToolTip = 'Specifies the level of reward that the customer has at this point.';
```

The following example illustrates the tooltip:

```
field("Reward ID";"Reward ID")
{
ApplicationArea = All;
ToolTip = 'Specifies the level of reward that the customer has at this point.';
}
```

If you now deploy the app, you will be able to read the tooltip text for the **Reward ID** field, and if you choose the *Learn more* link or press Ctrl+F1, a new browser tab opens the equivalent of `https://mysite.com/documentation/sales-rewards`.



Install code

After installing the extension, the **Reward List** page is empty. This is the result of the fact that the **Reward** table is also empty. Data can be entered manually into the **Reward** table by creating new records from the **Reward List** page. However, this task slows down the process, especially because the **Reward** table should be initialized with a standard number of reward levels when the extension is installed. To solve this, install codeunits can be used. A codeunit is an object that can be used to encapsulate a set of related functionality represented by procedures and variables. An install codeunit is a codeunit with the Subtype property set to Install. This codeunit provides a set of triggers that are executed when the extension is installed for the first time and when the same version is re-installed.

In this example, the following install codeunit initializes the **Reward** table with three records representing the 'GOLD', 'SILVER', and 'BRONZE' reward levels.

> **TIP**
> Use the shortcuts `tcodeunit` to create the basic structure for the codeunit.

```
codeunit 50105 RewardsInstallCode
{
    // Set the codeunit to be an install codeunit.
    Subtype = Install;

    // This trigger includes code for company-related operations.
    trigger OnInstallAppPerCompany();
    var
        Reward : Record Reward;
    begin
        // If the "Reward" table is empty, insert the default rewards.
        if Reward.IsEmpty() then begin
            InsertDefaultRewards();
        end;
    end;

    // Insert the GOLD, SILVER, BRONZE reward levels
    procedure InsertDefaultRewards();
    begin
        InsertRewardLevel('GOLD', 'Gold Level', 20);
        InsertRewardLevel('SILVER', 'Silver Level', 10);
        InsertRewardLevel('BRONZE', 'Bronze Level', 5);
    end;

    // Create and insert a reward level in the "Reward" table.
    procedure InsertRewardLevel(ID : Code[30]; Description : Text[250]; Discount : Decimal);
    var
        Reward : Record Reward;
    begin
        Reward.Init();
        Reward."Reward ID" := ID;
        Reward.Description := Description;
        Reward."Discount Percentage" := Discount;
        Reward.Insert();
    end;

}
```

For more information about install code, see Writing Extension Install Code.

Upgrade code

When you upgrade an extension to a newer version, if any modifications to the existing data are required to support the upgrade, you must write upgrade code in an upgrade codeunit. In this example, the following upgrade codeunit contains code that changes the BRONZE reward level to customer records to ALUMINUM. The upgrade codeunit will run when you run the Start-NAVAppDataUpgrade cmdlet.

> **IMPORTANT**
>
> Remember to increase the `version` number of the extension in the app.json file.

```
codeunit 50106 RewardsUpgradeCode
{
    // An upgrade codeunit includes AL methods for synchronizing changes to a table definition
    // in an application with the business data table in SQL Server and migrating existing
    // data.
    Subtype = Upgrade;

    // "OnUpgradePerCompany" trigger is used to perform the actual upgrade.
    trigger OnUpgradePerCompany();
    var
        InstallCode : Codeunit RewardsInstallCode;
        Reward : Record Reward;

        // "ModuleInfo" is the current executing module.
        Module : ModuleInfo;
    begin
        // Get information about the current module.
        NavApp.GetCurrentModuleInfo(Module);

        // If the code needs to be upgraded, the BRONZE reward level will be changed into the
        // ALUMINUM reward level.
        if Module.DataVersion.Major = 1 then begin
            Reward.Get('BRONZE');
            Reward.Rename('ALUMINUM');
            Reward.Description := 'Aluminum Level';
            Reward.Modify();
        end;
    end;
}
```

For more information about writing and running upgrade code, see Upgrading Extension.

Instrumenting your app with telemetry

Business Central emits telemetry data for several operations that occur when extension code is run. Create an Application Insights resource in Azure if you don't have one. For more information, see Create an Application Insights resource. Now, add the Application Insights Key to the extension manifest (app.json file):

```
"applicationInsightsKey":["<instrumenation key>"]
```

Replace `<instrumenation key>` with your key.

You can configure your extension to send this data to a specific Application Insights resource on Microsoft Azure. For more information, see Sending Extension Telemetry to Azure Application Insights.

Conclusion

This walkthrough demonstrated how an extension can be developed. The main AL objects and extension objects were used to store the reward levels, to view, and to edit them. The Designer was introduced as an alternative to modify visual aspects of page objects and to customize them from the web client instead of using code. Up to this point, the table and the page objects were empty, but the install codeunits were added and allowed to initialize the **Reward** table with a standard number of reward levels when the extension was installed. An upgrade code section was also included in this exercise to create a full picture of all processes involved when an extension is built. As a result, a user is enabled to assign one of the three reward levels to a customer and to change this scenario by upgrading the version of the extension.

TIP

To try building a more advanced Customer Rewards sample extension, see Building an Advanced Sample Extension.

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension
Converting Extensions V1 to Extensions V2
Configure Context-Sensitive Help
Sending Extension Telemetry to Azure Application Insights

Using Designer

2/17/2021 • 9 minutes to read • Edit Online

When developing extensions in the AL development environment, you have a wide range of possibilities. Designer in Dynamics 365 Business Central complements the development experience in Visual Studio Code. It provides an easy and convenient way to make immediate changes to your design by dragging and dropping the components on the page.

IMPORTANT

Designer is currently only available from a Dynamics 365 Business Central sandbox.
Every time you start designing, you're effectively creating a new extension. Your changes are *immediately visible to other users* in the sandbox environment.
Designer *cannot be used by multiple users at the same time* in sandboxes.

NOTE

Extensions created using Designer are removed when the sandbox environment is updated or relocated within our service. Thus, you should not rely on using the sandbox environment as a source control for these Designer extensions, but remember to frequently download and backup the Designer extension source. For more information, see Production and Sandbox Environments.

Designer capabilities

Here is a quick overview of capabilities in **Designer**:

FEATURES	APPLIES TO
Add components	fields, columns, actions in navigation bar
Move components	fields, columns, cues, parts, actions, and action groups
Remove components	fields, columns, cues, parts, actions, and action groups
Hide and unhide components	parts
Change field importance, like showing in collapsed FastTab header or under **Show More**	fields
Exclude field from Quick Entry	fields, columns
Set freeze pane and clear freeze pane	columns
Adjust column width	columns
Edit caption	FastTabs, cards, FactBoxes
Save extension/download code	general

FEATURES	APPLIES TO
Preview design	general

Important points to note

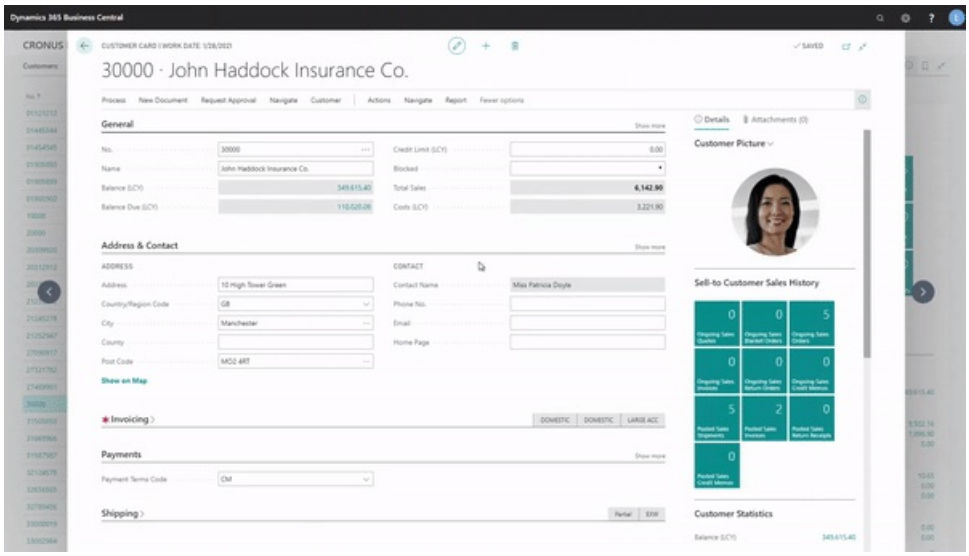
- Every time you start designing, you're effectively creating a new extension. Your changes are immediately visible to other users.
- The changes you make in Designer will apply to **all** users.
- You cannot remove specific fields that are bound to a page and a field must belong to an underlying table.
- You can only add fields, columns, or tiles to its applicable view from list, tall tiles, and wide tiles views. Some components can't be moved using drag-and-drop and are restricted to the view that they are in.
- You can only add fields/columns, from a predefined list, which is based on the source table. You can't create new ones.
- In the client, users can change many of these settings for their workspace only by using personalization. For more information, see Personalizing Your Workspace.

Start and stop designing

In the Business Central client, you start Designer by choosing **Designer** in the top-right corner of any page that you want to make modifications to, and start designing using drag-and-drop components. In Visual Studio Code, you can start Designer by using the **F6** shortcut, which launches a browser that opens the Business Central client in Designer.

After you are done with the changes, choose **Stop Designing**. You can name the extension and download code to save it for the tenant. Once you're done, the extension is automatically installed. If you choose to download the code, the project will be downloaded as a .zip folder. You extract the files and open the folder from Visual Studio Code, where you can deploy it as you would do with any other extension.

If you choose not to download the code at the end, you can still pull the code using the **Alt+F6** shortcut. You can also uninstall the extension from the **Extension Management** page or even download the source from there.



Drag-and-drop components

In Designer, you design and modify the current page. You can display existing table fields, move fields around, remove fields from the page, hide and move actions, and more. You can make changes to display the information by using drag-and-drop components.

Working with fields

To add a field or column to a page, in the banner, choose **More**, and then choose **Field**. A pane to the right appears that lets you add fields. Here you can see all of the table fields that are available for the specific page. The table fields displayed are based on the underlying table or tables. The field can have a status of **Placed**, which means that the field already exists on the page. A status of **Ready** means that the field doesn't already exist on the page. To add a field, drag and drop it to the wanted location.

If you want to remove a field or column, select the arrowhead indicator or on the component, and then choose **Remove**.

You can edit the caption of a FastTab for a group of fields by selecting the caption and start writing. A caption should provide a short and clear description.

Setting the freeze pane

Set freeze pane locks one or more columns to the left, even when you scroll horizontally. You can set the freeze pane, by selecting the arrowhead indicator or of the column that you want as the last column of the freeze pane, and then choose **Set Freeze Pane**. If you want to set the freeze pane back to its original designed location, select the arrowhead indicator or for the current freeze pane column, and then choose **Clear Freeze Pane**.

Setting the Importance on a field

Fields on non-list type pages, such as card and document type pages, include Designer options for setting the importance. The following table describes the options for setting the importance in Designer and how it corresponds to the Importance property in the page code.

OPTION	DESCRIPTION	IMPORTANCE PROPERTY VALUE
Show under "Show more"	Sets the field so that it appears only when the user selects **Show more**.	Additional
Show always	Sets the field to always display on the page. The field displays regardless of whether the user selects **Show more** or **Show less**. The field won't show in the FastTab heading if the FastTab is collapsed.	Standard
Show when collapsed	Sets the field to always display on the page. The field displays regardless of whether the user selects **Show more** or **Show less**. The fields will also display in the header of the FastTab if the FastTab is collapsed.	Promoted

Setting the Quick Entry on fields

You can use Designer to set the QuickEntry property on a field. The **QuickEntry** property determines whether the field is skipped when users press the **Enter** key to navigate through fields on a page. You use Quick Entry to help accelerate keyboard data entry by focusing only those fields a user typically needs to fill in.

To set the QuickEntry property from Designer, select the field or column heading, and then choose either **Include in Quick Entry** (sets the **QuickEntry** property to `true`) or **Exclude from Quick Entry** (sets the

QuickEntry property to `false`).

For more information about Quick Entry, from a user perspective, see Accelerating Data Entry Using Quick Entry in the Business Central Application Help.

Working with the Navigation Menu and Navigation Bar

Designer lets you add actions that link to pages and reports in the navigation bar using the bookmark icon on the target page or report. The bookmark icon is also available in the Tell Me window.

> **NOTE**
> Bookmarking is only available for pages and reports that are discoverable from Tell Me. For more information on how to make pages and reports searchable, see Adding Pages and Reports to Tell me.

This allows the user to efficiently create multiple links and build up the set of important or commonly used links for a specific profile in the Role Center navigation bar. For more information about bookmarking, see Bookmark a Link to a Page or Report on Your Role Center in the Business Central Application Help.

You can also move actions to reorder them in the navigation bar, or move them into groups or subgroups to design the Navigation Menu.

Hiding and showing Parts

Designer lets you hide and unhide part controls in a page directly from the client. When you select the **Hide** option on a part, it will be grayed out and will disappear from the page when you quit Designer mode. Similarly, you can select **Show** on a grayed part control to make it visible again to the user.

The extension generated when the user hides or shows a part overrides the Visible Property in code. For more information on how the **Visible Property** affects the visibility of a control, see UI Customization and Visibility of Controls.

Working with Actions

Designer lets you make changes to the actions that are defined in the action bar of a page. You can move, remove, hide, and show individual actions or action groups.

> **NOTE**
> You cannot modify actions that are defined on pages that are shown in parts, such as in FactBoxes or embedded lists.

> **TIP**
> In Designer, to run an action as normal, select the action and press **Ctrl+Click**.

Remove, hide, and show actions and groups

Actions and actions groups that are already hidden appear dimmed. To change the state of an action or action group, select it, and then choose one of the following options:

OPTION	WHAT IT DOES
Remove	This option is available for the actions that are shown only in a promoted category. Or actions that are shown in both a promoted category and another action menu. Choosing **Remove** deletes the action from the selected location so that it no longer appears. If the action is only shown in the promoted category, it will automatically be shown in the action menu where it's originally defined. You cannot remove actions on a Role Center page; you can only hide them.
Hide	This option is available for actions or action groups that currently are shown only in an action menu (not in a promoted category). Like **Remove**, choosing **Hide** will make the action or action group disappear from the action bar in the client. However, in Designer, the action or action group appears dimmed.
Show	This option appears if the action or action group has been previously hidden (dimmed). Choosing this option will make the action or action group appear in the action bar.

Move actions and action groups

Designer lets you move actions within the action bar. For example, you can:

- Move an action from an action menu to a promoted category.
- Move an action from one promoted category to another.
- Move an action within an action group or to a different action group.

To move an action or action group, drag and drop it to the wanted location, just like with fields and columns.

- You can move individual actions into the promoted categories, but you can't change the order of the actions in the category.
- You cannot move an action group into a promoted category.
- To move an action or action group into an empty action group, drag the action or action group to the target group and drop it in the **Drop an action here** box.
- On Role Center pages, you can't move actions among the different areas that are defined by the `area(creation)`, `area(processing)`, and `area(reporting)` controls.

Preview design on different display targets

The display type icons let you preview the changes you made on desktop, tablet, and phone clients. This way you can make sure that your design will work on the intended display target(s). You can flip to display tablet and phone designs in portrait and landscape orientation.

Controlling user access to Designer

Accessing Designer is controlled on a user or user group basis by the **D365 EXTENSION MGT** permission set. If a user is assigned this permission set, then Designer is available for the user in the client. To prohibit a user

from using Designer, just remove the user from the **D365 EXTENSION MGT** permission set.

> **NOTE**
>
> It is important that the **D365 EXTENSION MGT** permission set does not have a *company* specified; otherwise the user will not be able to access Designer.

See Also

Developing Extensions
Getting Started with AL
AL Development Environment

Keyboard Shortcuts

2/17/2021 • 2 minutes to read • Edit Online

The following table provides an overview of some of the shortcut key combinations that you can use when you are working in Visual Studio Code. For a complete overview, see Key Bindings for Visual Studio Code.

General in Visual Studio Code

KEYBOARD SHORTCUT	ACTION
Ctrl+Shift+P	Show All Commands
Alt+F6	Download source code
Alt+A Alt+L	AL Go! Generates a HelloWorld project
Ctrl+Shift+B	Package
F5	Publish
Ctrl+F5	Build and publish without debugging. **Note:** The keyboard shortcut has a different meaning when debugging.
F6	Publish and open Designer
Ctrl+F2	Update the compiler used by the service tier(s)

Editing in Visual Studio Code

KEYBOARD SHORTCUT	ACTION
Ctrl+Space	Look up suggestions for the current object
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Ctrl+F2	Select all occurrences
F12	Go to definition
Alt+F12	Peek definition
Shift+F12	Show references
Ctrl+Shift+Space	Look up parameter hints

KEYBOARD SHORTCUT	ACTION
Ctrl+K Ctrl+C	Add line comment
Ctrl+K Ctrl+U	Remove line comment
Ctrl+Shift+P	Show all commands
F2	Rename (use **Enter** to rename, use **Shift+Enter** to preview)
Shift+Alt+E	Look up events and insert event subscriber in code for a selected event.

Errors in Visual Studio Code

KEYBOARD SHORTCUT	ACTION
F8	Move to the next error or warning
Shift+F8	Move to the previous error or warning

Compile in Visual Studio Code

KEYBOARD SHORTCUT	ACTION
Ctrl+Shift+B	Compile and build the solution
Ctrl+F5	Build and deploy

Debugging in Visual Studio Code

For topics on debugging in AL, see Debugging and Snapshot Debugging.

KEYBOARD SHORTCUT	ACTION
F5	Start debugging session
Ctrl+F5	Publish without building. **Note:** The keyboard shortcut has a different meaning when not debugging.
Ctrl+Alt+F5	Start RAD publishing without debugging
Alt+F5	Start RAD with debugging
F7	Start a snapshot debugging session
Shift+F7	List all available snapshots
Alt+F7	Finish a snapshot debugging session

See Also

Developing Extensions
Getting Started with AL
AL Development Environment

The AL Formatter

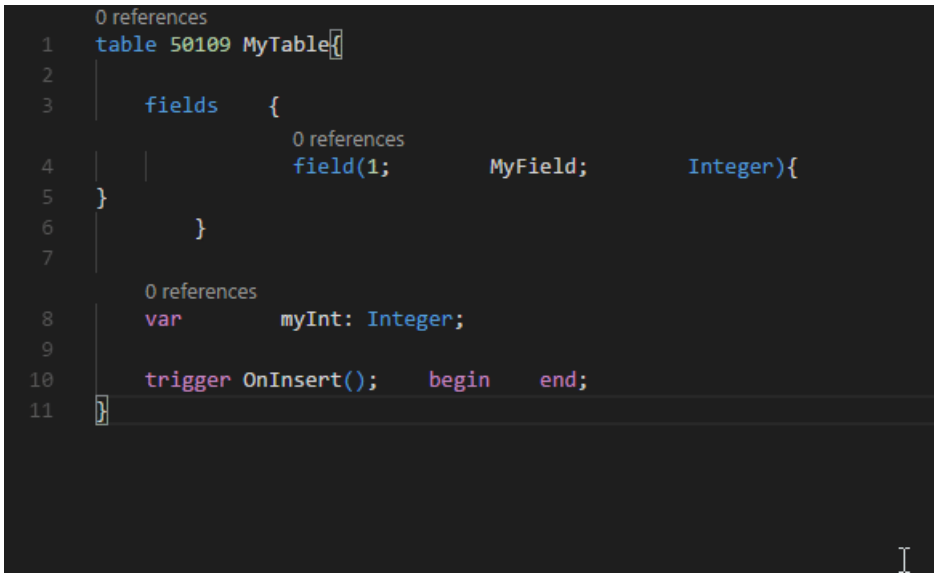
2/17/2021 • 2 minutes to read • Edit Online

The AL Language extension offers users the option to automatically format their source code. This capability increases the usability of the editor by allowing developers to instantly fix the indentation and formatting of their code. The auto-formatter analyzes the syntax tree of the AL code that you are formatting and, using rules developed based on the coding and style guidelines for AL, inserts and removes whitespace from key points in the document to make it more readable.

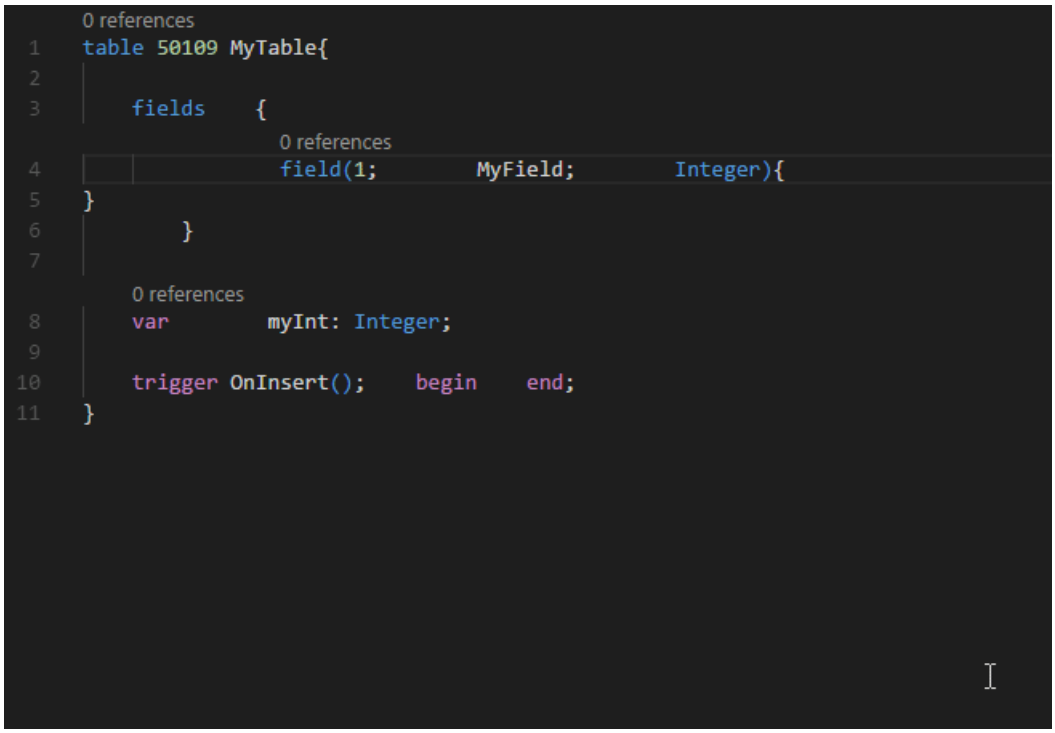
> **NOTE**
>
> The rules used by the auto-formatter cannot be configured by the user. This limitation is present to allow for a uniform style to be used throughout the community of AL developers.

Invoking the AL formatter

The auto-formatter can be invoked to format an entire AL document or a pre-selected range. In an existing project, open the document that you want to format, right-click inside the document, and select **Format Document**. In the default configuration for Visual Studio Code, the command can be run using the shortcut **Alt+Shift+F**.



To format a range, in an already opened project, open the document that you want to modify, select the specific range to format, right-click, and select **Format Selection**. In the default configuration for Visual Studio Code, the command can be run using the shortcut **Ctrl+K**, **Ctrl+F**.



See Also

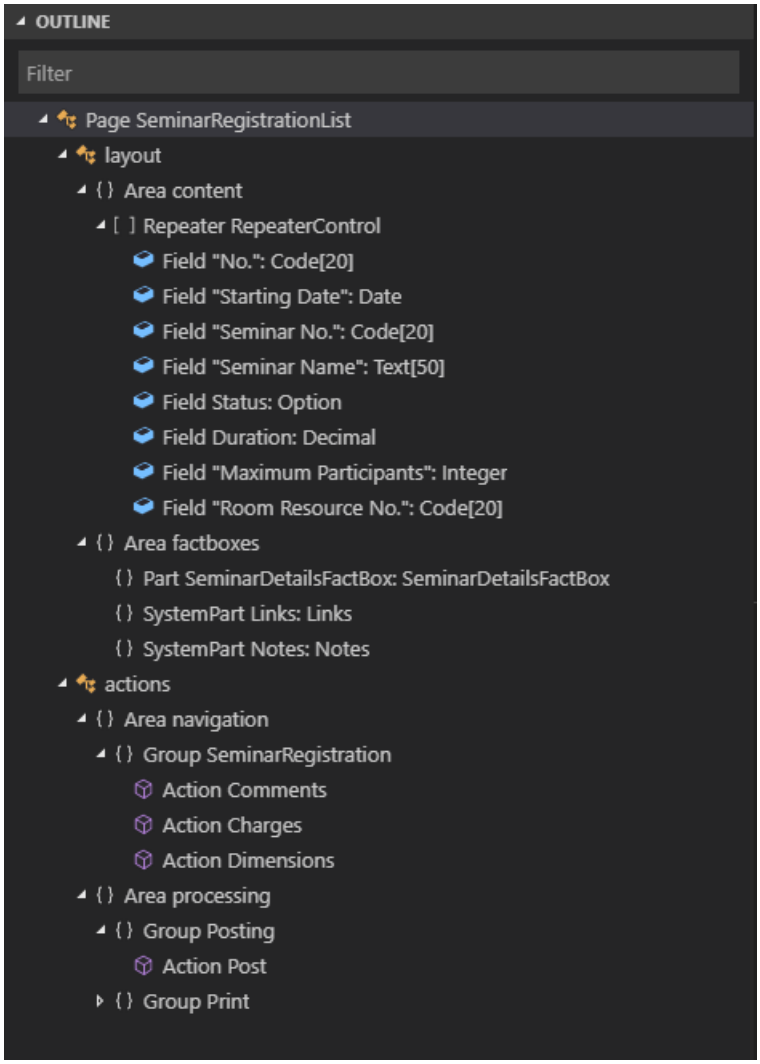
AL Development Environment
AL Outline View
AL Code Actions

AL Outline View

2/17/2021 • 2 minutes to read • Edit Online

Working with the AL Language extension you have access to the **Outline** view. The **Outline** view is a separate section in the lower left corner, right under the **Explorer** view.

The **Outline** view is enabled by default and shows the symbol tree of the currently active cursor, it also allows you to filter as you type. Double-clicking on any node makes your cursor jump to the selected definition or keyword. The **Outline** view will also display any errors in your project for easy inspection.



You manage the look and feel of the **Outline** view by defining a number of settings, that are all enabled by default. To set these, press **Ctrl+Shift+P**, and then choose **Preferences: Open Settings (UI)** for workspace settings, or choose **Preferences: Open User Settings** for user settings. Under **Extensions**, and **AL Language extension configuration** you will find the settings that are available for the AL Language extension for the `settings.json` file.

- `outline.icons` - Outline elements displayed with icons
- `outline.problems.enabled` - Show errors and warnings on outline elements
- `outline.problems.badges` - Badges displayed for errors and warnings
- `outline.problems.colors` - Colors used for errors and warnings

See Also

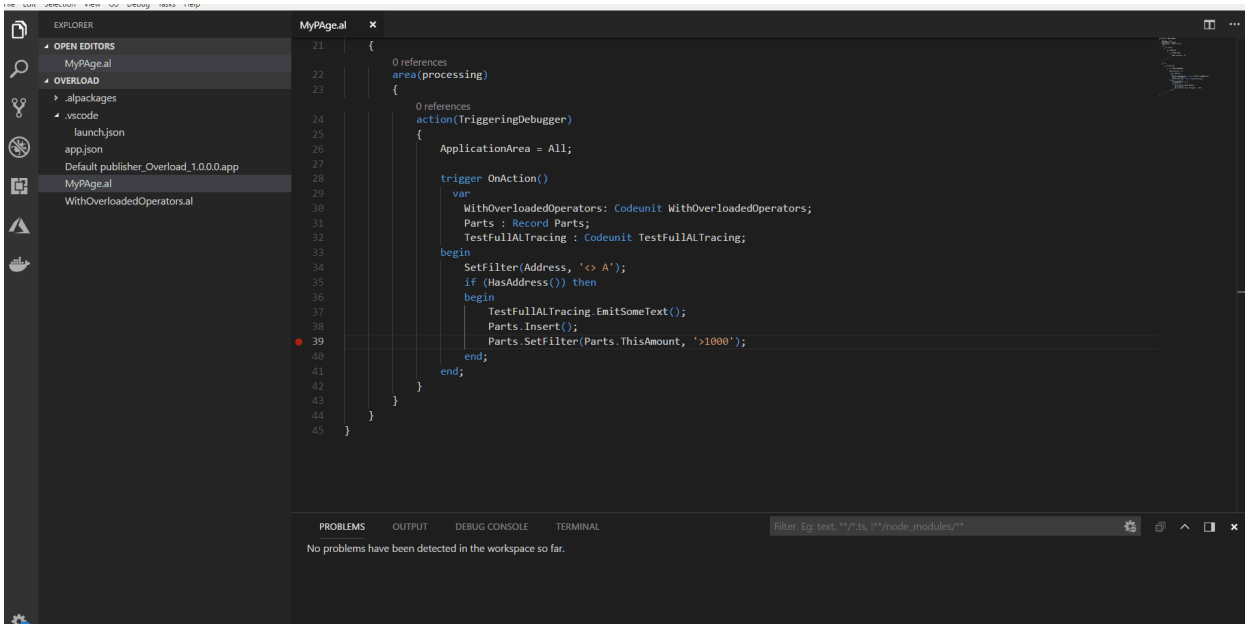
AL Code Navigation

2/17/2021 • 2 minutes to read • Edit Online

When you develop an AL extension, you may want to navigate around the source code frequently. To jump around the code or to access the reference code, you use the Go To Definition feature in Visual Studio Code.

Go To Definition

The **Go to Definition** feature navigates to the source of a type and opens the result in a new tab. You can use the `F12` shortcut key or right-click and select the **Go to Definition** feature from the right-click menu. The **Go to Definition** opens the source in the `.dal` format which contains the base application code. For example, the base application code may contain table metadata and application methods. In the following illustration, the Address type and the HasAddress type opens the `Customer.dal` file and locates the reference code of those types by using the Go To Definition feature.



With **Go to Definition**, you can step into the referenced code and set breakpoints on the external code and base application code. For more information, see Debugging in AL.

You can always use **Go to Definition** on Dynamics 365 Business Central code. However, if you want to use it on other extensions, the extension package which is now referenced, when originally published, must have the `showMyCode` property set to `true`. For example, if A is referencing B you can only use the Go To Definition on types of B, if B, when it was published, had the `showMyCode` flag set to `true`. For more information, see Security Setting and IP Protection.

Runtime 5.2 and Go to Definition

From runtime 5.2 and onwards, **Go to Definition** will resolve sources from the downloaded application dependency. You can navigate from within a symbol source file (.dal) to another symbol source file. For example, you can use **Go to Definition** from the `Customer` source DAL file in the Base App to the `Customer List` DAL source also defined in the Base App.

The following conditions still apply:

- **Go to Definition** is forward only. This is due how Visual Studio Code handles preview documents (DAL

files). There is no backward navigation support for preview files within Visual Studio Code. This means that if you navigate from your AL file to the `Customer` DAL source, and from there to the `Customer List` DAL source, and you issue a backward navigation (**Alt+Left arrow**), you will get back to the AL file and not what you would have expected; the `Customer` DAL source.

- Transitive references can only be resolved if the symbol app that defines the reference is a dependency on the project that contains the entry point for the **Go to Definition** symbol. For example, assume that you are in `HelloWorld.al` and want to **Go to Definition** on the `Car` table defined in the `Car.app` which is a dependency on your app. Then navigation will open the `CarTable.dal` preview file. And assume that from here you want to **Go to Definition** on `CarDistributor` table defined in the `CarDistributor.app` which is a dependency on `Car.app`, but *not* a dependency on the `HelloWorld.app`. In this case the source code **Go to Definition** will not work.

For more information about code navigation in Visual Studio Code, see Code Navigation.

See Also

Developing Extensions in AL
JSON Files
Debugging in AL
AL Code Actions

AL Code Actions

2/17/2021 • 2 minutes to read • Edit Online

The AL Language extension offers users the option to help fix issues in code. **Code Actions** is a Visual Studio Code feature providing the user with possible corrective actions right next to an error or warning. If actions are available, a light bulb appears next to the error or warning. When the user clicks the light bulb (or presses **Ctrl+.**), a list of available code actions is presented.

In AL Language extension these code actions are available in the current version:

- Multiple IF to CASE converting code action.
- Spell check code action.
- Interface implementer.
- Make method local.
- Use parenthesis for method call.
- Fix explicit `with` statements.
- Fix implicit with statements.

To enable AL Code Actions

1. Open the Command Palette by pressing **Ctrl+Shift+P** and then open the `settings.json` file.
2. Enter the setting `al.enableCodeActions` and set it to `true` like this `"al.enableCodeActions": true`
3. Save the settings file. You have now enabled code actions on your project.

Alternatively:

1. Open the Settings Page, **Ctrl+,** and choose either **User Settings** or **Workspace Settings** depending on which scope you want the code actions to apply to.
2. Navigate to **Extensions > AL Language extension configuration**.
3. Click on the **Enable Code Actions** checkbox. You have now enabled code actions on your project.

See Also

AL Development Environment
AL Outline View
AL Formatter
Directives in AL

Object Ranges in Business Central

2/17/2021 • 2 minutes to read • Edit Online

When you develop an app for Business Central online, you must request an object range in terms of licensing. Developing for Business Central is done using Visual Studio Code with the AL Language extension.

There are currently two available ranges that you can request. Both have some characteristics to keep in mind:

- *RSP Object Range* (ID range 1,000,000-69,999,999)

  This object range is tied to the RSP Program details.

  > **IMPORTANT**
  > We currently advise new publishers to *not* request an RSP object range

- *App Object Range* (ID range 70,000,000-74,999,999)

  This object range was originally designed just for apps in the Microsoft commercial marketplace to be used in Business Central online.

  > **IMPORTANT**
  > We currently advise new publishers to request an app object range.

Currently, you can implement apps developed in both the *RSP range* and the *app object range* in Business Central online and on-premises, as well as partner-hosted.

For more information, see Requesting an object range.

The following sections describe the different object ranges that you can find in the base application and extensions.

0-49,999

This range is assigned to Business Central base app functionality and must not be used in extensions or customizations.

50,000-99,999

This range is for customizations, and for test purposes. For Business Central online, a partner can develop an extension tailored to the individual tenant to fit the needs. The partner will develop this either by using a sandbox tenant or by obtaining a Docker image. Once the development is done, the extension can be deployed to the individual tenant.

Also, use this range as part of training and similar, such as if you are using a sandbox tenant or a build of Business Central on Docker.

100,000-999,999

The objects in this range are mainly designed when the Microsoft team localizes Business Central for a specific country or region. These objects cannot be used by partners.

1,000,000-69,999,999

This object range is intended for the Registered Solution Program (RSP). The partner can choose to use this range for developing extensions that can be used in Business Central online or on-premises. When used in Business Central online, these extensions are obtained as apps from appsource.microsoft.com.

70,000,000-74,999,999

Partners can obtain IDs in this range for extensions for Business Central online. These extensions are obtained as apps from appsource.microsoft.com.

For more information, see Get Started with Building Apps.

Download the Business Central licensing guide here.

See Also

Get Started with Building Apps
Getting Started with AL
Blog Post

Differences in the Development Environments

2/17/2021 • 3 minutes to read • Edit Online

Coming from the Dynamics NAV Development Environment and C/SIDE, there are some differences and optimizations that you should familiarize yourself with. The following sections go through some of these changes, but is in no way an exhaustive list.

> **TIP**
> A very useful tool working in Visual Studio Code, is IntelliSense which gives you a list of options in the current context. To activate IntelliSense from anywhere in the code, press **Ctrl+Shift**.

General development approach

While you could change the existing Dynamics NAV source code using C/SIDE, the best approach to developing with the AL Language development environment is to work with Extensions only. That means that you can extend the existing code by adding new functionality or integrating your code using Events, but it is not possible to change existing code, for example in codeunits.

With Business Central 2019 release wave 2, the possibility to change or "code-customize" the base application was introduced for on-premises installations, but at the same time it was announced that at some point in the future, the extension model will be the only option and code-customization will no longer be possible.

Data types

C/SIDE	AL LANGUAGE DEVELOPMENT ENVIRONMENT
Dates are parsed based on culture settings.	Locale independent and supports only: `yyyy-mm-ddD`.
Boolean values could be expressed as **yes/no**.	Boolean values are expressed as **true/false**.
For tables, integers could allow decimal values. For example, 5.0 converts to an integer, 5.4 throws an error at runtime.	For tables, Min, Max, InitValue numbers with a fraction are expressed as `decimal`, thus they are not a valid integer data type.
The largest constant integer could be `999999999999999`.	Transforms to `999'999'999'999'999.0`, a decimal value. In AL, this can be expressed as `999999999999999.0` or `999999999999999L`.

Syntax updates

C/SIDE	AL LANGUAGE DEVELOPMENT ENVIRONMENT
The token for multilanguage comment is @@@.	A multilanguage comment is marked with `Comment`.

Several properties have been renamed, to mention some:

C/SIDE	AL LANGUAGE DEVELOPMENT ENVIRONMENT
AutoFormatExpr	AutoFormatExpression
DataCaptionExpr	DataCaptionExpression
Layout	GridLayout
ProviderID	Provider

> **NOTE**
> Property values are considered syntax elements; thus they should follow the standard AL escaping rules.

Multilanguage properties

With the introduction of .xliff files, the ML properties, such as **CaptionML** and **TooltipML** are not used for this translation method. Use the equivalent properties instead, such as **Caption** and **Tooltip**, then make sure the manifest is set up to generate the `/Translations` folder and use the generated .xliff files for translations of the extension. For more information, see Working with Translation Files.

Pages

The `ActionContainer` elements in AL have been renamed; the following table lists the renamed elements:

C/SIDE	AL LANGUAGE DEVELOPMENT ENVIRONMENT
ActionItems	`Processing`
ActivityButtons	`Sections`
HomeItems	`Embedding`
NewDocumentItems	`Creation`
RelatedInformation	`Navigation`
Reports	`Reporting`

For instance, `area(Sections)` can be defined inside the `actions` section of the page.

Likewise, `Container` and `ContainerType` elements in C/SIDE have been renamed to `area(Content|FactBoxes|RoleCenter)` and can be defined inside the `layout` section of the page.

> **NOTE**
> For backwards compatibility we continue to support adding non-part pages as parts. We do, however, recommend that you redesign your page to only use Card part or List part, as we may remove support in a future update.

For syntax examples, see Page Object and Page Extension Object.

Naming

Controls, actions, and methods names must be unique on pages. In C/SIDE, you could create a Part control with the same name as a method, which would give you an error at runtime. This is now prevented, by disallowing duplicates. Similarly, trigger and trigger event names are disallowed on matching application object types. Likewise, actions and fields could have same names before, but that would have prevented page testability access, and will now throw a compilation error.

> **NOTE**
> Name on Controls and Actions on Pages is now mandatory.

Property dependencies

Some properties require that you set another property. An example is `PromotedCategory`, which requires that you have enabled the property `Promoted`. The following table lists some of the properties that have this dependency.

PROPERTY	DEPENDS ON THE PROPERTY...
PromotedCategory	Promoted
PromotedIsBig	Promoted
ValidateTableRelation	TableRelation
SourceTableTemporary	SourceTable
RunPageMode	RunObject

Limited functionality

The `InitValue` property of type `Duration` is not allowed in the AL Language Development Environment. The `InitValue` of type `DateTime` only allows for the value `0DT`.

See Also

Developing Extensions
Getting Started with AL
AL Development Environment

Adding Help Links from Pages, Reports, and XMLports

2/17/2021 • 2 minutes to read • Edit Online

When creating new pages, you can specify which Help file to open if the user selects the *Learn more* links in the UI of Business Central.

The context-sensitive Help link is generated based on a configuration setting in the `app.json` file and the name of the relevant Help file that you specify as part of the metadata for the page object. For more information, see Configure Context-Sensitive Help.

Examples

The following examples show how you can specify the *ContextSensitiveHelpPage* property from new pages, reports, and XMLports:

```
page 50100 MyPageWithHelp
{
    ContextSensitiveHelpPage = 'sales-rewards';
}
```

```
report 50100 MyReportWithHelp
{
    requestpage
    {
        ContextSensitiveHelpPage = 'sales-rewards';
    }
}
```

```
xmlport 50100 XmlPortWithHelp
{
    requestpage
    {
        ContextSensitiveHelpPage = 'sales-rewards';
    }
}
```

In all three examples, the ContextSensitiveHelpPage property is set to point at the same Help files. This is because all three example objects support the same feature that is explained in the *sales-rewards* Help topic. In your app, you can choose to structure the Help differently.

See Also

Configure Context-Sensitive Help
Translating Base App Help
JSON Files
Page Object
Report Object
XMLport Object
Table Object

Working with Translation Files

2/17/2021 • 4 minutes to read • Edit Online

Dynamics 365 Business Central is multilanguage enabled, which means that you can display the user interface (UI) in different languages. In Dynamics 365 Business Central this is done using XLIFF files, which is a standardized format used for computer-based translations.

Generating the XLIFF file

To add a new language to the extension that you have built, you must first enable the generation of XLIFF files. The XLIFF file extension is .xlf. The generated XLIFF file contains the strings that are specified in properties such as **Caption**, **CaptionML**, and **Tooltip**.

> **NOTE**
> To submit an app to AppSource, you must use XLIFF translation files.

In the app.json file of your extension, add the following line:

```
"features": [ "TranslationFile" ]
```

Now, when you run the build command (**Ctrl+Shift+B**) in Visual Studio Code, a `\Translations` folder will be generated and populated with the .xlf file that contains all the labels, label properties, and report labels that you are using in the extension. The generated .xlf file can now be translated.

> **IMPORTANT**
> Make sure to rename the translated file to avoid that the file is overwritten next time the extension is built.

By setting the `GenerateCaptions` flag in the app.json file, you specify that you want to generate captions based on the object name for pages, tables, reports, XMLports, request pages, and table fields. If the object already has a `Caption` or `CaptionML` property set, that value will be used, for table fields the `OptionCaption` is used. The syntax is the following:

```
"features": [ "TranslationFile", "GenerateCaptions" ]
```

GenerateLockedTranslations

APPLIES TO: Business Central 2020 release wave 2 and later

By setting the `GenerateLockedTranslations` flag in the app.json file, you specify that you want to generate `<trans-unit>` elements for locked labels in the XLIFF file. The default behavior is that these elements are not generated. For more information, see JSON Files.

```
"features": [ "GenerateLockedTranslations" ]
```

Label syntax

The label syntax is shown in the example below for the **Caption** property:

```
Caption = 'Developer translation for %1',  Comment = '%1 is extension name', locked = false, MaxLength=999;
```

> **NOTE**
>
> The `comment`, `locked`, and `maxLength` attributes are optional and the order is not enforced. For more information, see Label Data Type.

Use the same syntax for report labels:

```
labels
{
  LabelName='LabelText',Comment='Foo',MaxLength=999,Locked=true;
}
```

And the following is the syntax for **Label** data types:

```
var
a:Label'LabelText',Comment='Foo',MaxLength=999,Locked=true;
```

> **IMPORTANT**
>
> The **ML** versions of properties are **not** included in the .xlf file:
>
> - CaptionML
> - ConstValueML
> - InstructionalTextML
> - OptionCaptionML
> - PromotedActionCategoriesML
> - RequestFilterHeadingML
> - ToolTipML
>
> The TextConst Data Type is not included in the .xlf file either.

The XLIFF file

In the generated .xlf file, you can see a `<source>` element for each label. For the translation, you will now have to add the `target-language` and a `<target>` element per label. The `<trans-unit id>` attribute corresponds to the object ID in the extension. This is illustrated in the example below.

```
<?xml version="1.0" encoding="utf-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-transitional.xsd">
  <file datatype="xml" source-language="en-US" target-language="da-DK" original="ALProject16">
    <body>
      <group id="body">
        <trans-unit id="PageExtension 50110" maxWidth="999" size-unit="char" translate="yes"
xml:space="preserve">
          <source>Developer translation for %1</source>
          <target>Udvikleroversættelse for %1</target>
          <note from="Developer" annotates="general" priority="2">%1 is extension name</note>
          <note from="Xliff Generator" annotates="general" priority="3">PageExtension - PageExtension</note>
        </trans-unit>
      </group>
    </body>
  </file>
</xliff>
```

NOTE

You can have only one .xlf file per language. If you translate your extension to multiple languages, you must have a translation file per language. There is no enforced naming on the file, but a suggested good practice is to name it `<extensionname>.<language>.xlf`.

When the extension is built and published, you change the language of Dynamics 365 Business Central to view the UI in the translated language.

Translating other extensions

To translate other extensions, for example, adding translations to the Base Application, you must reference the project to be translated using the `dependencies` section in the app.json file. For more information, see JSON Files. When you have the dependencies added, you can add xliff files in your current project that translates the object captions of the referenced extension. Create a directory named **Translations** in the root of the extension, and place the translated xliff file there. When your extension is then built and published, change the language of Dynamics 365 Business Central to view the UI in the translated language.

Translation and Localization apps

NOTE

The following section only applies to versions released before Business Central 2019 release wave 2.

The .xlf files approach cannot be used for translating the base application. If you are working on a translation or localization app (for example for a country/region localization), you must take the .txt file containing the base application translation, and place the file in the root folder of your extension. When the extension is compiled, the .txt file is then packaged with the extension.

We recommend that you use only one .txt file per language. There is no enforced naming on the .txt files, but a suggested good practice is to name it `<extensionname>.<language>.txt`.

For more information about importing and exporting .txt files, see How to: Add Translated Strings By Importing and Exporting Multilanguage Files in Dynamics NAV.

See Also

How to: Add Translated Strings By Importing and Exporting Multilanguage Files in Dynamics NAV
Working with labels
Working with multiple AL project folders within one workspace
JSON Files

Instrumenting an Application for Telemetry

2/17/2021 • 2 minutes to read • Edit Online

This article describes how you can implement custom telemetry signals in your application for emitting telemetry data.. This data can then be collected and visualized for analyzing the application against the desired business goals, troubleshooting, and more.

Telemetry overview

One aspect of event logging is collecting data about how the application and your deployment infrastructure is working in order to diagnose conditions and troubleshoot problems that affect operation and performance. For example, this type of event logging includes Business Central Server events and trace events like SQL and AL method (function) traces.

Another aspect of logging is *telemetry*, which is collecting data about how your application functions and how it is being used in production. Telemetry can tell you about specific activities that users perform within the application in the production environment. Telemetry is also a useful tool for troubleshooting, especially instances where you are not able to reproduce the conditions experienced by the user or have no access to the user's environment. Telemetry can be divided into different levels or categories, like: telemetry for engineering, telemetry about the business, telemetry for customers.

Creating custom telemetry signal

There are two different resources where telemetry trace signals can be sent for monitoring and analyzing: Event Log and Microsoft Azure Application Insights. By default, the Business Central application is instrumented to emit several system telemetry trace signals to these destinations. Custom telemetry trace signals enable you to send telemetry data from anywhere in the application code to either of these destinations.

The procedure for creating custom telemetry signals is different for each resource. Your choice might also depend on whether you are developing for Business Central online or on-premises.

REOURCE	DESCRIPTION	ONLINE	ON-PREMISES	MORE INFORMATION

REOURCE	DESCRIPTION	ONLINE	ON-PREMISES	MORE INFORMATION
Application Insights	Create custom telemetry signals that are sent to an Application Insights resource in Azure. Application Insights is a service hosted within Azure that gathers telemetry data for analysis and presentation. Extension developers can specify whether the signal is only sent to the extension publisher or also to the VAR partner telemetry resource. You create these custom trace signals by using the LOGMESSAGE method in code.	✓	✓	See...
Event Log	Create custom telemetry trace signals that are sent to the Event Log of the Business Central Server machine. You create these custom trace signals by using the SENDTRACETAG method in code.		✓	See...

NOTE

Using Application Insights is recommended.

See Also

Monitoring and Analyzing Telemetry
Monitoring Business Central Server Events

The SMB Opportunity for App Publishers

2/17/2021 • 4 minutes to read • Edit Online

Our mission is to empower every individual and every organization on the planet to achieve more. This is particularly key for the 78 million small and mid-sized businesses (SMB) worldwide that increasingly want focus on tech intensity to be more resilient, more capable of transforming their products and services to face the constant changes and challenges in the marketplace. To embrace growth, every organization will have to have a business application which breaks down the silos of data, processes and workflow in their operations. This enables employees to respond to daily challenges and opportunities with agility. Market research shows that the business applications opportunity for software companies in the SMB space is predicted to be 51 billion dollars by 2025. As a developer, you want to make sure you bet on the winning platform. The BizApps market growth in this area is 17%; however Dynamics 365 platform growth has been growing at 47%!

Dynamics 365 Business Central

Dynamics 365 Business Central is an all-in-one business management solution that connects operations for small to mid-sized business. It ensures business continuity with a cloud solution that connects sales, service, finance, and operations to help teams adapt faster and deliver result. Business Central provides app publishers with a modern business platform that you can easily connect to, extend and build on — with little to no code required.

Microsoft AppSource

Microsoft AppSource is part of Microsoft's commercial marketplace where customers can find, try and get business solutions apps. It is the launch pad for your joint go-to-market activities with Microsoft and a flywheel for business growth. Using launch promotion, demand generation, and joint sales and marketing, your offer portfolio on AppSource can be the centerpiece of your cloud business engine.

In November 2020, Microsoft AppSource had more than 3 million monthly active users and more than +20.000 apps. Specifically for Business Central, there are more than 1000 apps available, and the number is growing fast.

Go-to Market Scenarios

As an AppSource publisher you can focus on these scenarios with Business Central and Microsoft AppSource:

- Connect
- Extend, or
- Embed

Connect

Connect your existing online service with Business Central through a powerful API. Here are a few examples which of existing connect apps which you can find on AppSource:

- Square payments, which allow you take payments with a Square terminal.
- Certify which allows users to post expense reports using Certify.com
- Scaptify, which connects your Shopify store with Business Central

Learn more about the API to build connect apps: Getting Started Developing Connect Apps

Extend

You can extend the default capabilities in Business Central to add extra productivity features or industry functionality to fit the needs of your customer base. The possibilities are plentiful.

In countries where Business Central is not localized by Microsoft, you can extend Business Central based on local requirements to respond to the regulatory and or competitive needs of that market.

Here are a few examples of some apps that extend Business Central:

- E-Ship and E-Receive from Lanham Associates, which extends warehouse management with barcoding and labelling, scanning functionality and interfaces with carriers and weigh scales.
- Continia Document Capture, which is an end-to-end solution for document recognition, invoice approval and digital archiving
- Philippines localization from Pasi, which computes the withholding tax as mandated by the Philippine government for both customers and vendors.
- SwissSalary 365, which is a certified and flexible payroll app that's intuitive and easy to use for the Swiss market

Learn more on how to build your app: Getting Started with AL

Embed

Embed Business Central as an integral component in an end-to-end industry solution. Here are a few examples of embed solutions available on Microsoft AppSource:

- 4PS construct, an all-in one solution for construction. 4PS has specifically developed an integrated software solution based on the Microsoft platform and tailored for construction, civil engineering, mechanical and electrical, service and maintenance, house builders and equipment rental companies
- LS Central from LS Retail. The complete, unified commerce online platform to manage your entire retail and food service operations
- Dynamics Empire Online, cegeka-dsa's open, secure and innovative real estate solution.

Learn more about embed apps: Embed app overview

Consultancy services

An increasing number of business users find web more convenient for buying where they can research, deploy and manage apps. Therefore, next to providing Apps, a publisher can also market consultancy services on Microsoft AppSource to connect with buyers. The provided content in these offers could be assessments, briefings, workshops, proof of concepts and implementations. In general the services are typically fixed in scope and duration, they are offered at a fixed price or free and have a defined outcome.

Here are a few examples of Consultancy services provided by publishers:

- Unified Commerce Readiness Intro: 1-Hr Assessment In this 1 hour assessment, LS Retail assesses how a 4-day engagement helps them with implementing their Unified Commerce Cloud solution based on Microsoft Dynamics 365.
- NAV-X Commission Mgt Gold 4-Hr Implementation in this 4 hour implementation workshop, NAV-X supports the customers in implementing their commission management app.

Learn more about Consultancy Services.

See also

Get Started with Building Apps

Get Started with Building Apps

2/17/2021 • 8 minutes to read • Edit Online

Dynamics 365 Business Central is a business management solution that helps companies connect their financials, sales, services, and operations to streamline business processes, improve customer interactions and make better decisions. With this modern business platform, you can easily and quickly tailor, extend, and build applications so they fit your specific needs with little to no code development.

Build a line-of-business app, such as for a specific industry, process, or department such as HR, finance, marketing, or operations. Then, publish your app to the Microsoft commercial marketplace, where customers can find and try your app, and get in touch with you. For more information, see What is the Microsoft commercial marketplace?.

Learn how you can become a Business Central app publisher in six steps in this article.

Step 1: Become a partner

Becoming a Microsoft partner gives you access to the Microsoft resources needed to build, market, and sell your apps. You don't have to be a Microsoft partner to begin developing your apps. But all of the steps below are required to gain access to the programs that enable you to publish, market, and sell your apps for Business Central.

Obtain your work account

Your work account or work email is the email address provided to you by your company. This email is usually in the format `you@yourcompany.com`. More information on work accounts can be found here.

Join the Microsoft Partner Network

Microsoft Partner Network (MPN) membership unlocks our best resources to differentiate your business, take your product to market, and sell your solutions. To become a partner, you must join the Microsoft Partner Network (MPN), at which time you will be assigned an MPN ID. MPN membership is free to all partners; you can enroll in the MPN here.

Once signed up, you will get an MPN ID – your gateway to access all the membership resources and benefits for your partnership with Microsoft. There is no cost to obtain a MPN ID as a Network member, and with options to upgrade to an Action Pack subscription or work toward a competency, you can access even more benefits.

Set up your Partner Center account

Once you have joined the Microsoft Partner Network (MPN), you can set up your Partner Center (PC) account. The Microsoft Partner Center is a generic portal where partners can sell and manage customer subscriptions for Microsoft services, such as Microsoft 365, Azure, Dynamics 365, and others, as well as for some third-party products. For more information, see the Partner Center documentation.

Your Partner Center account provides you with access to pricing information, tools and services, and enables you to manage admin credentials for your company's work account. Partner Center is also where you can purchase or renew subscriptions to Microsoft Action Packs, create a business profile to receive and manage sales leads from Microsoft, and see if you qualify for co-selling opportunities.

Step 2: Register as a publisher

The first step to becoming a publisher is to register in Partner Center (PC). PC is where you submit your apps for publication, promote your apps, and manage your offers. To begin the registration process, you must complete

these steps. One of our team members will follow up to help you complete your registration. Once registered, you can access PC.

For more information, see Partner Center Account.

PartnerSource Business Center (PSBC) account

Developing apps requires you to be known as Business Central developer and requires you to have a unique development license file with a specific object range.

To obtain an object range for developing a Business Central, you must first have access to PartnerSource Business Center (PSBC). You have access to PSBC if you have one of the following agreements:

- An active Partner Registration Agreement (PRA)
- A Registered Solution Program Addendum (RSPA)

The relevant contract can be requested through your local Regional Operations Center (ROC) Contracts and Agreements Team below:

- mbscon@microsoft.com if you are based in Europe, the Middle East, or Africa
- mbsagree@microsoft.com if you are based in the Americas
- mbslques@microsoft.com if you are based in the Asia Pacific region.

Step 3: Your unique app specifications

Requesting an object range

When you develop an app for Business Central, you must request access to an object range that holds a certain number of objects for your solution. To avoid overlap between objects used in different solution, each partner is assigned a number of objects in a unique object range. For example, a partner is assigned the object range 70,001,000 – 70,001,999. The object range gives them 1000 numbered objects that they can use to develop Business Central solutions.

Depending on where you will deploy your Business Central solution, online or on-premises, you can use different licensing methods and object ranges.

There are currently two available ranges that you can request. Both have some characteristics to keep in mind:

- *RSP Object Range* (ID range 1,000,000-69,999,999)

  This object range is tied to the RSP Program details.

  > **IMPORTANT**
  > We currently advise new publishers to *not* request an RSP object range

- *App Object Range* (ID range 70,000,000-74,999,999)

  This object range was originally designed just for apps in the Microsoft commercial marketplace to be used in Business Central online.

  > **IMPORTANT**
  > We currently advise new publishers to request an app object range.

Currently, you can implement apps developed in both the *RSP range* and the *app object range* in Business

Central online and on-premises, as well as partner-hosted.

You can request an object range by downloading the object range request form here. After completion, send them to your Regional Operational Center (ROC) for processing:

- mbscon@microsoft.com if you are based in Europe, the Middle East, or Africa
- mbsagree@microsoft.com if you are based in the Americas
- mbslques@microsoft.com if you are based in the Asia Pacific region.

Downloading your development license file

After your Regional Operational Center has processed your Agreements and Object Range Request forms, download your company's unique developer license from PartnerSource Business Center. Find it in the license key configuration section under the developer tools section.

Register your unique prefix or suffix

In your extension, the name of each new application object must contain a prefix or suffix. This rule applies to all objects. Email d365val@microsoft.com with a prioritized list of three-letter affixes of your choice. You must also submit your MPN ID and the publisher name that you will use for the app.

Step 4: Getting access to preview bits

Get access to preview builds by joining Microsoft Collaborate.

In Microsoft Collaborate, you get access to a set of Business Central builds:

- The current major version
- An upcoming major version
- Daily builds

You must have the following prerequisites to register on Microsoft Collaborate:

- Azure Active Directory (Azure AD).

  > **NOTE**
  > If you have Microsoft 365, then your company most likely has Azure AD

- Azure AD Global Administrator permission

  > **NOTE**
  > To find out if your company has an Azure AD account, check with your system administrator.

Step 4 A: How your Global Administrator must register for Collaborate

Only your company's Global Administrator can start the onboarding to Collaborate. They must register at https://aka.ms/Collaborate, choose the **Get Started** action, and then complete the registration form.

The administrator can then add the relevant colleagues.

Optional Step: Add your coworkers to Microsoft Collaborate

To add coworkers:

1. Sign in to Microsoft Collaborate with your Global Administrator account at aka.ms/Collaborate.
2. Choose the  icon in the top-right corner of the page, click on account settings, and choose **user**

management.

3. Choose the grey **ADD USERS** button, and leave the default choice to **Add existing users** as-is. Now you can search for the user(s) that you want to add to Collaborate. To add them you need to select them from the menu, and then click the grey **ADD SELECTED** button.

4. You have successfully added your coworkers to Collaborate. Users can now sign in to Microsoft Collaborate using the following link: aka.ms/Collaborate

Step 4 B: Getting access to the available builds and engagements

Once you have successfully registered on Microsoft Collaborate, Microsoft must assign you to the right programs and engagements before you can see the preview bits. Contact Dyn365BEP@microsoft.com and provide them with information about the relevant users. the following table illustrates the type of information that you must submit:

PUBLISHER DISPLAY NAME	MPN ID	FIRST NAME	LAST NAME	WORK ACCOUNT EMAIL
Contoso	12345	Eugenia	Lopez	Eugenia.Lopez@Contoso.com
Contoso	12345	Quincy	Watson	Quincy.Watson@Contoso.com

After sending the email, expect a response from Microsoft within 1-2 business days.

Step 5: Resources while you develop your solution

Find below some guiding resources on how to develop your apps for Business Central.

- Microsoft Learn

  Learn new skills and discover the power of Microsoft products with step-by-step guidance. Start your journey today by exploring our learning paths and modules.

- Microsoft Docs

  Find The developer and administration content on Microsoft Docs

- Join the conversation

  In the dedicated Yammer network, join the conversation on developing apps

- Join the monthly Office hour calls

  Join the monthly Office hour calls to learn more about a hot topic

- Get coaching from experts

  Need help with developing your apps? There is a community of ISV Development Centers specialized in Business Central ready to engage with you.

Set up Azure DevOps for your development processes

Optionally, use the CI/CD workshop document at https://aka.ms/cicdhol if you want to do it yourself. Alternatively, choose between vendors offering tools or services around DevOps.

Talk to one of the ISV Development Centers for guidance.

Step 6: Publish your app in the Microsoft commercial marketplace

Once your app is ready for submission, you can list your app in the Microsoft commercial marketplace by submitting it in Partner Center. For more information, see Create a Dynamics 365 Business Central offer.

Before you submit, we recommend that you review the technical validation checklist and marketing validation checklist. The two articles list all requirements that you **must meet before you submit** an app for validation. If you do not meet these mandatory requirements, your extension will fail validation

See also

The SMB Opportunity for App Publishers
The Lifecycle of Apps and Extensions for Business Central
Update Lifecycle for AppSource Apps FAQ
Maintain AppSource Apps and Per-Tenant Extensions in Business Central Online
Microsoft Responsibilities for Apps on Business Central online

Marketing Validation Checklist

2/17/2021 • 3 minutes to read • Edit Online

The storefront details on AppSource is the first impression that prospects get regarding your offer. First impressions last, so make sure to invest some time in developing the content on the storefront, so it gives a good impression from the beginning. Failing to do so will jeopardize the hard work you put in, when developing your offer, likely leaving the prospect confused or looking elsewhere. Accordingly, we recommend you put in the time, effort and due diligence when developing this content.

You use Partner Center to submit your offer to AppSource. In the following, you can find information on all the marketing-related items that you need to fill out in Partner Center prior to submitting your app to AppSource. Follow this marketing validation checklist and get your app passed on the first submission.

What do I need to know before I begin?

You need to have the following three items in mind, when you are creating your storefront and marketing material.

ITEM	REQUIREMENT	DETAILS
Branding	Make sure to read the branding guidelines carefully before you start referencing the product name.	Read more
Language	Discover what to consider for each storefront detail when it comes to language.	Read more
Microsoft images	Make sure not to use any Microsoft images (e.g., Business Central icon or Dynamics 365 logo).	Read more

How do I fill out the marketing section in Partner Center?

The following section walks you through the marketing-related components, that is, *offer setup*, *properties*, *offer listing*, and *availability*, which you need to fill out in Partner Center prior to submitting your app to AppSource.

Offer Setup

ITEM	REQUIREMENT	DETAILS
App type	Read more about the types of apps you can submit to AppSource.	Read more
Contact type	How do you want potential customers to interact with your offer?	Read more
Test drive	If you choose “free trial” as the contact type of your offer, you need to enable a test drive.	Read more

ITEM	REQUIREMENT	DETAILS
Customer leads	Provide connection details to the CRM system where you would like to send customer leads.	Read more

Properties

ITEM	REQUIREMENT	DETAILS
Categories	Choose a primary, a secondary, and up to four subcategories.	Read more
Industries	Choose up to two industries for your offer.	Read more
App version	Specify the version number of your offer.	Read more
Terms and conditions	Outline the terms and conditions that the customer must accept before they can use your offer.	Read more

Offer listing

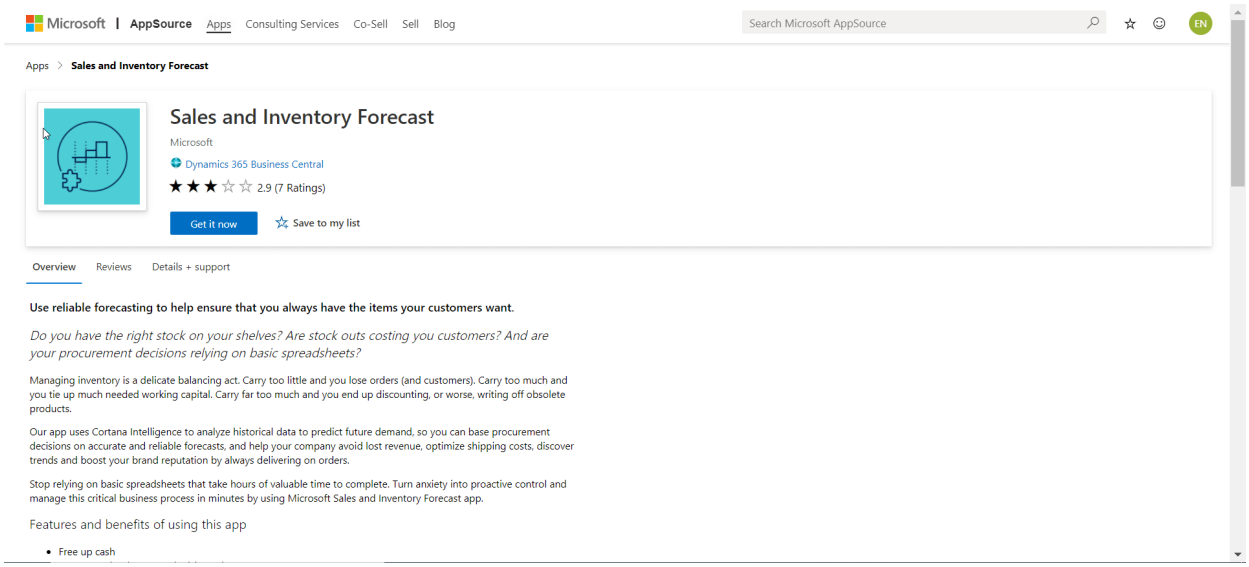
ITEM	REQUIREMENT	DETAILS
Offer name	Enter a descriptive name for the offer.	Read more
Offer summary	A single sentence summarizing the purpose or function of the offer.	Read more
Description text	Make an elaborate and compelling description that outlines the benefits and usage scenarios of your offer (include supported editions, countries, and languages).	Read more
Search keywords	Add search keywords to help users find your offer when they search in the marketplace.	Read more
Products your app work with	Add specific products that your app works with.	Read more
Help link	The link to your offer's learning resources.	Read more
Privacy policy link	The link to your offer's privacy policy.	Read more
Support link	The link to your offer's support page.	Read more
Supporting documents	Add supporting sales and marketing assets such as white papers, brochures, checklists, or PowerPoint presentations.	Read more

ITEM	REQUIREMENT	DETAILS
Logos	Upload a large size logo file with dimensions between 216 pixels x 216 pixels and 350 pixels x 350 pixels.	Read more
Screenshots	Provide a minimum of 3 screenshots that showcase your offer.	Read more
Videos (optional)	Add up to 4 videos that demonstrate your offer. These should be hosted on an external video service.	Read more

Availability

ITEM	REQUIREMENT	DETAILS
Markets (countries)	Choose the markets that your app is available in (make sure that it resembles the supported countries paragraph in the description text).	Read more
Hide key	The hide key is a token that is used to view the preview of your offer in AppSource before going live.	Read more

How does my offer look when it's live on AppSource?



Technical Validation

2/17/2021 • 7 minutes to read • Edit Online

Below you will find a checklist of all requirements that you **must meet before submitting** an extension for validation. You will also find a description of how the Business Central Validation team is performing technical and manual validation and how you can implement a validation pipeline to perform the same technical validation yourself.

Technical Validation Checklist

If you do not meet these mandatory requirements, your extension will fail validation. To get code validation helping you bring your extension package to AppSource, you can enable the **AppSourceCop** code analyzer. For more information, see Using the Code Analysis Tool.

REQUIREMENT	EXAMPLE/GUIDANCE
Develop your extension in Visual Studio Code.	Developing AL Language extensions
The app.json file has mandatory settings that you must include. Here you can also read more about dependency syntax and multiple countries per a single app syntax.	Mandatory app.json settings
Coding of `Date` must follow a specific format (**no longer region-specific**)	Use the format `yyyymmddD`. For example, `20170825D`.
Remote services (including all Web services calls) can use either HTTP or HTTPS. However, HTTP calls are only possible by using the HttpRequest AL type.	Guidance on HTTP use
Only JavaScript based Web client add-ins are supported. The zipping process is handled automatically by the compiler. Simply include the new AL `controladdin` type, JavaScript sources, and build the app.	Control Add-Ins
The .app file must be digitally signed.	Signing an APP Package File
The user scenario document must contain detailed steps for all setup and user validation testing.	User Scenario Documentation
Set the application areas that apply to your controls. Failure to do so will result in the control not appearing in Dynamics 365 Business Central.	Application Area guidance
Permission set(s) must be created by your extension and when marked, should give the user all setup and usage abilities. A user must not be required to have SUPER permissions for setup and usage of your extension.	Exporting Permission Sets Managing Users and Permissions
Before submitting for validation, ensure that you can publish/sync/install/uninstall/reinstall your extension. **This must be done in a Dynamics 365 Business Central environment**.	How to publish your app

REQUIREMENT	EXAMPLE/GUIDANCE
Thoroughly test your extension in a Dynamics 365 Business Central environment.	Testing Your Extension
Do not use `OnBeforeCompanyOpen` or `OnAfterCompanyOpen`	Replacement Options
Include the proper upgrade code allowing your app to successfully upgrade from version to version.	Upgrading Extensions
Pages and code units that are designed to be exposed as Web services must not generate any UI that would cause an exception in the calling code.	Web Services Usage
You are required to prefix or suffix the **Name** of your fields and objects. This eliminates collision between apps.	Prefix/Suffix Guidelines
We strongly recommend you are using automated testing, using the AL Test Toolkit. You are not required to include the test package with your extension.	Testing the Advanced Sample Extension
DataClassification is required for fields of all tables/table extensions. Property must be set to other than `ToBeClassified`.	Classifying Data
You must use the Profile object to add profiles instead of inserting them into the **Profiles** table.	Profile Object
Use `addfirst` and `addlast` for placing your actions on Business Central pages. This eliminates breaking your app due to Business Central core changes.	Placing Actions and Controls

Technical validation performed by the Business Central validation team

The primary responsibility of the technical validation is to ensure that the Business Central online service is stable and that the apps can be installed and run without destabilizing the service.

The technical validation is for a large part automated and will validate the steps described in the technical checklist above through some pipelines.

The submitted apps will be extracted and investigated following this list:

1. The apps are investigated. All dependencies must be included in the submission. We will lookup prior versions of the apps in the depot. If your app has a dependency on a third party app in AppSource, you should not include this, we will locate it in the depot. **Any unresolved dependencies will cause the submission to be rejected.**
2. If the version numbers haven't changed and the countries haven't changed, the validation is skipped and **the apps will not be updated.**
3. App.json is investigated for mandatory fields. **If mandatory fields are missing, the submission is rejected.**
4. Affixes for the submission are located. **If affixes haven't been registered or cannot be located, the submission is rejected.**
5. Business Central Artifacts are located for the version the apps is submitted for (*Current*, *NextMinor*, or

NextMajor).

6. For every country in the submission list, we perform the same validation:
   - A sandbox container based on the Business Central Artifacts with the right country version is created.
   - Any dependency apps not included in the submission are installed. **If any installation fails, the submission is rejected.**
   - In order of dependencies, all apps in the submission are tested using AppSourceCop analyzer. For more information, see AppSourceCop Analyzer
     - If any **breaking changes are identified, the submission is rejected.**
     - If mandatory affixes **are not included on all object names, the submission is rejected.**
   - In order of dependencies, all prior versions of the apps are published and installed. **If any installation fails, the submission is rejected.**
   - In order of dependencies, all new versions of the apps are published and upgrade is run (apps must be digitally signed, else they won't install). **If any installation/upgrade fails, the submission is rejected.**
   - A simple connection test is run; opening a role center and check simple actions and pages. If the connection test fails, **the submission investigated and potentially rejected.**
7. If all country validations succeed and no errors are found then **the submission is accepted.**

> **IMPORTANT**
> Microsoft recommends that all partners are performing the same checks as described above before submitting apps for validation to maximize chances of validation success.

Running technical validation yourself

With the latest version of BcContainerHelper, you can run a single command, which should perform the same validation steps and give you a good indication of whether your apps will pass validation or not:

```
$validationResults = Run-AlValidation `
    -licenseFile "path/url to license file" `
    -validateCurrent `
    -installApps @( "path/url to your foreign dependencies, apps which will not be part of the validation
(or blank if this is the first)" ) `
    -previousApps @( "path/url to your previous version of the .app files (or blank if this is the first)" )
`
    -apps @( "path/url to the new version of the .app files" ) `
    -countries @( "countries you want to validate against (f.ex. us,ca)" ) `
    -affixes @( "affixes you own (f.ex. fab,con)" ) `
    -supportedCountries @( "supported countries (f.eks. us,ca)" )
$validationResults | Write-Host -ForegroundColor Red
```

All array parameters can also be specified as a comma-separated string. For more information, you can also check this blog post Run-AlValidation and Run-AlCops.

Please include app and all library apps in both previousApps and apps and please include all countries on which you want to validate.

> **NOTE**
> The Run-AlValidation cannot see whether the affixes to specify have been correctly registered with Microsoft using your MPN ID and app publisher name, please make sure registration is in place.

IMPORTANT

The Computer on which you run this command must have Docker and the latest BcContainerHelper PowerShell module installed and be able to run Business Central on Docker.

If you are having issues with Business Central on Docker, you might be able to find help here: https://freddysblog.com/2020/10/12/troubleshooting-business-central-on-docker.

You can use https://aka.ms/getbc?artifacturl=bcartifacts%2fsandbox%2f%2fus%2flatest to create an Azure VM, which has all prerequisites installed to run Business Central on Docker.

NOTE

Microsoft recommends that all partners set up DevOps processes to ensure that this validation process happend automatically and regularly.

You can find resources for how to set up a build pipeline, which performs all these steps here: https://aka.ms/cicdhol and you can find sample repositories, performing these steps here:

- https://dev.azure.com/businesscentralapps/HelloWorld.AppSource (for Azure DevOps)
- https://github.com/BusinessCentralApps/HelloWorld.AppSource (for GitHub Actions)

Manual validation performed by the Business Central validation team

The primary responsibility of the manual validation is to ensure that the apps are working as described.

Manual validation is not done on all submissions. They will be done as sample tests.

For manual validation, we spin up a container with the right artifacts (same as used during technical validation) and the necessary apps are installed. Rapidstart packages needed for the manual test are installed.

The manual test validation document is run manually and if the document doesn't match the app functionality the submission is rejected.

IMPORTANT

Microsoft recommends that all partners are performing the manual validation as the last check before submitting for validation.

This can be done either in online sandbox environments or in sandbox docker containers.

See Also

Developing AL Language extensions

How to Make Compelling Videos

2/17/2021 • 9 minutes to read • Edit Online

Why use video? It is well worth investing time and resources to create marketing videos for your app, it is taken seriously in a business environment.

Reasons why video is a superior medium

- Videos offers a very rich, stimulating communication medium that engages multiple senses.
- Video engages the mind and triggers emotions, which makes it more compelling than text-based content.
- Our brains have an easier time processing visual stories than bullet points or straight facts.

A recent Demand Gen survey indicated that 58% of B2B buyers consume video content, while Hyperfine media states that 59% of executives would rather watch video than read text. Also, 50% of executives look for more information after seeing a product/service in a video.

Speak to Specific Personas in your videos

You should create a video for each of the three core personas in the company:

- WHY persona: Owner/executive/leadership
- HOW: Business line manager
- WHAT: IT buyer, User

A horizontal generic message that attempts to speak to everyone will likely not reach anyone in an emotionally engaging way. Wasting a prospect's time by requiring him/her to listen to irrelevant data or information will only create frustration and lead him/her to form a negative bias towards your company.

Choose the video format that is relevant for the audience that you want to target

Video type 1: "Why" video

How to set up "Why" videos

- Recommended length: 60-90 seconds
- Purpose:
  - Your video should clearly communicate WHY prospects need to buy your solution now.
- Focus:
  - Make sure the prospect is the hero of the story, not you or your company. Prospects are not interested in hearing about your company at this stage. They are simply trying to determine if what you offer is of value to THEM.
  - Your video should speak to the principal challenges and goals of your core decision-maker persona.
  - Describe the desired end state they will achieve by using your app.
  - A client/customer speaking about the benefits they received from your app is far more credible and compelling than anyone from your organization.
    - Don't only rely on "features" to acquire new customers.

How to speak to a WHY persona in a video

- Target audience:
  - Owner/executive/leadership
  - They have limited time and financial resources as well as many competing priorities and resource requirements
  - You need to elevate the discussion to a strategic level, where you highlight market share, competitiveness, profitability, differentiation, revenue loss, and more.
- Message:
  - The question you must answer beyond a doubt is WHY should they invest the time and money to buy your app? What will they get out of it?
  - Why should they spend money on a new system now? Can't they put it off?
  - The WHY messaging teaches people something and it is industry specific and results oriented, as well as being memorable. It engages the emotional/limbic brain and leads to meaningful action.

Video type 2: "How & What" video

How to set up "How and What product videos"

- Recommended length: Up to 3 minutes.
- Purpose:
  - This video goes into greater depth communicating the main benefits of your app as well as HOW you solve your prospects' problems. You can include some WHAT content. • Focus:
  - Demonstrating the proof of your claims is critical during this video.
  - Show very specific dashboards or visually show how you address prospect challenges.
  - If possible, use contrast to create desire and a sense of urgency. For example, you could show a complex, ugly data-filled forecast spreadsheet next to a beautiful visual dashboard stating "your sales forecast before and after."

How to speak to a HOW persona: (Business line manager)

- Target audience: Business Line Manager
  - HOW focuses on the operational benefits your solution will provide and HOW your organization will support the implementation.
  - Speaking to the HOW persona starts to separate you from the pack. • Message: o HOW content is VISUAL in nature and ACTION oriented. It allows your prospects to identify with you at a FUNCTIONAL business level and to Add-on with you. It provides evidence that your organization has relevant industry experience. Tribal acceptance increases, while risk decreases.
  - HOW messaging begins to appeal to the limbic brain because it is focused primarily on emotional business pains and problems.

How to speak to a WHAT persona: (IT buyer, User)

- Target audience: IT-buyer, User
  - WHAT people are often tasked with finding a solution and are important influencers in the decision, but they are not the financial decision makers, and their opinions are easily overturned by HOW and WHY people in the organization.
  - Therefore, don't invest all of your marketing time, money, and effort into providing content just for them.
- Message:
  - You need to survive the WHAT inquisition and provide information about product-related features, functionality, and data so that prospects clearly understand your solution offering.
  - However, this will seldom trigger an emotional response and, therefore, it is likely there will be little or

no emotional engagement with your content.

- WHAT content is binary. WHAT content is a commodity. WHAT content is boring. Logical WHAT content is a necessary evil because many prospects initially go looking for it, but stopping here means remaining relevant only to WHAT personas.

Video type 3: "Getting started" video

How to set up "Getting started videos"

- Recommended length: 2–3 minutes maximum.
- Purpose: This video should prove it is quick and easy to get up and running with your app.
- Target: What personas (Users, It buyers)

Video type 4: "Customer testimony" video

How to set up "Customer testimonial videos" - Recommended length: Up to 2 minutes

- Purpose:
  - Social reinforcement: Customer stories are the best proof of gain.
- Focus:
  - A story coming directly from your client in the form of a testimonial is stronger than having your prospects take your word for it. If prospects see that other similar people orcompanies have already purchased your solution, then their natural response will be to more readily accept it as a solution for themselves.

Video tips

How to structure your video and practical things to keep in mind when producing videos

How to structure the flow in your video?

- Gain immediate attention in the first 10 seconds of the video Stimulate curiosity by include a hook phrase/comment that will elude to solving a pain point. Ask questions about the prospects' core business challenges or ask about something they would like to do but can't accomplish today.
- Highlight the prospects' problems: Use an empathetic approach when describing their current situation and demonstrate that you understand their current business challenges. They must relate to this if they are to continue watching.
- Give them new learning Teach them something they don't know. Demonstrate you have expertise and knowledge about their business or industry that they might not. Show you can offer strategic value to them.
- Paint a picture of a desired outcome they would love to have or state they crave to experience Highlight the benefits, rewards, and value they will enjoy after they purchase from you. Include both what it looks like and how it will feel.
- Prove what you're saying is true Prospects don't trust us when we say our products are great. Include objective and credible proof in the form of data, charts, graphs, quotes, statistics, or testimonials as evidence of your claims.
- Ask them to take action Include a call to action at the end of all videos. When viewers watch your videos, they should feel inspired to take the next step towards purchasing. Tell them what to do next and include an interactive link to the next step in the buying cycle. Use scarcity to compel them to action. Provide a timelimited offer or, for example, say it is "only for the first 20 customers".

Practical things to keep in mind when producing and distributing your video

Does and don't when producing your video

- Don’t make the video too long As our attention span is 8 seconds the ideal length of video is 90 seconds (minimum 30 seconds/maximum 2 minutes).
- Add interactivity where possible Overlay text, charts, animation, questions etc. Visually call out key messages.
- Make sure your audio is high quality.
- Make your video easily shareable
- Enable your video to be shared on multiple media. Track views and attention span. Observe and measure viewer patterns so that you can learn from prospects’ actual behaviors and then improve future content.

How to make a good narrative that speak to the right persona in the right way?

- Your narrative should have a beginning, middle, and end.
  - Lead with a story, not with your app or the technology.
  - Don’t turn your videos into a product pitch.
  - You’ll build more brand affinity and trust by shedding light on a problem your prospects care about rather than by pitching your solutions to them directly.
- The brain is on alert at the beginning of the video and at the end.
  - Make sure the first and last ten seconds are compelling, memorable, and interesting.
- Speak directly to a particular persona in the second person.
  - Do not talk about them in the third person, and avoid using terms like “our clients” and “companies”; instead, use “you” language as often as possible.
  - Use a lot of industry specific vocabulary, terminology, and visuals. If possible, film onsite at a customer’s location rather than in your office or in a studio.
- Speak to a particular persona:
  - Do not try to appeal to everyone at once, as you may not fully engage anyone with this approach.
  - Keep your delivery casual and authentic to instill trust. Speak directly to the prospect as if you were having a fireside chat
  - The prospect should be the hero of the story, i.e. do not speak about you and your company.
- Ask rhetorical questions that stimulate pain and anxiety in your prospects in order to demonstrate that you understand their business problems.
  - For example: Are your margins decreasing? Having cash flow problems because you can’t collect payments sooner than 90 days? Had another large write off? Lost an important customer recently due to a late delivery?
- Use visual and auditory language to help the prospect imagine a new possible future.
  - For example: “imagine seeing” , “picture yourself”, or “ how would you like to hear your clients say…” and so on.
- Use contrast whenever possible.
  - Compare prospects’ experience now versus what it could be after the implementation of your solution.
  - Call out your competitive differentiators while anchoring your solution in prospects’ minds so that they can compare all others against the bar you set.

How to make a good narrative that speak to the right persona in the right way?

- Where possible, use tangible, concrete language.
  - Include quantifiable proof in the form of data or visual pictures.
  - No vague claims like “transform your business with the cloud”. This is an emotionless statement.
- Providing customer references and testimonials is much more compelling and effective than selling your company or product yourself.
  - Let others speak for you. A customer testimonial video will always be more believable and compelling than a video of you saying the same thing.

- Surprise and delight them.
    - Use humor to make them smile. We take ourselves and our problems too seriously. Be warm, memorable, and unique.

Guideline on Creating an Effective Sales Landing Page for Your App

2/17/2021 • 10 minutes to read • Edit Online

Building a landing page that drives a successful buying transaction

Microsoft will drive qualified traffic to AppSource. Though, once a prospect becomes aware of your app, it will be your job to guide them through to a successful buying transaction. Deliberately mapping and architecting the buying journey is critical to ensure a high level of engagement and conversion. Only presenting your app's features and functionality, or just providing a free trial, will not ensure prospects will become buyers. For this you need to have a good landing page that is built to help you capture attention, accelerate your customer acquisition process, and drive buying behavior. The recommendations on this page will help you do so.

Examples of how other partners have implemented our best practices

To inspire you in creating a good landing page for your app, two of our valued partners, LS Retail and Industry Built, have offered to provide a sample of what a best practice landing page for a Microsoft Dynamics 365 Business Central partner could look like.

Have a look at their app landing pages and use them as inspiration to build your own landing page:

- Industry Built's Build Food app
- LS Retail's LS Express Start app

In the following checklist, we have "broken down" the elements, on their landing pages in order to showcase best practices on design and messaging. More specifically, we are looking into layout and structure elements, content elements, visual elements, anxiety reducing elements and support elements.

Additionally, we have provided specific recommendations on how to apply these elements to help you increase conversion and maximize the effectiveness of your product's sales landing page.

We urge you to review and implement these best practices on your landing page – in so doing you will contribute in providing the Microsoft community of customers with a consistent buying experience across publishers.

Layout and structure elements

ELEMENT	DESCRIPTION	EXAMPLE
Company	Include the company logo on the page	
App name & app logo	Include a visual logo of your product name and a one sentence positioning statement.	
Top menu choices	Use clean, straightforward and descriptive menu options.	

ELEMENT	DESCRIPTION	EXAMPLE
Search box	Include a search box so visitors can quickly find what they are looking for.	
Emotional tribal anchor photos	Visuals create an emotional Add-onion. The brain skims over non-emotional photos.	
Visual	Make your page easy to scan, with lots of strong visual imagery.	

Logo

- The upper-left corner of the landing page is the most valuable section of the entire landing page.
- Place your company logo in this location.

If you need help formulating a positioning statement, try the value proposition generator located at here.

- There should ideally be 5 or fewer choices; do not include more than 7 options.
- The menu text should state what the prospect gains if they click on the menu item
- The text should be written from their perspective, not yours.

Recommended menu items:

- How to Buy, Benefits Gained, Why Us, and Contact.

The upper-right corner of the page is usually an ideal spot

- Faces evoke more emotion than landscapes or machines, and so on.
- Include a happy customer that looks similar to your prospect in terms of age, demographic, and industry, and which shows them dealing with the issues that your prospect can relate to.
- Try not to use stock photos of people or objects.

Engagement

- Too much text forces the brain to skim, skip, and exit. Text engages the logical, analytical brain, but not the emotional brain.
- Keep it clean and straightforward in terms of design and layout. Use lots of pictures, graphs, and screen shots to enhance engagement.

Content elements: Text and messaging

ELEMENT	DESCRIPTION	EXAMPLE	
Include a headline question	Get your prospects' attention by asking them a compelling pain-based question that they can relate to.	"Struggling to manage your ingredient inventory and fretting over allergens?"	

ELEMENT	DESCRIPTION	EXAMPLE	
	You want the prospect to mentally say "YES" as often as possible and to peak their curiosity enough to read more.		
	Your questions should be intriguing and customer-centric.		
	In general, 8 out of 10 people will read headline copy, but only 2 out of 10 will read the rest.		
Microsoft Dynamics 365 product description	Somewhere on the landing page, make sure you include the standard Microsoft Dynamics 365 Business Central product description provided by Microsoft *This is a requirement because your product is adding value to and building on this foundational solution.*	*Insert this paragraph:* Microsoft Dynamics 365 Business Central is a comprehensive business management solution for small and medium-size businesses (SMBs) that have outgrown their basic accounting software. From day one, this new application makes ordering, selling, invoicing, and reporting easier and faster. Dynamics 365 Business Central is deeply integrated with Microsoft 365 and includes built-in intelligence, so it is easy to use and helps users make better business decisions.	
Messaging (Address their pains)	Pain is a strong motivator of action.		
	- Identify 1-3 key sources of the client's most prominent pain early on the page.		
	- Call out the fears that are likely to be holding them back.		
	- Your landing page text and messaging should predominantly focus on the pain the prospect is experiencing, and NOT the features of your product or service.		

ELEMENT	DESCRIPTION	EXAMPLE	
Clearly demonstrate to your prospects that you genuinely understand their industry and unique business problems.	- Describe the business challenges they are facing now and the ways their revenue growth, margins, productivity and so on, are being negatively impacted by not taking action now.		

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Product benefits)	Paint a clear, visual and desirable picture of what is possible.	
	Describe the most significant benefits and rewards that your prospect will realize after purchase.	- For example, "Save time and money (benefits) by having a system that does all the tracking and calculations for you (features)."
	- Don't only list features and app functionality, start with the benefit first, then you can follow with the features.	
	- Paint a picture of a possible experience the prospect will immediately desire.	
Clearly articulate a compelling desired outcome	- If possible, use industry-specific language and vocabulary to resonate with your prospect deeply.	
	- Choose a particular persona to speak to directly.	
	- Engage prospects by speaking directly to them using first person "you" language.	

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Prove your claims)	Include specific calls-to-action on your app page.	"Reduce how long it takes to set up your recipes in the morning from 1 hour to 10 minutes."
	Don't make general and abstract claims.	

- Use data as often as possible to support your statements.

If you make specific claims, support your claims with proof, while Quantifying impacts and gains.

- The more specific and concrete your promise of value is, the better.
- Abstract concepts such as "more efficiency, more productivity, transform your business" are not emotionally

impactful or convincing and do not compel a prospect to act.

Target market - If you support multiple countries or languages, this is a key selling feature. • Find a way to show this visually.

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Compelling call-to-action)	Include specific calls-to-action on your app page	

- This can be your free trial; a time-limited special price; a scheduled walk-through demonstration; and so on.
- The words "free" and "save" are highly emotional words in the English language, so they should be used.
- Use bright colors, such as orange, yellow, or red, to call attention to your buttons.

Button text should use benefit language rather than descriptive language.

- For example, instead of "Download" write "Click here to start saving money now."
- Try not to send prospects away from your page – always have an embedded next step in your call to action that brings them back to your landing page.

ELEMENT	DESCRIPTION	EXAMPLE
Messaging (Create a sense of urgency by teaching the prospects)	Help your prospect gain a sense of urgency to buy by teaching them one thing about how they can be more efficient or profitable now.	Your bakery profitability will decrease over the next five years due to an increase of 3% in the cost of key inputs, such as wheat and sugar. Want to know five key strategies that can help you mitigate this challenge? Click here to find out how to preserve your profit margin

- Show them how their performance in one key business area is below that of their competitors.
- For an example you can provide a quick online self-assessment, a top-10 tips blog post, and much more. Visual elements

ELEMENT	DESCRIPTION	EXAMPLE	
Pictures (Differentiation comparison images)	Show them, don't tell them	Show the before and after state.	

- This is a visual image of how your prospects do things now versus how they will be able to do it in the future.
- You are not telling them but showing them using a visual.

ELEMENT	DESCRIPTION	EXAMPLE
Compelling proof screen shots	Visually demonstrate all the claims that you are making.	Quickly and easily view inventory items

- Graphic dashboards are the most effective method.
- Zoom in on the main benefit-related features.
- Make sure it is readable, and the benefit is obvious.
- Include a caption.
- Data should be industry specific so that it resonates with the viewer.

You want prospects to see how their data/process would look in your system.

ELEMENT	DESCRIPTION	EXAMPLE
Videos (Tell your story using videos not text)	Include as many videos as possible.	

- Videos have a much higher level of engagement and viewing time and convey much more than you can ever say with words.

Include at least one customer testimonial video on your app landing page.

- Your client should speak specifically about the pains they had before and the benefits they gained after, not product features. It should be all about your customers, not you.

Include one product demonstration video.

- See the video best practices https://aka.ms/ReadyToGo.

Elements that reduce anxiety and risk, while increasing trust

ELEMENT	DESCRIPTION	EXAMPLE
Customer testimonials	Don't sell your product; let your customers do that for you.	

- Social proof is more credible and trustworthy to prospects. The purpose of testimonials is to reduce the buyer's anxiety and fear.

Your testimonials should answer the following questions:

- "Will this work for my situation?"
- "What benefit will I really get if I buy this?"
- "Is this going to be too hard?"
- "How long is this going to take?"
- "Can I trust this company?"

ELEMENT	DESCRIPTION	EXAMPLE
Reduce risk	Prospects are afraid of being scammed and taken advantage of on the internet. They are naturally cautious and highly suspect.	Source: Microsoft.com

- You want to convert prospects to buyers.
- Make it easy for them to buy, while reducing their anxiety. Transparency is the key to building trust.
- Make sure that you include a link to a BUY NOW page, which includes full pricing details.
- Give them a compelling offer they cannot refuse. Offer a time-limited trial or special pricing discount if they buy in 30 days.
- Use scarcity to compel action. Offer a 100% money-back guarantee.

We recommend providing three offerings, optimized for three different customer segments. For more recommendations on pricing, see the pricing guide located at https://mbspartner.microsoft.com/BFI/Topic/64

ELEMENT	DESCRIPTION	EXAMPLE
Live chat	Include live chat, with a photo of one of your team members smiling at an appropriate time to increase conversion, such as when a prospect clicks the back button on your pricing page.	

- Include their name if possible to build trust.

Support elements: Interactivity and contact options

ELEMENT	DESCRIPTION	EXAMPLE
SHORT lead capture form	Include a lead capture form on your page.	

- Only ask for their name and email address, you can get the rest later.
- Your forms should not have more than 4 or 5 fields to fill out. You have not yet earned the right or enough trust to ask for too much information at this point.

Most lead capture forms are way too long, demanding, and intimidating, and have low completion rates.

> **NOTE**
> Nobody has the time or is willing to fill out an annoying form, which is of no value to them, especially if it is purely self-serving from your standpoint.

ELEMENT	DESCRIPTION	EXAMPLE
Contact	Provide prospects with different contact options based on their readiness to interact with you.	

- Ideally, include a phone number and an email address with an employee photo.
- This alone could double your conversion rate.

ELEMENT	DESCRIPTION	EXAMPLE
AppSource app page link & social share	Include a link back to your listing on AppSource, so the prospect can return when ready.	Return to AppSource

- Also, enable visitors to share and forward your app with others!

ELEMENT	DESCRIPTION	EXAMPLE
Close them! Add a get started button	Include a very specific call-to-action button with the option to buy or try.	

Getting Started with C/SIDE and AL for On-Premises

2/17/2021 • 2 minutes to read • Edit Online

To get started with a mixed development environment of C/SIDE and AL, you must follow the steps below.

Steps to install Business Central on-premises with C/SIDE and AL development environment

1. Install Business Central on-premises and make sure to include the **AL Development Environment**.
2. Download Visual Studio Code.
3. From Visual Studio Code, locate **Extensions** in the left navigation bar, and then choose **Install from vsix**.
4. Browse to the equivalent folder of `C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment` and then choose **Install**.
5. Now, press **Alt+A, Alt+L** to trigger the **AL Go!** command, choose a project, the target platform, and then choose **Your own server**.
6. Authenticate with the credentials you use for signing into Business Central on-premises.
7. In the launch.json file, update the `"server": "https://localhost"` setting with the URL for server running Business Central on-premises and save the file.
8. In the app.json file, add the `"target": "OnPrem"` setting.
9. Now, use the Business Central Administration Console to ensure that the settings on the **Development** tab are set as follows:
   - **Allowed Extension Target Level** is set to **OnPrem**.
   - **Enable Developer Service Endpoint** checkbox is selected.
   - **Enable Loading Application Symbol References at Server Startup** checkbox is selected.
10. Make sure to read and ensure any additional settings here Running C/SIDE and AL Side-by-Side.

> **TIP**
> For information about which sandboxes you can choose, see Sandbox Environments for Dynamics 365 Business Central Development.

> **NOTE**
> Build and get inspired by our sample library on GitHub.

See Also

AL Development Environment
FAQ for Developing in AL

Running C/SIDE and AL Side-by-Side

2/17/2021 • 3 minutes to read • Edit Online

Business Central on-premises supports development using both C/SIDE and AL, as well as Designer side-by-side. When new objects are added or changed in C/SIDE these changes must be reflected in the symbol download in Visual Studio Code using the AL Language extension. To enable this reflection, a command and argument called `generatesymbolreference` has been added to finsql.exe and you can run it as illustrated below.

Get started generating symbols and compiling all objects

Open a command prompt **Run as administrator** and change to the directory where the finsql.exe file has been installed as part of Dynamics NAV Development Environment.

Use the `generatesymbolreference` command specified with the database and server name to add symbol references to the **Object Metadata** table for the specified database.

Given the `generatesymbolreference` command, C/SIDE will traverse all the objects in the database and generate symbols for them. This command should be run at least once to generate the initial set of symbols to which incremental updates can be applied.

Syntax example

```
finsql.exe Command=generatesymbolreference, Database=<DatebaseName>, ServerName=<ServerName>\<Instance>
```

For example:

```
finsql.exe Command=generatesymbolreference, Database="Demo Database NAV (11-0)", ServerName=.\NAVDEMO
```

> **TIP**
> The finsql.exe includes several parameters that you can set to suit you environment. For more information, see Using the Development Environment from the Command Prompt.

This is a lengthy operation. When you run the command, the console returns to an empty command prompt, and does not display or provide any indication about the status of the run. However, the finsql.exe may still be running in the background. It can take several minutes for the run to complete, and the symbols will not be generated until such time. You can see whether the finsql.exe is still running by using Task Manager and looking on the **Details** tab for **finsql.exe**.

When the process ends, a file named **navcommandresult.txt** is saved to the Dynamics NAV Client connected to Business Central installation folder. If the command succeeded, the file will contain text like `[0] [06/12/17 14:36:17] The command completed successfully in '177' seconds.` If the command failed, another file named **naverrorlog.txt** will be generated. This file contains details about the error(s) that occurred.

NOTE

The symbol references are stored in the **Symbol Reference** column of the **Object Metadata** table of the database. For on-premises installations, if you experience problems with generating symbols, check the information in the naverrorlog.txt file.

Continuously generate symbols each time you compile objects in C/SIDE

The `generatesymbolreference` flag enables incremental symbol generation through the UI or through the compile command passed on the command line. To update the symbols for a set of objects from the UI, start C/SIDE with the `generatesymbolreference` flag, make any desired modifications to your application objects, and compile them.

NOTE

Use `generatesymbolreference` set to yes as a command line argument each time you start finsql.exe to have all compilations add a symbol reference to the **Object Metadata** table. The default setting of the argument is no.

NOTE

If you make changes in C/SIDE and start the C/SIDE development environment without the `generatesymbolreference` flag set to `yes`, the symbols downloaded from Visual Studio Code will not reflect your changes.

Syntax example

```
finsql.exe generatesymbolreference=yes
```

This flag is also a part of the `Compile-NavApplicationObject` PowerShell command and you can use it to compile and generate symbols on a filtered set of application objects through PowerShell. This alternative should be considered if you do not work with the UI in C/SIDE. For more information about it, see Compile-NavApplicationObject.

Business Central on-premises server setting

In addition to the symbol generation setting you have chosen above, you must enable the Business Central on-premises server setting.

1. Go to **Business Central Administration**.
2. Scroll to the **Development** tab and expand the tab.
3. Choose the **Edit** button, and then select the **Enable loading application symbols at server startup** checkbox.

IMPORTANT

This setting must be enabled to allow any symbol generation. If the setting is not enabled, the `generatesymbolreference` setting does not have any effect.

See Also

Creating Runtime Packages for Business Central On-Premises

2/17/2021 • 2 minutes to read • Edit Online

If you want to distribute extensions, you can generate runtime packages that do not contain AL code, but only the final artifacts used by the server at runtime. Runtime packages thereby allow you to protect the intellectual property represented by your AL source code.

When the runtime package is generated on the server, the developer license is checked for permissions to the used extension IDs. The extension in a runtime package can then be installed on servers that do not have a developer license; the server only needs permissions to run the objects, but not to modify or insert them.

Start using runtime packages

The first step in using runtime packages is to have an extension developed and published to an on-premise instance. Next, use the following PowerShell command to connect to the server, find the extension, and download the runtime package.

```
Get-NavAppRuntimePackage
```

For more information about this cmdlet, see Get-NAVAppRuntimePackage cmdlet.

The following example gets the NAV App runtime package with the provided name and version.

```
Get-NAVAppRuntimePackage -ServerInstance DynamicsNAV -AppName 'Proseware SmartApp' -Version 2.3.4.500 -Path
'Prosware SmartApp_2.3.4.500_runtime.app'
```

For publishing and installing the package, use the Publish-NavApp and the Install-NAVApp PowerShell cmdlets.

Limitations

The limitation of runtime packages is that they only work for on-premise installations and therefore cannot be submitted to AppSource. Moreover, debugging into an extension to view the source code is not allowed by default; the `ShowMyCode` flag is by default set to `false`.

> **NOTE**
> Runtime packages are guaranteed to work only if published to a platform with the same version as the one where they were produced.

> **NOTE**
> If you set the `ShowMyCode` flag to `true` when running the `Get-NavAppRuntimePackage` cmdlet, you can enable debugging and you thereby also allow viewing the source code.

See Also

Publish-NAVApp cmdlet
Install-NAVApp cmdlet

Sandbox Environments for Dynamics 365 Business Central Development

2/17/2021 • 2 minutes to read • Edit Online

To get started developing for Dynamics 365 Business Central it is important to understand the different options you have at hand. You can either choose to run a sandbox environment deployed as a Dynamics 365 Business Central service, or you can run a container-based image either hosted as an Azure VM or locally. Both options provide the AL development tools; the container-based sandbox additionally provides access to the C/SIDE development tools. You can also choose to run a sandbox environment with production data using the **Business Central Admin Center**. For more information, see Business Central Admin Center.

> **NOTE**
> Extensions that have been published to a sandbox environment from Visual Studio Code or created using Designer are removed when the environment is updated or relocated within our service. For more information, see Production and Sandbox Environments.

> **IMPORTANT**
> It is not supported to publish, from Visual Studio Code, an extension with the same identifiers as an extension published to AppSource. Identifiers include the combination of appID and version or name, publisher, and version. If you do publish such an extension, it can be removed at any time.

Development sandbox overview

The following table outlines the most important capabilities on the offered development sandbox environments for Dynamics 365 Business Central.

CAPABILITY	ONLINE SANDBOX	CONTAINER SANDBOX
Deployment	Dynamics 365 Cloud Service managed by Microsoft	Azure VM or on-premises managed by ISV/VAR
Production data	Manually uploaded using Rapid Start packages. Or, available through the Business Central Admin Center.	Manually uploaded using Rapid Start packages
Production services	Manually configured	Not available
Cost	Part of the Business Central subscription	Locally hosted - free, Azure-hosted - cost incurred
Development	Full capabilities of the development environment. Designer functionality, such as: Add/Remove components, Move components, Set/clear Freeze pane, Edit captions	Full capabilities of the development environment. Designer functionality, such as: Add/Remove components, Move components, Set/clear Freeze pane, Edit captions

CAPABILITY	ONLINE SANDBOX	CONTAINER SANDBOX
Tools	Visual Studio Code, Designer	Visual Studio Code, Designer, on-premise tools such as SQL Server Management Studio, and C/SIDE.
Debugging	Enabled	Enabled
Database access	No	Yes
Extensions	Must be manually installed.	Must be manually installed.
From AppSource	Available.	Not available.
From File	Available.	Available.
From Visual Studio Code	Available.	Available.

Getting started

Based on the overview above and the requirements for your development environment, you can get started with a sandbox by following the links below:

- Online Sandbox with Demo Data
- Online Sandbox with Production Data
- Container Sandbox

See Also

Getting Started with AL
Keyboard Shortcuts
AL Development Environment
Production and Sandbox Environments

Get started with the Container Sandbox Development Environment

2/17/2021 • 3 minutes to read • Edit Online

Dynamics 365 Business Central offers a container-based image environment that enables access to the AL development environment.

You set up a container sandbox by running the **Container Sandbox Environment** page from Dynamics 365 Business Central. You will have to decide whether you want an Azure-hosted or locally hosted container sandbox. See the next section for details.

> **TIP**
> Dynamics 365 Business Central also offers an online sandbox. For more information, see Sandbox Environments for Dynamics 365 Business Central Development.

Choosing an Azure-hosted or local-hosted container sandbox

When you set up the container sandbox, you can choose to host the sandbox on Microsoft Azure or on a local computer. Both environments offer the same capabilities and use Docker to provide the infrastructure for the container-based application. The difference is:

- With Azure hosting, Docker is installed and configured for you. However, Azure hosting requires that you sign up for an Azure subscription, and additional costs may be incurred for each container sandbox.
- Local hosting requires that your computer is running Windows 10, Windows Server 2016, or Windows Server 2019, and you install and configure Docker before setting up the container sandbox.

Set up an Azure-hosted container sandbox

1. If you you do not already have one, sign up for an Azure subscription.

   To get a free subscription and read more information, see https://azure.microsoft.com.

2. In Dynamics 365 Business Central, choose the icon, enter **Container Sandbox Environment**, and then select the relevant link.

3. Choose **Host in Azure**. If prompted, enter the user name and password of your Azure subscription.

   The Azure portal opens to display a custom deployment form.

4. Fill out the required fields on the form, and then select **Purchase**.

> **NOTE**
> You must set the **Accept Eula** setting to **Yes** in order to continue.

Set up a local-hosted container sandbox

1. If not already done, install Docker.

   To install and configure Docker, choose the version of Docker that is appropriate for the host operating

system:

- For Windows 10, use Docker Community Edition. For more information, see Install instructions.
- For Windows Server, use Docker Enterprise Edition. For more information, see Install instructions.

2. In Dynamics 365 Business Central, choose the  icon, enter **SANDBOX ENVIRONMENT (CONTAINER)**, and then select the relevant link.
3. Choose **Host Locally**.
4. Save the **CreateBCSandbox.ps1** file to your computer.
5. Run Windows PowerShell ISE as an administrator.
6. Open the **CreateBCSandbox.ps1** file.
7. Set the `$containername = ''` variable to the name that you want to give the container, for example `$containername = 'mybc'`.

   This name is only used internally in your envirorment for identification purposes.
8. Set the `$accept_eula = ''` variable to `'$true'`.
9. Press F5 to run the script.

   The console pane displays the progress of the script. When the script has completed successfully, information like the following will display:

```
...
Container IP Address: 172.22.147.63
Container Hostname  : mybc
Container Dns Name  : mybc
Web Client          : http://mybc/BC/
Dev. Server         : http://mybc
Dev. ServerInstance : BC

Files:
http://test:8080/ALLanguage.vsix

Initialization took 116 seconds
Ready for connections!
Reading CustomSettings.config from mybc
Creating Desktop Shortcuts for mybc
```

10. Write down or copy the following parameter/values from the console: `Dev. Server`, `Dev. ServerInstance`, and `Files`. You will need this information later to set up Visual Studio Code for extension development.

You now have a container sandbox set up on your computer. The following shortcuts have been added to your desktop:

- **<Container name> Web Client** - opens the Web client for the your application in the container.
- **<Container name> PowerShell Prompt** - opens a Windows PowerShell prompt in the container. This gives you access to the Dynamics NAV (/powershell/business-central/overview), which you can run against the container sandbox environment.
- **<Container name>Command Prompt** - opens a Windows command prompt in the container.

For more information about working with a container sandbox, see Running a Container-Based Development Environment.

Set up Visual Studio Code

After the container sandbox is set up, you must set up Visual Studio Code for extension development. To do this,

you need the values for `Dev. Server`, `Dev. ServerInstance`, and `Files` parameters that you retrieved from the Windows PowerShell ISE console when you ran the **CreateBCSandbox.ps1** script.

1. In Visual Studio Code, go to **Extensions**, and install the AL Language extension from the Marketplace. You now have the AL Language extension enabled.
2. In Visual Studio Code, press **Ctrl+Shift+P** and then choose **AL Go!**.
3. Choose where to create the project, and then choose the **Your own server** option.
4. Open the generated `launch.json` file, update the `"server"` setting with the value of the `Dev. Server` parameter and the `"serverInstance"` setting with the value of the `Dev. ServerInstance` to reflect the container you just created. For example:

   ```
   "server": "http://mybc",
   "serverInstance": "BC",
   "authentication": "Windows",
   ```

5. Save the launch.json file.

You have now set up Visual Studio Code with the AL Language extension.

See Also

Running a Container-Based Development Environment
Working with Sandboxes and Entitlements
Sandbox Environments for Dynamics 365 Business Central Development
AL Development Environment

Working with Development Sandboxes and Entitlements

2/17/2021 • 4 minutes to read • Edit Online

The experience that a user has in Dynamics 365 Business Central depends on the purchased subscription plan. In Dynamics 365 Business Central, there are two main plans; the Essential and the Premium plan, plus a few more. For more information, see Licensing in Dynamics 365 Business Central. For detailed information about the Essential and Premium plans, see Business Central on the Microsoft Dynamics 365 site.

When you develop in a Docker sandbox, the Essential experience is automatically assigned to you (you set the experience on the **Company Information** page), which makes it difficult to test how a user with the Premium plan assigned will experience what you have developed.

> **NOTE**
> There is no license check in a Docker Sandbox except for on Purchase and Sales documents. There is a different behavior in these documents as the **TEAMMEMBER** license has partial access. In particular Invoices, Orders, Quotes and Credit Memos share the same table and the **TEAMMEMBER** license has access only to Quotes.

Setup for users with different plans

To mimic users with a specific subscription plan assigned, you can set them up with the user groups as detailed in the table below. When you add user to the group, the permission sets defined for the group will apply to the user. For more information, see To group users in user groups.

> **NOTE**
> In the table below *non-default* means not assigned by default, but the plan allows this to be assigned to the user.

USER NAME THE TYPE OF SUBSCRIPTION PLAN ASSIGNED TO THE GIVEN USER	USER GROUPS
EXTERNAL ACCOUNTANT Dynamics 365 Business Central External Accountant	D365 EXT. ACCOUNTANT D365 EXTENSION MGT (non-default) D365 TROUBLESHOOT (non-default) D365 SECURITY (non-default)
PREMIUM Dynamics 365 Business Central Premium Dynamics 365 Business Central for IWs	D365 BUS PREMIUM D365 EXTENSION MGT (non-default) D365 TROUBLESHOOT (non-default) D365 SECURITY (non-default)
ESSENTIAL Dynamics 365 Business Central Essential	D365 BUS FULL ACCESS D365 EXTENSION MGT (non-default) D365 TROUBLESHOOT (non-default) D365 SECURITY (non-default)

USER NAME THE TYPE OF SUBSCRIPTION PLAN ASSIGNED TO THE GIVEN USER	USER GROUPS
INTERNALADMIN Internal Administrator (Microsoft 365 Global administrator role)	D365 INTERNAL ADMIN D365 TROUBLESHOOT D365 BACKUP/RESTORE D365 SECURITY (non-default)
TEAMMEMBER Dynamics 365 for Team Members	D365 TEAM MEMBER D365 TROUBLESHOOT (non-default) D365 SECURITY (non-default)
DEVICE Dynamics 365 Business Central Device	D365 FULL ACCESS D365 EXTENSION MGT (non-default) D365 BUS PREMIUM (non-default)* D365 TROUBLESHOOT (non-default) D365 SECURITY (non-default) *) Please note: usage need to be according to terms in Licensing Guide
DELEGATEDADMIN Delegated Admin agent - Partner Delegated Helpdesk agent - Partner	D365 EXTENSION MGT D365 FULL ACCESS D365 RAPIDSTART D365 BACKUP/RESTORE D365 TROUBLESHOOT D365 SECURITY (non-default)

TIP

For more information about how to choose a user experience, see Changing Which Features are Displayed.

Assigning the Premium plan to test users

Depending on how you are running your Docker sandbox, you assign the experience in different ways.

Azure VMs

If you use https://aka.ms/bcsandbox to create your Dynamics 365 Business Central Sandbox Container Azure VM, the Azure Resource Manager template has two fields; **Assign Premium Plan** and **Create Test Users**, which by default are set to **Yes**.

Assign Premium Plan specifies whether or not your admin user should be assigned a Premium plan. **Create Test Users** specifies whether or not you want the setup to include test users.

BCContainerHelper

If you are using `New-BCContainer` to create your Dynamics 365 Business Central Sandbox container, you must make sure that you are using version 0.2.8.3 or later.

Use the switch `assignPremiumPlan` on `New-BCContainer` like this:

```
New-BCContainer -accept_eula -updateHosts -containerName test -artifactUrl (Get-BCArtifactUrl -country us) -
assignPremiumPlan
```

This assigns the Premium plan to your default admin user. Internally this just adds a record to the **User Plan** table.

To create the test users, you must call the `Setup-BCContainerTestUsers` method:

```
Setup-BCContainerTestUsers containerName test -tenant default -password $securePassword
```

specifying the container and the password that you want to use for the new users.

Internally, the `Setup-BCContainerTestUsers` downloads an app which exposes an API, publishes and installs the app, and then invokes the `CreateTestUsers` API with the password needed. After this, the app is uninstalled and unpublished.

If you want to see code behind the app, it is available here.

Docker run

If you are using Docker run to run your containers, you have a little more work to do.

First of all, you must override the `SetupNavUsers.ps1` by sharing a local folder to `c:\run\my` in the container and place a file called `SetupNavUsers.ps1` in that folder with the following content:

```
# Invoke default behavior
. (Join-Path $runPath $MyInvocation.MyCommand.Name)

Get-NavServerUser -serverInstance $ServerInstance -tenant default |? LicenseType -eq "FullUser" | ForEach-
Object {
    $UserId = $_.UserSecurityId
    Write-Host "Assign Premium plan for $($_.Username)"
    Invoke-Sqlcmd -ErrorAction Ignore -ServerInstance 'localhost\SQLEXPRESS' -Query "USE [$TenantId]
    INSERT INTO [dbo].[User Plan$63ca2fa4-4f03-4f2b-a480-172fef340d3f] ([Plan ID],[User Security ID]) VALUES
('{8e9002c0-a1d8-4465-b952-817d2948e6e2}','$userId')"
}
```

This will assign the Premium plan to the admin user in the database.

> **TIP**
> To set up test users, you can clone the createtestusers repository and modify the code to create the users on the `oninstall` trigger with the password that you want.

See Also

Programming in AL
Sandbox Environments for Dynamics 365 Business Central Development
Container Sandbox
Changing Which Features are Displayed
Production and Sandbox Environments

JSON Files

2/17/2021 • 15 minutes to read • Edit Online

In an AL project there are two JSON files; the `app.json` file and the `launch.json` file that are generated automatically when you start a new project. The `app.json` file contains information about the extension that you are building, such as publisher information and specifies the minimum version of base application objects that the extension is built on. Often the `app.json` file is referred to as the manifest. The `launch.json` file contains information about the server that the extension launches on.

> **NOTE**
> For information about data migration and creating a `migration.json` file, see The Migration.json File.

> **IMPORTANT**
> The `rad.json` and the `snapshots.json` files should not be modified.

App.json file

The following table describes the settings in the `app.json` file. For an example `app.json` file, see Business Central Performance Toolkit.

SETTING	MANDATORY	VALUE
id	Yes	The unique ID of the extension. When the `app.json` file is automatically created, the ID is set to a new GUID value. **Note:** The appId is used at runtime to bind table names contained in the application. Changing the appId will result in data from old tables not being used.
name	Yes	The unique extension name.
publisher	Yes	The name of your publisher, for example: **NAV Partner, LLC**.
brief	No, but required for AppSource submission	Short description of the extension.
description	No, but required for AppSource submission	Longer description of the extension.
version	Yes	The version of the app package.
privacyStatement	No, but required for AppSource submission	URL to the privacy statement for the extension.
EULA	No, but required for AppSource submission	URL to the license terms for the extension.

SETTING	MANDATORY	VALUE
help	No, but required for AppSource submission	URL to an online description of the extension. The link is used in AppSource and can be the same as the value of the `contextSensitiveHelpUrl` property or a different link, such as a link to your marketing page.
url	No, but required for AppSource submission	URL of the extension package.
logo	No, but required for AppSource submission	Relative path to the app package logo from the root of the package.
dependencies	No	List of dependencies for the extension package. For example: `"dependencies": [ {"id": "4805fd15-75a5-46a2-952f-39c1c4eab821", "name": "WeatherLibrary", "publisher": "Microsoft", "version": "1.0.0.0"},{}]` . **Note:** For dependencies to the System Application and Base Application these are no longer listed as explicit dependencies, but captured in the `application` setting as a reference to the application package. Must be filled in with the version number of the Application package. See `application` below. **Note:** The version specified defines the minimum version for the dependency. At runtime and when downloading symbols, the latest version of the dependency satisfying the specified name, publisher and, minimum version will be returned. When `runtime` is set to 4.0 or earlier, use `appId` instead of `id`.
screenshots	No	Relative paths to any screenshots that should be in the extension package.
platform	Yes, if system tables are referenced in the extension	The minimum supported version of the platform symbol package file, for example: "16.0.0.0". See the Symbols for the list of object symbols contained in the platform symbol package file.

SETTING	MANDATORY	VALUE
application	Yes, if base application is referenced in the extension	The supported version of the system and base application package file, for example: "16.0.0.0". The file name of this reference is Microsoft_Application.app and the `name` is `Application`. For code-customized base applications, the Microsoft_Application.app file can be modified to reference the code-customized base application instead. It is important to keep `"name": "Application"` in the extension, but information about publisher can be changed and the .app file can be renamed. For more information, see The Microsoft_Application.app File.
idRange	Yes	For example: `"idRange": {"from": 50100,"to": 50149}`. A range for application object IDs. For all objects outside the range, a compilation error will be raised. When you create new objects, an ID is automatically suggested.
idRanges	Yes	For example: `"idRanges": [{"from": 50100,"to": 50200},{"from": 50202,"to": 50300}]`. A list of ranges for application object IDs. For all objects outside the ranges, a compilation error will be raised. When you create new objects, an ID is automatically suggested. You must use *either* the `idRange` *or* the `idRanges` setting. Overlapping ranges are not allowed and will result in a compilation error.
showMyCode	No	This is by default set to `false` and not visible in the manifest. To enable viewing the source code when debugging into an extension, add the following setting: `"showMyCode": true`

SETTING	MANDATORY	VALUE
target	No	By default this is `Cloud`. The setting currently has the following options: `Internal`, `Extension`, `OnPrem`, and `Cloud`. The `Internal` and `Extension` settings are being deprecated with runtime 4.0 and replaced by the `OnPrem` and `Cloud` respectively. For on-premises, you can set this to `OnPrem` to get access to otherwise restricted APIs and .NET Interop. The Business Central Server setting must then also be set to `OnPrem`. **Note:** System tables that have the Scope property set to `Internal` / `OnPrem` cannot be accessed from extensions that have `target` set to `Cloud` / `External` through direct reference or through RecordRef. For more information, see Compilation Scope Overview
contextSensitiveHelpUrl	No, but required for AppSource submission	The URL for the website that displays context-sensitive Help for the objects in the app, such as `https://mysite.com/documentation/`. If the app does not support all locales currently supported by Business Central, then include a parameter for the locale in this URL, `/{0}/`, and also specify the relevant locales in the `supportedLocales` setting.
helpBaseUrl	No	The URL for the website that overtakes all Help for the specified locales. This property is intended for localization apps specifically since the setting overwrites the default URL of `/{0}/dynamics365/business-central`. If you set this value, you must also specify one or more languages in the `supportedLocales` setting.
supportedLocales	No	The list of locales that are supported in your Help if different from all locales. The value on the list is inserted into the URL defined in the `contextSensitiveHelpUrl` and `helpBaseUrl` properties. The first locale on the list is default. An example is `"supportedLocales": ["da-DK", "en-US"]` for an app that supports only Danish and English (US).

SETTING	MANDATORY	VALUE
runtime	Yes	The version of the runtime that the project is targeting. The project can be published to the server with an earlier or the same runtime version. The available options are: `1.0` - Business Central April 2018 Release `2.0` - Business Central Fall '18 Release `3.0` - Business Central Spring '19 Release `4.0` - Business Central 2019 release wave 2 `5.0` - Business Central 2020 release wave 1 `6.0` - Business Central 2020 release wave 2 `6.1` - Business Central 2020 release wave 2 update 17.1 `6.2` - Business Central 2020 release wave 2 update 17.2 `6.3` - Business Central 2020 release wave 2 update 17.3 `6.4` - Business Central 2020 release wave 2 update 17.4
features	No	Specifies a list of options. The `TranslationFile` option generates a `\Translations` folder that is populated with the .xlf file that contains all the labels, label properties, and report labels that you are using in the extension. The `GenerateCaptions` option depends on the `TranslationFile` setting. It generates captions for objects that do not have a `Caption` or `CaptionML` specified, these are then written to the .xlf file. The `GenerateLockedTranslations` flag is used to generate `<trans-unit>` elements in the XLIFF file for locked labels. The syntax is `"features": [ "TranslationFile", "GenerateCaptions", "GenerateLockedTranslations" ]`. For more information, see Working with Translation Files. When the `NoImplicitWith` flag is specified, `ImplicitWith` will be disabled by default. This flag is useful when all code has been rewritten to avoid any future usage of `ImplicitWith`. For more information, see Pragma ImplicitWith and Deprecating Explicit and Implicit With Statements.

SETTING	MANDATORY	VALUE
internalsVisibleTo	No	Specifies a list of modules that have access to the objects that are marked as `Internal` using the **Access** property from the current module. The syntax is `{ "appId": "d6c3f231-08d3-4681-996f-261c06500e1a", "name": "TheConsumer", "publisher": "Microsoft"}]`. For more information see Access Property and InternalEvent Attribute. **Note:** Using `internalsVisibleTo` in Business Central online will throw a warning from AppSourceCop and PTECop. `Access = Internal` is *not* designed as a security boundary, but for API development.
propagateDependencies	No	Specifies whether the dependencies of this project should be propagated as direct dependencies of projects that depend on this one. Default is `false`. If set to `true` then any dependencies of the current package will be visible to consumers of the package. For example, if A depends on B that depends on C, by default, A will not be able to use types defined in C. If B has `"propagateDependencies" : "true"`, then A will be able to use types defined in C without taking a direct dependency. **Note:** `propagateDependencies` applies to all dependencies, there is no option to exclude specific dependencies.
preprocessorSymbols	No	Defines any symbols to use with preprocessor directives. The syntax is `"preprocessorSymbols": [ "DEBUG" ]`. For more information, see Preprocessor Directives in AL.
applicationInsightsKey	No	The instrumentation key of the Azure Application Insights resource for monitoring operations, for example, like app secrets retrieval by extensions. For more information, see Monitoring and Analyzing Telemetry.
keyVaultUrls	No	List of URLs of key vaults that the extension from which the extension can retrieve secrets. For example: `"keyVaultUrls":[ "https://myfirstkeyvault.vault.azure.net "https://mysecondkeyvault.vault.azure.ne ]`. For more information, see App Key Vaults.

SETTING	MANDATORY	VALUE
suppressWarnings	No	Specifies that warnings issued by, for example, a specific analyzer rule should not be shown in the **Output** window. Syntax is `"suppressWarnings": [<warning ID>,<warning ID2>,...]`. For example `"suppressWarnings": [ "AL0458" ]`. It is also possible to use `#pragma` directives for suppressing warnings for specific areas of code. For more information, see Pragma Warning Directive and Suppressing Warnings.

Launch.json file

The following table describes the settings in the `launch.json` file. The `launch.json` file has two configurations depending on whether the extension is published to a local server or to the cloud.

Publish to local server settings

SETTING	MANDATORY	VALUE
name	Yes	"Your own server"
type	Yes	Must be set to `".al"`. Required by Visual Studio Code.
request	Yes	Request type of the configuration. Can be set to `"launch"` or `"attach"`. Required by Visual Studio Code. For more information, see Attach and Debug Next.
server	Yes	The HTTP URL of your server, for example: `"https://localhost|serverInstance"`
port	No	The port assigned to the development service.
serverInstance	Yes	The instance name of your server, for example: `"US"`
authentication	Yes	Specifies the server authentication method and can be set to `"UserPassword"`, `"Windows"`, or `"AAD"`. Currently, AAD authentication is supported only for Dynamics 365 Business Central sandboxes. AAD authentication cannot be used for on-premise servers.
startupObjectType	No	Specifies whether the object to open after publishing is a Page type (`"Page"`) or Table type (`"Table"`) object. The default is `"Page"`.

SETTING	MANDATORY	VALUE
startupObjectId	No	Specifies the ID of the object to open after publishing. Only objects of type Page and Table are currently supported.
schemaUpdateMode	No	Specifies the data synchronization mode when you publish an extension to the development server, for example: `"schemaUpdateMode": "Recreate"` The default value is Synchronize. For more information, see Retaining table data after publishing This feature is not supported in Dynamics NAV.
environmentType	No	Specifies which type of environment to use to connect to Business Central. Possible values are `OnPrem`, `Sandbox`, or `Production`.
environmentName	No	Specifies which named production or sandbox environment to use in cases where multiple sandboxes are owned by the same tenant.
breakOnError	No	Specifies whether to break on errors when debugging. The default value is `true`.
breakOnNext	No	Specifies the session type that the server will connect to. The options are: `WebserviceClient` - web API-based client including ODdata and SOAP clients, `WebClient` - standard web client, `Background` - background sessions, such as job queues, see Task Scheduler. This setting applies to Attach and Debug Next and to Snapshot Debugging. For *Attach* debugging, `breakOnNext` defines the next client session that the debug engine will attach to for the same user who has initiated an attach debug session from Visual Studio Code. For *Snapshot* debugging, `breakOnNext` defines the next session to hook AL code execution recording for a given user on a tenant, or if this is not specified with the userId in the configuration settings; the first user on the tenant.
breakOnRecordWrite	No	Specifies if the debugger breaks on record changes. The default value is `false`.

SETTING	MANDATORY	VALUE
launchBrowser	No	Specifies whether to open a new tab page in the browser when publishing the AL extension (Ctrl+F5). The default value is `false`. If the value is not specified or set to `true`, the session is started. If the value is explicitly set to `false`, the session is not started unless you launch your extension in debugging mode.
enableSqlInformationDebugger	Yes	Specifies whether the debugger shows the SQL information. The default value is `true`. For more information, see Debugging SQL behavior.
enableLongRunningSqlStatements	Yes	Specifies whether the debugger enables long running SQL statements in the debugger window.
longRunningSqlStatementsThreshold	Yes	Sets the number of milliseconds spent before a SQL statement is considered as long running in the debugger.
numberOfSqlStatements	Yes	Sets the number of SQL statements to be shown in the debugger.
dependencyPublishingOption	No	Available options are: `Default` - set dependency publishing will be applied `Ignore` - dependency publishing is ignored `Strict` - dependency publishing will fail if there are any apps that directly depend on the startup project and these apps are not part of the workspace. For more information, see Working with multiple projects and project references.
disableHttpRequestTimeout	No	Specifies if the default setting for HTTP request timeout in Visual Studio Code is switched off. The default value is `false`. If the value is set to `true` requests can run without timeout.

attach	No	Sets the session to attach to. There are two options;
		`Attach to the next client on the cloud sandbox`
		and
		`Attach to the next client on your server`
		. Use the first option to attach to a cloud session, and the second option to attach to a local server. For more information, see Attach and Debug Next.

SETTING	MANDATORY	VALUE
forceUpgrade	No	Always run upgrade codeunits, even if the version number of the extension is the same as an already installed version. This can be useful for troubleshooting upgrade issues. **Note:** The `forceUpgrade` setting requires the package ID to be changed.
useSystemSession	No	Runs install and upgrade codeunits in a system session. This will prevent debugging install and upgrade codeunits.
snapshotFileName	No	Specifies the snapshot file name used when snapshot debugging files are saved. For more information, see Snapshot Debugging.

Publish to cloud settings

SETTING	MANDATORY	VALUE
name	Yes	"Microsoft cloud sandbox"
type	Yes	Must be set to `"al"`. Required by Visual Studio Code.
request	Yes	Request type of the configuration. Must be set to `"launch"`. Required by Visual Studio Code.
startupObjectType	No	Specifies whether the object to open after publishing is a Page type (`"Page"`) or Table type (`"Table"`) object. The default is `"Page"`.
startupObjectId	No	Specifies the ID of the object to open after publishing. Only objects of type Page and Table are currently supported.
tenant	No	Specifies the tenant to which the package is deployed. If you specify multiple configurations, a drop-down of options will be available when you deploy. This parameter must contain a tenant AAD domain name, for example `mycustomer.onmicrosoft.com`.
environmentType	No	Specifies which type of environment to use to connect to Business Central. Possible values are `OnPrem`, `Sandbox`, or `Production`.

SETTING	MANDATORY	VALUE
environmentName	No	Specifies which named production or sandbox environment to use in cases where multiple sandboxes are owned by the same tenant.
applicationFamily	No (Yes for Embed apps)	The application family in the cloud server, for example `Fabrikam`. This property is reserved for Embed apps.
breakOnError	No	Specifies whether to break on errors when debugging. The default value is `true`.
breakOnNext	No	Specifies the session type that the server will connect to. The options are: `WebserviceClient` - web API-based client including OData and SOAP clients, `WebClient` - standard web client, `Background` - background sessions, such as job queues, see Task Scheduler. This setting applies to Attach and Debug Next and to Snapshot Debugging. For *Attach* debugging, `breakOnNext` defines the next client session that the debug engine will attach to for the same user who has initiated an attach debug session from Visual Studio Code. For *Snapshot* debugging, `breakOnNext` defines the next session to hook AL code execution recording for a given user on a tenant, or if this is not specified with the userId in the configuration settings; the first user on the tenant.
breakOnRecordWrite	No	Specifies if the debugger breaks on record changes. The default value is `false`.
launchBrowser	No	Specifies whether to open a new tab page in the browser when publishing the AL extension (Ctrl+F5). The default value is `false`. If the value is not specified or set to `true`, the session is started. If the value is explicitly set to `false`, the session is not started unless you launch your extension in debugging mode.
enableSqlInformationDebugger	Yes	Specifies whether the debugger shows the SQL information. The default value is `true`. For more information, see Debugging SQL behavior.

SETTING	MANDATORY	VALUE
enableLongRunningSqlStatements	Yes	Specifies whether the debugger enables long running SQL statements in the debugger window.
longRunningSqlStatementsThreshold	Yes	Sets the number of milliseconds spent before a SQL statement is considered as long running in the debugger.
numberOfSqlStatements	Yes	Sets the number of SQL statements to be shown in the debugger.
dependencyPublishingOption	No	Available options are: `Default` - set dependency publishing will be applied `Ignore` - dependency publishing is ignored `Strict` - dependency publishing will fail if there are any apps that directly depend on the startup project and these apps are not part of the workspace. For more information, see Working with multiple projects and project references.
disableHttpRequestTimeout	No	Specifies if the default setting for HTTP request timeout in Visual Studio Code is switched off. The default value is `false`. If the value is set to `true` requests can run without timeout.

attach	No	Sets the session to attach to. There are two options;
		`Attach to the next client on the cloud sandbox`
		and
		`Attach to the next client on your server`
		. Use the first option to attach to a cloud session, and the second option to attach to a local server. For more information, see Attach and Debug Next.

See Also

AL Development Environment
Debugging in AL
Security Setting and IP Protection
AL Language Extension Configuration
Configure Context-Sensitive Help
App Key Vaults

The Migration.json File

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

Data migration allows you to move table and field data between extensions. The `migration.json` file provides a pointer to the ID of an app that one or more tables will be moved to. This allows you to move table and field data from, for example, a code-customization on the base application to an extension of the base application. The `migration.json` file can be added into the app project of an extension that a table is moved from to specify the ID of the app that the table will be moved to. It can, for example, be placed at the root of the AL project. The `migration.json` file must be created manually following the steps and syntax as described below.

In the extension `app.json` file, ensure that `"target": "OnPrem"`. For more information, see JSON Files.

Creating the migration.json file

1. In the root folder of the app project that will migrate data to a different app project, choose **New File**.
2. Name the file `migration.json`.
3. Edit the file by adding one or more IDs inside the `"apprules":[]` section, such as the following:

```
{
    "apprules": [
        {
            "id": "12345678-abcd-abcd-abcd-1234567890ab"
        }
    ]
}
```

4. Save the `migration.json` file in the project.

You now have the migration file in place for the data migration from one app project to another. This will be used for performing the data migration steps. For more information, see Migrating Tables and Fields Between Extensions.

See Also

JSON Files
Migrating Tables and Fields Between Extensions

AL Language Extension Configuration

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

The AL Language extension has a number of settings that can be specified for a specific user or for a workspace. To activate the settings, press **Ctrl+Shift+P**, and then choose **Preferences: Open Settings (UI)** for workspace settings, or choose **Preferences: Open User Settings** for user settings. Under **Extensions**, and **AL Language extension configuration** you will find the settings that are available for the AL Language extension. For tips on how to optimize Visual Studio Code, see Optimizing Visual Studio Code for AL Development.

Settings

The following table describes the user and workspace settings for the AL Language extension:

SETTING	VALUE
Assembly Probing Paths	Sets the list of directory paths where the compiler searches for referenced .NET assemblies. For example: `"al.assemblyProbingPaths": ["./.netpackages", "C:/Program Files/Assemblies"]`
Browser	Specifies the browser in which to open the Business Central client when launching the application from Visual Studio Code.
Code Analyzers	Sets the list of paths to code analyzers to use for performing code analysis. For example : `"al.codeAnalyzers": ["${AppSourceCop}", "${CodeCop}"]` .
Compilation Options	Specifies the compilation options; `generateReportLayout`, which controls whether report layout files are generated during build and `parallel`, which controls whether to use concurrent builds. Both options are default set to `true`.
Editor Services Log Level	Sets the logging verbosity level for the AL Language Editor Services host executable. Possible values are `Verbose`, `Normal`, `Warning`, and `Error`.
Editor Services Path	Specifies the path to the Editor Services host executable.
Enable Code Actions	Specifies whether code actions should be enabled for all source files in the current project. Default is `false`.
Enable Code Analysis	Specifies whether code analysis should be performed for all source files in the current project. Default is `false`. If this is set to `true`, you must specify the **Code Analyzers** setting with the list of code analyzers to use.

SETTING	VALUE
Incognito	Specifies whether to open the browser in Incognito/InPrivate mode when launching the application from Visual Studio Code. This option will take effect only if the **Browser** option is set to a non-default value.
Package Cache Path	Sets the directory path where reference symbol packages are located.
Rule Set Path	Sets the path to the file containing the customized rules to use when running code analysis.
Incremental Build	Specifies whether a project, when it is built using **Ctrl+Shift+B**, **Ctrl+F5**, or **F5**, will reuse the last known tracked compilation which will enhance the compilation time significantly. For more information about project to project references, see Working with multiple projects and project references. **Note:** Setting this to `true` will not do an end-to-end build, as it is depending on an already-compiled state. To get a clean, full build, this flag must be set to `false`. Default is `false`.
Snapshot Output Path	Sets the directory path where snapshot files are saved. Default is `./.snapshots`.
Snapshot Debugging Path	Sets the directory path where the snapshot debugger sources are located. Default is `./.snapshot`.

See Also

AL Development Environment
Debugging in AL
JSON Files
Working with multiple projects and project references

Security Setting and IP Protection

2/17/2021 • 2 minutes to read • Edit Online

When developing an extension, your code is by default protected against downloading or debugging. Read below about the security setting and adding Intellectual Property (IP) protection against downloading or debugging into an extension to see the source code in the extensions.

The extension development package provides a pre-configured setting for IP protection against viewing or downloading the code of the extensions. However, this setting can also be controlled in the manifest; the `app.json` file.

IP protection setting

When you start a new project, an `app.json` file is generated automatically, which contains the information about the extension that you are building on. The `app.json` file contains a setting called `showMyCode`, which controls whether it is possible to debug into the extension, when that extension is taken as a dependency. The default value of this property is set to **false**. This means that debugging into an extension or going to definition to view the code is not allowed. For a more refined setting, you can specify the `NonDebuggable` attribute on methods and variables. For more information, see NonDebuggable Attribute.

`showMyCode` does not apply to Profiles, Page Customizations and Views, because these objects cannot define any custom logic in procedures or triggers. The code for Profiles, Page Customizations, and Views defined in an extension with `showMyCode` set to **false** can then still be accessed and copied using Designer.

> **NOTE**
> The `showMyCode` setting is not visible in the `app.json` file when it is generated. If you want to change the value, you must add the setting as shown in the code snippet below.

> **NOTE**
> Even though `showMyCode` is set to **false**, you will still be able to view that code if an extension is deployed through Visual Studio Code, as opposed to deploying using a cmdlet or via AppSource.

Changing the IP protection setting

If you want to allow debugging into an extension to view the source code, you can add the `showMyCode` property in the `app.json` file and set the property value to **true**. For example, if a developer develops extension A and he or someone else on the team develops extension B, and B depends on A, then debugging B will only step into the code for A if a method from A is called and if the `ShowMyCode` flag is set to **true** in the app.json for extension A as shown in the example below:

```
"showMyCode": true
```

By adding this setting, you *enable debugging* into an extension to view the source code when that extension is set as a dependency.

See Also

JSON Files
AL Development Environment
NonDebuggable Attribute

Developing for Multiple Platform Versions

2/17/2021 • 2 minutes to read • Edit Online

The AL language extension is compatible with multiple platform versions. You can install the AL Language extension from the Visual Studio Code marketplace and use it to develop solutions for Dynamics 365 Business Central.

Defining the platform version

To set the platform version, add the **runtime** property in the `app.json` file. This attribute defines the platform version that the extension is targeting. Depending on the platform version, some features become available, while some features are not supported. For example, OData-bound actions can only be used when the platform version is 2.0 or higher.

> **NOTE**
> The AL Language extension is not compatible with Dynamics NAV 2018 version backwards. For Dynamics NAV 2018 development, the traditional method should be used. You must install the Visual Studio Code extension from the `ALLanguage.vsix` file shipped on the DVD.

Version compatibility

The following two elements are compared when you publish an extension.

1. The runtime version of the extension defined in the app.json file.
2. The runtime version of the platform that the extension is targeting.

In the app.json file, set the extension **runtime** version lower than the platform version. When you set the extension to a higher **runtime** version, the extension package may contain certain features that the platform may not support which would result in an error. Therefore, you must lower the extension runtime version than the one that platform supports in order to publish your extension.

Things to be aware of

1. An error will be thrown when you publish an extension with a higher runtime version than the one that platform supports. For example, if you set the runtime value to `2.0`, you get the following error message.

   ```
   The runtime version of the extension package is currently set to '2.0'. The runtime version must be set to '1.0' or earlier in the app.json file in order to install the extension package on this platform.
   ```

2. When you lower the extension runtime version, you may get warnings about the newest features not supported by the earlier versions of the platform.

3. A best-effort compilation is made when you publish an extension compiled with a lower runtime version. This is allowed in order to avoid recompilation of the extension package every time you upgrade the platform.

See Also

Optimizing Visual Studio Code for AL Development

2/17/2021 • 2 minutes to read • Edit Online

Visual Studio Code is built to handle many smaller, dependent projects, and not one large project, however, as the base application is not yet split into modules or components that allows managing the code in smaller projects, we recommend the following performance optimizations.

Open your `settings.json` file in the project (or global settings if you prefer that) pressing **Ctrl+Shift+P**. Set:

- `"al.enableCodeAnalysis": false` to turn off code analysis completely, read more here Using the Code Analysis Tool.
- `"al.backgroundCodeAnalysis": false` to turn off running code analysis in the background, but code analysis will be enabled when building with **Ctrl+Shift+B**. This is an alternative if analyzers are required with `"al.enableCodeAnalysis": true`.
- `"al.enableCodeActions": false` to turn off AL Code Actions, read more here AL Code Actions.
- `"al.incrementalBuild": true` to allow the compiler to reuse the existing background compilation for creating the package.
- `"editor.codeLens": false` to turn off code lens in Visual Studio Code, see Code Navigation.
- Add the build folder to the exclusion list for Windows Defender.

See also

Development in AL
Best Practices for AL

Compilation Scope Overview

2/17/2021 • 2 minutes to read • Edit Online

In Dynamics 365 Business Central there are different layers of controlling what can be published to the server and accessed from within a project.

Configuring extension target on the server

By setting the **Allowed Extension Target Level** flag in the server configuration, you control what can be published to the server by setting the **Cloud** or **OnPrem** flag. For more information, see Configuring Business Central Server.

Configuring extension target for an extension

In the manifest of an extension (the app.json file), you can set the **target** property to specify the compilation target of an extension. The available values are `Internal`, `Extension`, `OnPrem`, and `Cloud`.

> **IMPORTANT**
> The `Internal` and `Extension` values have been deprecated starting with runtime 4.0 and replaced by the `OnPrem` and `Cloud` respectively.

The `target` property informs the compiler which APIs can be used within the current project.

- If you specify `"target":"OnPrem"` you can use any platform APIs and .NET types. This is the most permissive target. It is only if you specify this target that you can use methods marked with `Scope('OnPrem')` or tables with the property Scope (Tables) set to `OnPrem`.
- If you specify `"target":"Cloud"` you can only use APIs that are safe for use in a cloud environment. If the target is set to `Cloud`, the extension cannot use any method marked with `Scope('OnPrem')` or table with the property `Scope` set to `OnPrem`, even if those elements are declared within the same extension. For example, if you have two extensions; A and B. Extension A has an app.json target setting `"target": "OnPrem"` and defines a method with the Scope Attribute set to `[Scope('OnPrem')]`. Extension B has a dependency on extension A and extension B has an app.json target setting `"target": "Cloud"` and tries to call the method in extension A marked with scope `OnPrem`, you will get a compile error when you are trying to compile extension B. For more information, see JSON Files.

Methods and tables scope

Methods can be marked with the `[Scope()]` attribute to specify the compilation target and tables can be marked with the `Scope` property. Both instruct the compiler which target the method or table can be used in. For more information, see Scope Attribute and Scope (Table) Property.

See Also

AL Development Environment
Developing Extensions in AL
JSON Files
Scope Attribute
Scope (Table) Property

Debugging

2/17/2021 • 7 minutes to read • Edit Online

The process of finding and correcting errors is called *debugging*. With Visual Studio Code and the AL Language extension you get an integrated debugger to help you inspect your code to verify that your application can run as expected. You start a debugging session by pressing **F5**. For more information about Debugging in Visual Studio Code, see Debugging.

An alternative to classic debugging, is snapshot debugging, which allows you to record running code, and later debug it. For more information, see Snapshot Debugging.

> **IMPORTANT**
> To enable debugging in versions before Business Central April 2019, the `NetFx40_LegacySecurityPolicy` setting in the Microsoft.Dynamics.Nav.Server.exe.config file must be set to **false**. This requires a server restart.

> **IMPORTANT**
> To use the development environment and debugger, you must make sure that port `7049` is available.

There are a number of limitations to be aware of:

- "External code" can only be debugged if the code has the `showMyCode` flag set. For more information, see Security Setting and IP Protection.
- The debugger launches a new client instance each time you press **F5**. If you close the debugging session, and then start a new session, this new session will rely on a new client instance. We recommend that you close the Web client instances when you close a debugging session.
- Pausing the debugging session is not supported.

To control table data synchronization between each debugging session, see Retaining table data after publishing.

> **TIP**
> To be able to debug an online environment with an Embed app published in it, make sure to specify the `applicationFamily` parameter in your launch.json file. You define the application family for your Embed app during onboarding.

Breakpoints

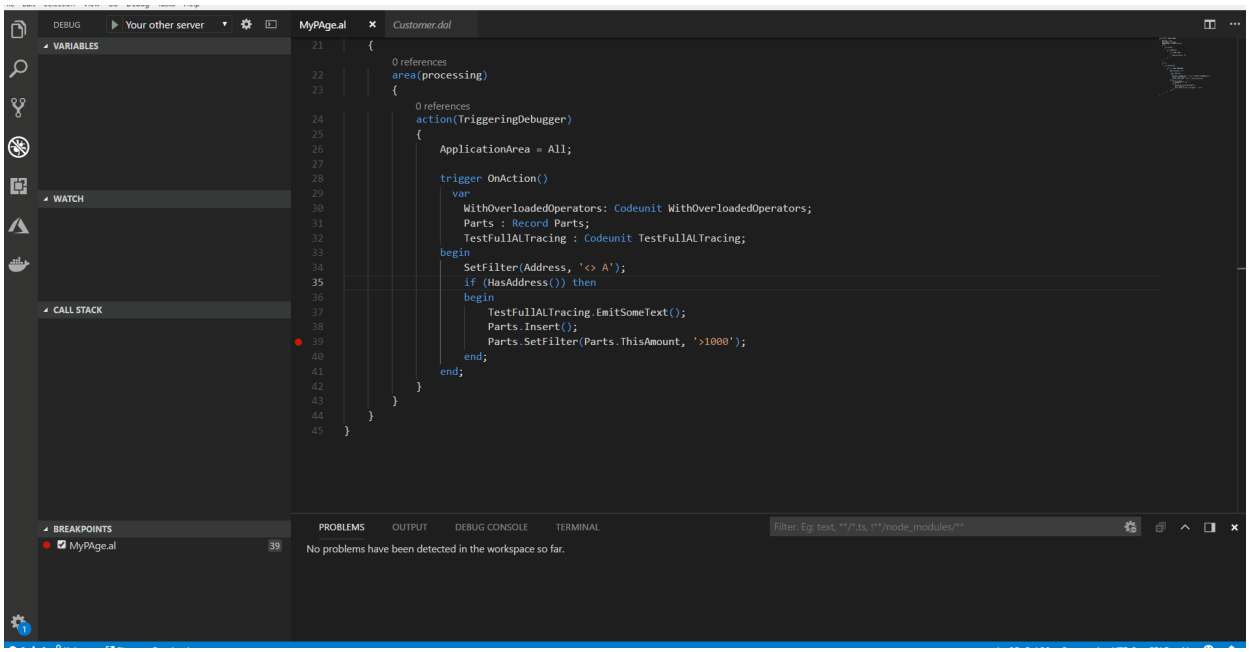
The basic concept in debugging is the *breakpoint*, which is a mark that you set on a statement. When the program flow reaches the breakpoint, the debugger stops execution until you instruct it to continue. Without any breakpoints, the code runs without interruption when the debugger is active. You can set a breakpoint by using the **Debug Menu** in Visual Studio Code. For more information, see Debugging Shortcuts.

Set breakpoints on the external code that is not part of your original project. You can step into the base application code by using the **Go to Definition** feature, and set breakpoints on the referenced code which is generally a `.dal` file. To set a breakpoint on the external code or base application code, you do the following:

- Use **Go to Definition** which opens the "external file" and then a breakpoint can be set.

- Using the debugger, step into the code, and then set a breakpoint.

In the following video illustration, the `Customer.dal` is an external file. A breakpoint is set in the `Customer.dal` file which is referenced from your AL project to stop execution at the marked point.



For more information about **Go to Definition**, see AL Code Navigation.

Break on errors

Specify if the debugger breaks on the next error by using the `breakOnError` property. If the debugger is set to `breakOnError`, then it stops execution both on errors that are handled in code and on unhandled errors.

The default value of the `breakOnError` property is **true**, which means the debugger stops execution that throws an error by default. To skip the error handling process, set the `breakOnError` property to **false** in the `launch.json` file.

> **TIP**
> If the debugging session takes longer, you can refresh the session by pressing the **Ctrl+Shift+P** keys, and select **Reload Window**.

Break on record changes

Specify if the debugger breaks on record changes by using the `breakOnRecordWrite` property. If the debugger is set to break on record changes, then it breaks before creating, modifying, or deleting a record. The following table shows each record change and the AL methods that cause each change.

RECORD CHANGE	AL METHODS
Create a new record	Insert Method (Record)
Update an existing record	Modify Method (Record), ModifyAll Method (Record), Rename Method (Record)
Delete an existing record	Delete Method (Record), DeleteAll Method (Record)

The default value of the `breakOnRecordWrite` property is **false**, which means that the debugger is not set to break on record changes by default. To break on record changes, you can set the `breakOnRecordWrite` property to **true** in the `launch.json` file. For more information, see JSON Files.

Debugging large size variable values

Variables that contain values that are larger than 1024 bytes are truncated (`...`) and cannot be fully inspected from the **VARIABLES** window. In order to inspect a large size variable value, instead use the **DEBUG CONSOLE** and write the name or qualified name of a variable to inspect at the prompt and then press **Enter**.

Attach and Debug Next

If you do not want to publish and invoke functionality to debug it, you can instead attach a session to a specified server and await a process to trigger the breakpoint you have set. For more information, see Attach and Debug Next.

Debugging shortcuts

KEYSTROKE	ACTION
F5	Start debugging
Ctrl+F5	Start without debugging
Shift+F5	Stop debugging
Ctrl+Shift+F5	Start debugging without publishing. Using this command on a changed, but not published code may trigger false existing breakpoints. For example, if you modify method "foo", add two lines and put a breakpoint on the second line and then start debugging without publishing, that breakpoint will not be hit, or if it is hit is not your new code that it breaks. If it breaks, it will break on the line that the server thinks the breakpoint is, based on the last published code.
Alt+F5	Start RAD with debugging. For more information, see Working with Rapid Application Development.
F10	Step over
F11	Step into
Shift+F11	Step out
F12	Go To Definition

For more shortcuts, see Debugging in Visual Studio Code. For working with Snapshot Debugging, see Snapshot Debugging.

Debugging SQL behavior

Traditionally, debugging AL has been about examining behavior of the language runtime, for example, looking into the content of local variables at a breakpoint. As of Business Central April 2019, the AL debugger also offers

the capability to examine the impact that your AL code has on the Business Central database. The `enableSQLInformationDebugger` setting enables this functionality. For more information, see JSON Files.

View database statistics

In the **VARIABLES** pane in debugger, expand the **<Database statistics>** node to get insights such as the current network latency between the Business Central Server and the Business Central database, the total number of SQL statements executed, and the total number of rows read, as well as insights into the most recent SQL statements executed by the server. The following insights are part of the database statistics:

INSIGHT	DESCRIPTION
Current SQL latency (ms)	When the debugger hits a breakpoint, the Business Central Server will send a short SQL statement to the database and measure how long time it takes. The value is in milliseconds.
Number of SQL Executes	This number shows the total number of SQL statements executed in the debugging session since the debugger was started.
Number of SQL Rows Read	This number shows the total number of rows read from the Business Central database in the debugging session since the debugger was started.

> **TIP**
> You can also get database insights from the AL runtime by using the SqlStatementsExecuted() and SqlRowsRead() methods.

View SQL statement statistics

The database insights also let you peek into the most recent and the latest long running SQL statements executed by the server. To view a list if these, expand either the **<Last Executed SQL Statements>** or **<Last Long Running SQL Statements>** node. The following insights are part of the SQL statement statistics:

INSIGHT	DESCRIPTION
Statement	The SQL statement that the AL server sent to the Business Central database. You can copy this into other database tools, such as SQL Server Management Studio, for further analysis.
Execution time (UTC)	The timestamp (in UTC) of when the SQL statement was executed. You can use this to infer whether the SQL statement was part of the AL code between current and last breakpoint (if set).
Duration (ms)	The duration in milliseconds of the total execution time of the SQL statement measured inside the Business Central Server. You can use this to analyze whether you are missing indexes (Business Central keys), or to experiment with performance of database partitioning and/or compression.
Approx. Rows Read	This number shows the approximate number of rows read from the Business Central database by the SQL statement. You can use this to analyze whether you are missing filters.

The number of SQL statements tracked by the debugger can be configured in the Business Central Server. The

default value is 10.

> **NOTE**
>
> For Business Central on-premises, the Business Central Server instance has several configuration settings that control the SQL statistics that are gathered and then displayed in debugger, like whether long running SQL statements or SQL statements are shown. If you are not seeing the insights that you expect to see in debugger, check the server configuration. For more information, see Configuring Business Central Server.

NonDebuggable attribute

The ability to debug certain methods and/or variables can be restricted. For more information, see NonDebuggable Attribute.

See Also

Attach and Debug Next
Developing Extensions
JSON Files
AL Code Navigation

Snapshot Debugging

2/17/2021 • 8 minutes to read • Edit Online

> **NOTE**
> With Business Central 17.2 - Snapshot Debugging is available in production cloud environments.

Snapshot debugging allows a delegated admin to record AL code that runs on the server, and once it has run, debug the recorded *snapshot* in Visual Studio Code. For a delegated admin to create and download a snapshot file that exists on the server on behalf of an end-user, the delegated admin must be part of the **D365 Snapshot Debug** permission group. For more information, see Assign Permissions to Users and Groups. One of the advantages of snapshot debugging is that it provides the ability to inspect code execution and variables in the production environment in a cloud service, on a specified user session.

Snapshot debugging introduces the concept of *snappoints*. A snappoint is a breakpoint in Visual Studio Code that is set when creating a snapshot, they do not, however, stop execution of code like when using regular debugging. Snappoints instruct execution to log the state at the breakpoint for later offline inspection. Snapshot debugging will record AL code as it runs on the server, but will only collect variable information on:

- Snappoints
- AL exceptions

> **IMPORTANT**
> To enable snapshot debugging it is very important that the symbols on the tenant match the symbols on the server. This is not automatically detected, and must be manually checked. In this release, you can ensure this by copying the specific sandbox and download symbols from that copy. Furthermore, any code that snappoints are set in, must have been deployed, otherwise debugging will not work. For more information, see the section Downloading symbols on the snapshot debugger endpoint.

Snapshot debugging keyboard shortcuts

In the following sections you can read more about how to initialize, view the status, and finalize a snapshot debugging session. For these actions, the following keyboard shortcuts are useful to familiarize yourself with. For additional keyboard shortcuts, see Keyboard Shortcuts.

KEYBOARD SHORTCUT	ACTION
F7	Start a snapshot debugging session
Shift+F7	List all available snapshots
Alt+F7	Finish a snapshot debugging session

Initializing a snapshot debugging session

From Visual Studio Code, you start a snapshot by creating a snapshot configuration file. There are two template configurations for a snapshot, which are accessed by selecting **Add Configuration** in Visual Studio Code.

- AL: Initialize a snapshot debugging session locally
- AL: Initialize a snapshot debugging session on cloud

Choose whether to run the session on a cloud service or locally. The configuration file will now contain the following information:

SETTING	DESCRIPTION
`userId`	The GUID of the user on whose behalf a snapshot debugging will be started. For on-premises, this can also be the user name in user password authentication scenarios. The user must be able to start, or have a session type opened that is specified in the `breakOnNext` parameter. For more information, see JSON Files.
`sessionId`	A session ID for the user specified above in `userId`.
`snapshotVerbosity`	Determines how much execution context to be recorded. If **SnapPoint** is specified, then only methods that hit a snappoint will be recorded.

When a configuration is defined, a snapshot debugging session can be initialized by pressing **Ctrl+Shift+P** and then selecting **AL:Initialize Snapshot Debugging** or by pressing **F7**.

To record the AL execution, the server will now wait for a connection to happen where the following rules apply:

- If a `sessionId` is specified for a userId for a given tenant then it will be that session that will be snapshot debugged.
- If only a `userId` is specified for a given tenant then the next session that is specified in the `breakOnNext` configuration parameter is snapshot debugged.
- If no `userId` is specified then the next session on a given tenant that validates the `breakOnNext` parameter will be snapshot debugged.

Once a snapshot debugging session is initialized the snapshot debugging session counter on the status-bar will be updated and look like this:

Status of a snapshot debugging session

Clicking on the status bar icon or pressing **Shift+F7** will bring up a list of all available snapshots.
The status list will show the state of a snapshot-debugged session.

A snapshot debugging session can be in one of the three states:

- **Initialized** - A request is issued and the server is waiting for the next session to be snapshot debugged based on the above rules.
- **Started** - You have attached to an end-user session to snapshot debug.
- **Finished** - When the snapshot debugging session has finished.
- **Downloaded** - When the snapshot file is downloaded.

Finishing a snapshot debugging session

You finish a snapshot debugging session by pressing **Alt+F7**. This brings up all snapshot sessions that have

been started. Choosing one will close the session debugging on the server and download the snapshot file.

> **IMPORTANT**
> The snapshot file can contain customer privacy data and must therefore be handled according to privacy compliance and should be deleted when it is not needed anymore.

Snapshot debugging sessions that have produced a snapshot file can be debugged. The location of a snapshot file is controlled by the `al.snapshotOutputPath` configuration parameter. By default it is local to the current workspace and it is called `./.snapshots`. For more information, see AL Language Extension Configuration.

Downloading symbols on the snapshot debugger endpoint

In order to download symbols on a production server, you need permission related entries:

- Be a delegated admin
- The read-only access to the **Published Application** table emphasized in the **D365 EXTENSION MGT** permission set should also be granted.

Debugging requires that symbols on the server are matched with the symbols that the user has locally. If this is not the case, and you set a breakpoint on a given line in Visual Studio Code, the line of code may differ from what is on the server.

Symbols download is using the **snapshotInitialize** debug configuration settings in Visual Studio Code, which is set up when you choose either **AL: Initialize a snapshot debugging session locally** or **AL: Initialize a snapshot debugging session on cloud**.

```
{
            "name": "snapshotInitialize: MyServer",
            "type": "al",
            "request": "snapshotInitialize",
            "environmentType": "OnPrem",
            "server": "http://localhost",
            "serverInstance": "BC170",
            "authentication": "UserPassword",
            "breakOnNext": "WebClient"
        },
```

> **IMPORTANT**
> Debugging requires that symbols on the server are matched with the symbols that the user has locally. If this is not the case, and you set a breakpoint on a given line in Visual Studio Code, the line of code may differ from what is on the server. This is why you must download symbols from production servers for snapshot debugging in order for a breakpoint set on one line to match with what the server understands of this line. This is to avoid a scenario where you set a breakpoint in a DAL file on line 12, but line 12 on the server is an empty line or a completely different line if the symbols are not the same.

Debugging a snapshot file

There are two user actions that will start snapshot debugging:

- Creating a new launch debug configuration and specifying the snapshot file name in the `snapshotFileName` configuration setting. This is the only setting that is needed besides the type, request, and name.
- Clicking on the status icon or by pressing **Shift+F7** and selecting a finished snapshot-debugged session.

Once a snapshot debugging session starts in Visual Studio Code, code execution will stop at the first snappoint. AL exceptions will be treated as snappoints, with the only difference that they cannot be removed by user actions. Other snappoints are just regular breakpoints that can be removed or re-added by user actions. If no snappoints are specified the first recorded methods; the first line is the entry breakpoint.

The user can set breakpoints and continue execution to that breakpoint for testing, for example, if a line is hit, but it is the snappoint that carries the real information.

Snapshot debugging versus regular debugging

Snapshot debugging is almost the same as a regular debugging with the differences mentioned in the following:

SNAPSHOT VERSUS REGULAR DEBUGGING
Breakpoints can be added and removed and they will be hit if given a breakpoint; the breakpoint is in the execution context of a recorded state. This means that if walking the execution stack for a breakpoint and the next stepped line is reached, then the code will break on the breakpoint.
A snappoint is a breakpoint in Visual Studio Code that is set when creating a snapshot, they do not, however, stop execution of code like when using regular debugging. Snappoints instruct execution to log the state at the breakpoint for later offline inspection.
You can always navigate through all the breakpoints with **Continue** (F5). The order may not be the same as the execution order on the Business Central server. This is due to the fact that some calls on the server are AL calls with non-walkable stacks. Some are direct server calls on the server like triggers. A snapshot debugging session on the Business Central server can only record AL calls and walk AL stack traces.
This is also true when stepping. The rule of thumb is that breakpoints within the reach are hit first, and if there are none; the next line is hit. Breakpoints on triggers may not always qualify as code within reach.
Variable data is only shown on snappoints.
If there are no frames available snapshot debugging will stop.
Stepping out of triggers with no recorded stack information will move execution to the first recorded method's first line. This may be very far from the user's execution of interest. For example, stepping out from an `OnOpenPage` trigger with a snappoint may land on deep inside base code execution where recording has started. Navigating with **F5** will start over breakpoint resolution, thus this is an exit strategy from a scenario like this.
A snappoint may resolve as a non-reachable breakpoint if there was no execution state on the server hitting the snappoint.
A snapshot debugger session with a Business Central server will be closed if not attached to after 30 minutes.
If a snapshot debugger session is started, it has to be finished after 10 minutes.

See Also

Debugging
Attach and Debug Next
Developing Extensions
JSON Files
AL Code Navigation
EnableLongRunningSQLStatements Property
EnableSQLInformationDebugger Property
LongrunningSQLStatementsThreshold Property

Attach and Debug Next

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

If you do not want to publish and invoke functionality to debug it, you can instead attach a session to a specified server and await a process to trigger the breakpoint you have set. Then debugging starts when the code that the breakpoint is set on is hit. This is particularly useful for debugging Web API sessions.

> **NOTE**
> To use the attach functionality, you must make sure that your app is published with **Ctrl+F5** first, or with **Alt+Ctrl+F5** for RAD publishing, before you start the debugging session with **F5**. To debug using attach, you must make sure to debug on a *new* session. Creating a new server session from the client can be achieved for example by launching a new client session. Pressing **F5** (Refresh) in a browser may not create a new server session, because it is cached, but if a session is expired and refreshed that will create a new session.

> **IMPORTANT**
> Only the user who starts a Visual Studio Code attach session can issue the Web request on the server.

Attach configuration

You activate the attach functionality by creating a new configuration in the `launch.json` file. The configuration has two flavors; **Attach to the next client on the cloud sandbox** and **Attach to the next client on your server**. Use the first option to attach to a cloud session, and the second option to attach to a local server.

In the attach configuration, the `breakOnNext` setting specifies the next client to break on when the debugging session starts and allows only one option. The available options are: `WebServiceClient`, `WebClient`, and `Background`. The example below illustrates a configuration for a local server.

```
...
{
            "name": "My attach to local server",
            "type": "al",
            "request": "attach",
            "server": "https://localhost",
            "serverInstance": "BC160",
            "authentication": "Windows",
            "breakOnError": true,
            "breakOnRecordWrite": false,
            "enableSqlInformationDebugger": true,
            "enableLongRunningSqlStatements": true,
            "longRunningSqlStatementsThreshold": 500,
            "numberOfSqlStatements": 10,
            "breakOnNext": "WebClient"
        }
...
```

Attach support

The following configurations for attach are supported:

BUSINESS CENTRAL	WEB CLIENT	WEB SERVICE CLIENT	BACKGROUND SESSION
On-premises	Supported	Supported	Supported
Sandbox	Not supported	Supported	Not supported

To start an attach session

1. In Visual Studio Code, under **Debug**, choose **Add configuration**.
2. Choose whether to attach to a cloud or a local session.
   The `launch.json` file is now populated with the correct attach configuration settings. If you selected a local session, change the default settings to point to your local server in the `server` and `serverInstance` settings.
3. Set `breakOnNext` to specify the client type on which to break.
4. In your code, set at least one breakpoint using **Debug** from the toolbar, choose **New breakpoint**, and then choose which type of breakpoint to add. You can always add more breakpoints while debugging.
5. It is important to make sure to publish your app by pressing **Ctrl+F5**, alternatively **Alt+Ctrl+F5** for RAD publishing. Your app *will not be* published if you only press **F5**.

   > **IMPORTANT**
   > If you modify the app code during the debugging session, make sure to re-publish the app using **Ctrl+F5**.

6. After publishing the app, press **F5** to start a debugging session.

   > **NOTE**
   > If you have more attach configuration settings, you must first select which configuration to start.

7. Debug and inspect the code. You can add more breakpoints while debugging.
8. Stop the attach debugging session by selecting **Detach** in the Visual Studio Code toolbar.

See Also

AL Development Environment
Developing Extensions in AL
Debugging
Snapshot Debugging
JSON Files
EnableLongRunningSQLStatements Property
EnableSQLInformationDebugger Property
LongrunningSQLStatementsThreshold Property
NumberOfSQLStatements Property

Working with Rapid Application Development

2/17/2021 • 2 minutes to read • Edit Online

Working with Visual Studio Code and Dynamics 365 Business Central you can benefit from Rapid Application Development (RAD) on large code projects. RAD allows faster development on projects with a large number of files by doing a delta compilation and publishing only on those application objects that have changed during development in Visual Studio Code. RAD publishing is an interim state and does not replace a full publish.

How RAD works

The files that have been changed by the application developer within Visual Studio Code are persisted in a special RAD (.rad) file during builds. This file is saved in the .vscode folder of the code project. RAD changes are the changes of application objects within a RAD session. Only application objects, page customization objects, and profile objects are handled for RAD. RAD changes will not be persisted during save, only during build, publish, and debug.

> **IMPORTANT**
>
> The `rad.json` file should not be modified.

> **IMPORTANT**
>
> If you change many files and close Visual Studio Code without a build (**Ctrl+Shift+B**), publish (**Ctrl+F5**, **Ctrl+Shift+F5**) or debug (**F5**, **Shift+F5**) all the RAD changes will be lost. This means that if you, in the next Visual Studio Code session perform a RAD publishing, this is done on the latest changes and not the prior changes. This can lead to an incomplete published package if it succeeds. It is therefore a best practice to do a regular publish. You can always check the RAD file in the code project to see what application objects are going to be changed during publishing.

In scenarios when application IDs are renamed, or refactored it is also a best practice to first do a full publishing, and then a RAD publishing for the consecutive changes. RAD does not check for application ID changes and ID changes can occur in a wrongly published application.

A RAD published file will not contain the following files that are normally packaged during regular publishing:

- Translation files
- Permission files
- Custom word and report rdl layout files
- Table data
- Web service definitions

These files will need to be re-generated with full publishing (**Ctrl+F5**). A RAD file will be deleted as a result of a successful publishing.

> **NOTE**
>
> If RAD publishing fails, then you must do a full publishing before performing another RAD publishing. The final state of an application must be built using full publishing, and never with RAD publishing.

RAD shortcuts

There are two commands for starting a RAD-based action.

SHORTCUT	DESCRIPTION
Ctrl+Alt+F5	Start RAD publishing without debugging.
Alt+F5	Start RAD with debugging.

See also

Developing Extensions in AL
Debugging

Signing an APP Package File

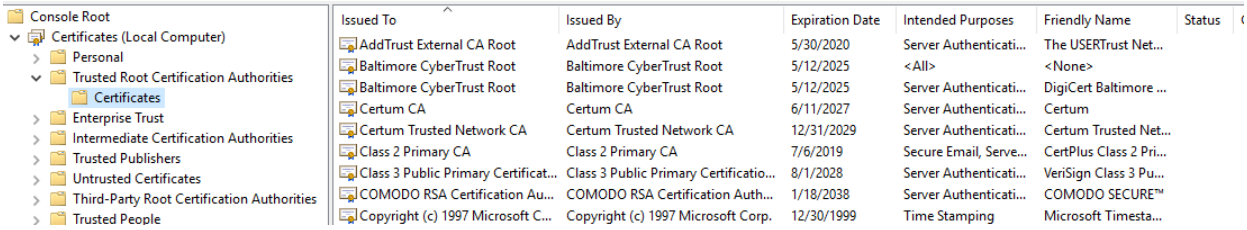
2/17/2021 • 3 minutes to read • Edit Online

Code signing is a common practice for many applications. It is the process of digitally signing a file to verify the author and that the file has not been tampered with since it was signed. The signature of the APP package file is verified during the publishing of the extension using the `Publish-NAVApp` cmdlet. For more technical information on signing, see Authenticode.

> **NOTE**
>
> If you want to publish an unsigned extension package in your on-premise environment, you need to explicitly state that by using the -*SkipVerification* parameter on the `Publish-NAVApp` cmdlet. An extension without a valid signature will not be published on AppSource.

The signing of an APP package file must be performed on a computer that has Dynamics 365 Business Central installed. If you are running Dynamics 365 Business Central on Docker for your development environment, that environment will meet this requirement. You must also have the certificate that will be used for signing on the computer. The certificate must include code signing as the intended purpose. It is recommended that you use a certificate purchased from a third-party certificate authority.

> **IMPORTANT**
>
> If you publish the extension as an app on AppSource, the APP package file must be signed using a certificate purchased from a Certification Authority that has its root certificates in Microsoft Windows. You can obtain a certificate from a range of certificate providers, including but not limited to GoDaddy, DigiCert, and Symantec, see the image below.



Steps for signing your .app file

1. Prepare your computer for signing.
2. Make sure that you sign the .app file on a computer that has Dynamics 365 Business Central installed.
3. Copy the certificate that you purchased from a third-party certificate authority to a folder on the computer. The example uses a pfx version of the certificate. If the certificate you purchased is not in a pfx format, create a PFX file. The file path for the sample command is `C:\Certificates\MyCert.pfx`. (Optionally, create your own certificate for local test or development purposes using the Self-signed certificate information).
4. Install a signing tool such as SignTool or SignCode to the computer. The sample command will use SignTool.
5. Copy your extensions .app file to the computer if it is not already on the computer. The file path for the sample command is `C:\NAV\Proseware.app`.
6. Run the command to sign the .app file.
7. The following example signs the Proseware.app file with a time stamp using the certificate in the password-protected MyCert.pfx file. The command is run on the computer that was prepared for the signing. Once the command has been run, the Proseware.app file has been modified with a signature. This file is then used

when publishing the extension.

```
SignTool sign /f C:\Certificates\MyCert.pfx /p MyPassword /t
http://timestamp.verisign.com/scripts/timestamp.dll “C:\NAV\Proseware.app”
```

IMPORTANT

It is recommended to use a time stamp when signing the APP package file. A time stamp allows the signature to be verifiable even after the certificate used for the signature has expired. For more information, see Time Stamping Authenticode Signatures. Depending on the certification authority, you may need to acquire a specific certificate in order to time stamp, an Extended Validation certificate from DigiCert for example.

NOTE

If you are using the BCContainerHelper PowerShell module to run Dynamics 365 Business Central on Docker, you can use the function `Sign-BCContainerApp` to perform all the steps above.

Self-signed certificate

For testing purposes and on-premise deployments, it is acceptable to create your own self-signed certificate using the New-SelfSignedCertificate cmdlet in PowerShell on Windows 10 or MakeCert.

The following example illustrates how to create a new self-signed certificate for code signing:

```
New-SelfSignedCertificate –Type CodeSigningCert –Subject “CN=ProsewareTest”
```

The following MakeCert command is used to create a new self-signed certificate for code signing:

```
Makecert –sk myNewKey –n “CN=Prosewaretest” –r –ss my
```

See Also

Getting Started with AL
Keyboard Shortcuts
AL Development Environment

Working with multiple AL project folders within one workspace

2/17/2021 • 2 minutes to read • Edit Online

Visual Studio Code offers the multi-root workspace feature which enables grouping different project folders into one workspace. The AL Language extension also supports the multi-root functionality and allows you to work with multiple AL folders including roots and projects within one workspace.

Working with multiple project folders

Go through the following steps to work simultaneously on several related projects.

1. On the **File** tab of Visual Studio Code, select **Add Folder to Workspace...** .
2. Save the workspace file if you plan to open it again.
   This will create a `code-workspace` file that contains an array of folders with either absolute or relative paths. If you want to share your workspace files, choose the relative paths.
3. Modify the settings of your files in the **Settings** editor. You can change your user settings, global workspace settings, or individual folder settings.

For more information about multi-root workspaces in Visual Studio Code, see Multi-root Workspaces.

Grouping a set of disparate project folders into one workspace

It is not mandatory to use only AL-based roots. Different kinds of projects can be mixed, and each AL project will have its configuration values for the following settings:

- `al.packageCachePath`
- `al.enableCodeAnalysis`

The `al.packageCachePath` setting allows you to specify the path to a folder that will act as the cache for the symbol files used by your project. It can be specified in the **User Settings**, **Workspace Settings**, or **Project Settings**. The `al.enableCodeAnalysis` setting allows you to enable the execution of code analyzers on your project. It can likewise be specified in the **User Settings**, **Workspace Settings**, or **Project Settings**. For more information, see AL Language Extension Configuration.

See also

Development in AL
Best Practices for AL
Working with Multiple Projects and Project References

Working with Multiple Projects and Project References

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

A project reference in an AL-based workspace is defined as a dependency in the `app.json` file and exists as a project in the workspace. There is no special visual representation of a project reference.

> **IMPORTANT**
>
> A *project reference* is the full `id`, `name`, `publisher`, and `version` of an existing project in the workspace. This is contrary to an application reference where it is enough to specify a minimal version.

In the example below, the project called **Leaf** defines two dependencies to the projects **Middle** and **Root**. Since both **Root** and **Middle** are projects in the workspace they are considered project references.

The advantage of working with project references is that there is no need to download the symbols for a project reference. They are there as the symbols for the reference project and will be resolved as they are modified. For example, if you add a new method to a codeunit in the **Root** project and reference the codeunit in the **Leaf** project, the method will automatically resolve as you touch the **Leaf** project.

When a project is built with **Ctrl+Shift+B** the following will happen:

1. The .app file is copied to the `.alpackages` folder of all projects that depend on it.
2. All project references that might be "dirty" are also built.

> **NOTE**
>
> If reference resolution stops working then building the project reference and re-initializing the workspace using **Reload Window** resolves references.

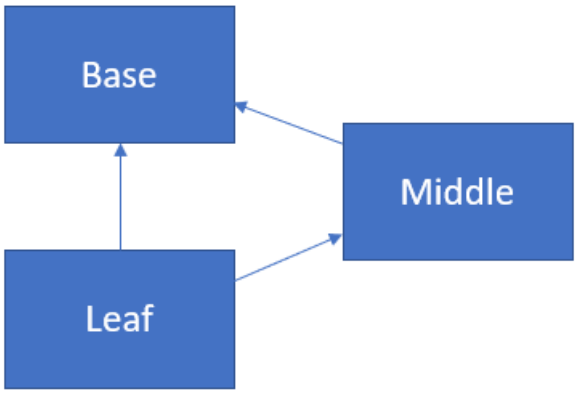
Publishing changes

With the introduction of project references the publishing logic in a workspace has changed. Publishing, either with **Ctrl+F5** or RAD publishing using **Alt+Ctrl+F5**, will do a set publishing of all the projects that have changed with defining a startup project. The startup project is always the active project. What does this mean?

A project is considered changed if any of its application objects have changed in the sense that the application object already is in the `rad.json` or will be in the `rad.json` once the project has been built. This means that if

you change an application object, you save it, and then close Visual Studio Code, but have not built, the `rad.json` will not be updated and then the project will not be considered "dirty".

For example, in a workspace with three projects; **Leaf**, **Middle**, and **Base**. **Leaf** depends on **Middle** and **Base**, and **Middle** depends on **Base** as illustrated below:



Assuming that:

1. All three projects; **Leaf**, **Base**, and **Middle** have changed.
2. The **Leaf** project is the current project that is published.

Then all three projects; **Base**, **Middle**, and **Leaf** will be part of the set that will be published.

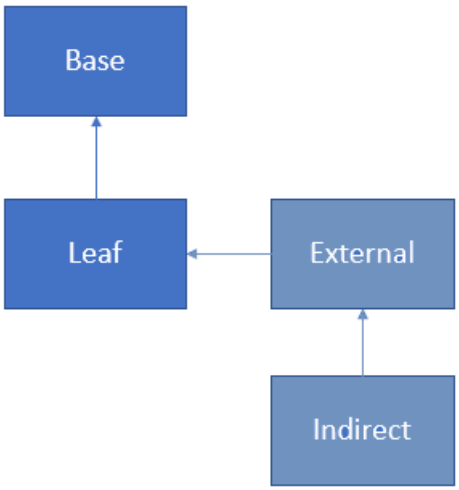
In a scenario where **Middle** has not changed, but **Leaf** is still the startup project, then only **Base** and **Leaf** will be published.

A new file is created to package set dependencies called `*.dep.app`. This is the file that gets transferred to the server and it is deleted if publishing of the dependency set is successful.

Server publishing changes

Although server publishing is an internal step, it does have an impact on the dependency publishing and is useful to know.

For example, in a workspace with two projects; **Leaf** depends on **Base**, and **External** and **Indirect** are projects outside of the workspace as illustrated below:



Assuming that:

1. A workspace exists with **Leaf** and **Base** as workspace projects.
2. **Base** is published.

3. On the server **Base**, **Leaf**, **External**, and **Indirect** are already installed apps.

The following happens on the server:

- All apps that depend on **Base** will be uninstalled, including **External** and **Indirect** dependency.
- Any other app that directly depend on **Base** and are not published in the global scope - in this case **Leaf** and **External** - are unpublished.
- **Base** will be uninstalled, unpublished, and then published.
- **Leaf** and **External** will be published, installed and then compiled against the newly published **Base**. Important to notice here is that the **Extenal** app also will be published.

Launch.json file setting

To control how dependency publishing is performed on the server, the `launch.json` file has a setting `dependencyPublishingOption` with the following options:

SETTING	DESCRIPTION
Default	Set dependency publishing will be applied.
Ignore	Dependency publishing is ignored. This setting should be used cautiously, see note below.
Strict	Dependency publishing will fail if there are any installed apps that depend on the startup project.

> **NOTE**
> With the `Ignore` setting only **Leaf** will be published against what has already been published on the server for **Middle** and **Base**. If a change has been done on **Base** that would break **Leaf**, even though local compilation would pass, the server compilation will fail in this scenario. The benefit of using this option is to gain publishing time when **Base** is a large project. Assuming that **Base** is published, then **Leaf** and **Middle** will be left untouched on the server. Only runtime errors will reveal if **Base** has broken **Middle** and **Leaf**.

Incremental Build setting

If the `al.incrementalBuild` setting is set to `true` on workspaces with project to project refernces, all resolution will happen from the referenced project, instead of happening from an app in the `\packagecache` folder which will enhance the build time. For more information, see AL Language Extension Configuration.

See also

Development in AL
Best Practices for AL
Working with multiple AL project folders within one workspace
JSON Files

The Lifecycle of Apps and Extensions for Business Central

2/17/2021 • 6 minutes to read • Edit Online

When you build an app or extension to Business Central and get that published to AppSource, it becomes an app like so many others - the app itself can be updated, and the platform that it sits on, Business Central online itself, will also get updated. But what happens after your app gets published?

When your app has passed all of our validations and has gone live to App Source, customers can install your extension and use it for their business. But you are expected to keep it compliant with the service and update it if something changes.

The following sections describe the different upgrade scenarios that we have seen play out as we update Business Central. For more information about your responsibility for keeping your app updated and the resources that are available to you, see Maintain AppSource Apps and Per-Tenant Extensions.

Scenario 1: Business Central service update

You don't need to make any bug fixes, feature adds, or app changes to your app. It continues to work fine without any interaction on your part.

Impact of service updates

The monthly service upgrades to Business Central do not impact your app. Your app just gets moved along and no upgrade code from your app needs to get used. Business Central itself gets upgraded on your tenant, and once complete, the customer sees no difference with your app.

Scenario 2: App update

You (our partner) add some features to your app and also some minor bug fixes. The app is submitted for validation. The app passes validation and gets checked into the service. This is now the active app for any new tenants and also for existing tenants that have never had your app installed before

Impact of app updates

Customers can either do an uninstall and then reinstall on their own, or they can ask their partner do it on their behalf from the Extension Management window within Business Central. Otherwise, they would have to wait until our every 6-month major release. That is the only time we do a force upgrade of extensions (except for critical bug hotfix extension updates)

Scenario 3: Reported bugs in your app

You (our partner) has various customers report some bugs that are impacting their usage of the app. The bugs aren't critical but they are important. The partner makes the fixes in the app and resubmits for validation. The app passes validation and gets checked into the service. This is now the active app for any new tenants and also for existing tenants that have never had your app installed before

Impact of bugs

We still do not force the upgrade of this app to this latest version on all of the tenants. Some tenants may not be using the functionality that includes this bug and continue to work fine on the current version of the app. Therefore, you should work directly with all of the impacted customer tenants to uninstall and reinstall to get the latest app version that contains fixes for the bug.

Scenario 4: Critical bug in your app

If a bug which leads to core functionality being broken or data loss/corruption/misrepresentation is found in the application, and the issue prevents customers from performing time-critical tasks, the validation and deployment of the application can be prioritized. The partner must create a support ticket for this case and they must immediately provide a fixed app for validation through Partner Center. The validation team makes this a top priority and does validation as soon as possible. If the fixed application passes validation, it will be checked into the service and will become available for environment administrators to install.

Scenario 5: Microsoft feature breaks your app

Microsoft has to break your app file for a needed Business Central core change. Some reasons for breaking could be security, bugs in the underlying code, high priority feature adds, and so on. Keep in mind, we do our very best to not break your app through our changes. We try and find proper ways of doing the changes without breaking your app. However, if we can't find a proper (non-breaking) way, then we could break your app. This won't be as likely in a minor update release (unless a security change is required on our part and that is the change that breaks you), but it can be more likely in our major (every 6-month) releases.

Impact of breaking changes

Here is our process when this takes place:

- First of all, Microsoft will not make a breaking change in the production environment at any point. Therefore, existing tenants are not expected to see this breaking change occur.
- When we make a breaking change, we do it in a build branch that is for a future release (monthly service minor or major release)
- We notify the partner in advance and give the partner ample time to fix their app, get it validated, and have it ready
- The fixed app will already be in our service and slotted as required for when your tenant is to be moved to the Business Central release that has the app breaking change
- As a result, the customer (tenant owner) should never see their Business Central break. Because the tenant gets moved from one monthly service update of Business Central to another, the tenant is being upgraded to the release of ours that breaks the specific app. However, our service detects that there is a new required version of that app (your fixed version). Therefore, we auto install the fixed version of the app for the tenant

Conclusions

You're responsible for your app. You own the process of updating the app and providing upgrade code if the schema changes between versions of the app.
If a customer uninstalls your app, and then installs it again later, then when they install the app the second time, they get the latest version from AppSource.

How Microsoft handles your app

When Microsoft upgrades a tenant with a service update, your app is tested against the new service version. If the app breaks, Microsoft rolls back to the previous healthy state. Your customer never learns that anything was about to break.

When a tenant uninstalls and reinstalls an extension via the Extension Management page or AppSource, there is platform logic that determines whether an *Install* or an *Upgrade* must take place. We detect which version of the extension the tenant previously had installed and perform the appropriate action. Therefore, the result of manually uninstalling/installing the extension is the exact same as an automated upgrade.

Additionally, there will not be any data loss during uninstall, install, or upgrade actions. Data for extensions is stored in its own tables in the tenant database. Before an extension gets installed, it first get synchronized on the tenant database. This step is implicit and happens automatically when a tenant installs an extension. This

synchronization process creates the database tables for the extension. Once the extension is installed and the tenant is using it, extension-specific data will get stored in these tables.

When an extensions gets uninstalled, these tables do not get removed. Therefore, when the extension gets reinstalled (or upgraded), the data is still available. You do not need to worry about data loss for choosing the uninstall/install route. However, do keep in mind that if any actions are being performed on the tenant while the extension is uninstalled, the extension's events and such will not be firing, and your app may miss the creation of new data. Try to perform the uninstall/install while the tenant is not online.

For more information, see When apps or PTEs cannot be updated by Microsoft.

See Also

Publishing and Installing an Extension
Retaining table data after publishing
Upgrading Extensions
Add your App to AppSource
Checklist for Submitting Your App
Upgrading AppSource Apps in Production
Maintain AppSource Apps and Per-Tenant Extensions

Update Lifecycle for Customizations of Business Central Online

2/17/2021 • 2 minutes to read • Edit Online

When you create a tenant-specific customization, or an extension that is scoped to a single Business Central environment (often referred to as per-tenant extensions), you must take the lifecycle of the extension into consideration. For more information about the extension lifecycle and events that can cause incompatibilities between the extension and base application, see The Lifecycle of Apps and Extensions for Business Central.

You are responsible for your extension. You own the process of updating the extension and providing upgrade code if the schema changes between versions of the Business Central base application. When an update is available for your Business Central environment, all extensions, both AppSource extensions and tenant customizations, must be compatible with the next version of the base application before the update can be installed on the environment. You are responsible for ensuring that your extensions are compatible with the update version.

In this article, we describe the process for ensuring update compatibility for tenant customizations.

Automated Extension Validation

An automated service validates all tenant customizations before an update is marked for a scheduled update of an environment. This validation compares the extension's dependencies with the updated version of the Business Central application to see if any changes caused incompatibilities with the new version.

If the validation service discovers any tenant customizations that are not compatible with the update, an email notification is sent to the tenant administrators listed on the **Notification recipients** tab of the Business Central administration center. For more information, see Managing Tenant Notifications.

> **IMPORTANT**
> At least one email address must be specified as a notification recipient to receive the update notifications. If you do not speciy an email address, you will not be notified of updates and other changes to the tenant.

The email notification provides information on the incompatible extension, detail on which properties must be updated, and steps to bring the extension into compatibility.

Update Failure Notification

If a Business Central environment has an extension that is not compatible with the update version, the update cannot be applied. If an update fails due to an incompatible extension, a notification is sent to the tenant administrators listed on the **Notification recipients** tab of the Business Central administration center.

The notification is similar to that provided by the automated extension validation. It provides information on the incompatible extension, detail on which properties must be updated, and steps to bring the extension into compatibility.

Automatic Extension Removal

The publisher of an extension must maintain compatibility with the new release of Business Central. An extension that is not compatible with the update within 90 days of the first notification of incompatibility will be

removed, and then the environment will be updated.

See Also

Retaining table data after publishing
Upgrading Extensions
Updating Environments

Maintain AppSource Apps and Per-Tenant Extensions in Business Central Online

2/17/2021 • 7 minutes to read • Edit Online

As a partner, keeping your apps and per-tenant extensions (PTEs) up to date is your responsibility. Business Central is regularly updated with major and minor releases. These updates provide customers with a business application that is always compliant, secure, and enriched with new platform and application functionality. Often customers choose Business Central because of this promise of having an always up-to-date business solution.

To not break this promise, developers that bring apps to Microsoft AppSource, and resellers that provide PTEs to respond to the unique needs of customers, have a responsibility to align their code to the Microsoft release rhythm.

An inability for Microsoft to update tenants because of publishers incompatible code causes serious disruption in the service and must be avoided since it impacts the trustworthiness of the service and customer satisfaction.

Resources

To help app publishers keep up with their update responsibilities, Microsoft provides the following resources:

- Release plans about what's new and planned

  For more information, see Dynamics 365 release plans.

- Access to pre-release bits

  Business Central partners have access to the next major, next minor, and daily pre-release bits in Docker. These bits can be used to test apps against upcoming updates.

- Information about what will be deprecated

  With all Business Central releases, Microsoft controls and regulates breaking changes with major releases and communicates upcoming breaking changes at least one year in advance. If developers missed this above info, the compiler in Visual Studio Code also warns for potential controls that will become obsolete in future versions and how to deal with them.

- Policy definitions and terms

  The publisher agreement and the commercial marketplace certification policies describe the responsibilities of app providers for how to publish and maintain apps in the Microsoft monthly rhythm.

  When a PTE gets installed, the publisher also agrees to the terms to keep that code current and updatable.

- Training and coaching

  Microsoft provides a set of tools, training, and documentation to help partners find the info they need to keep up with these responsibilities on continuous integration and continuous deployment. External providers, including ISV Development Centers, MasterVARs, and training centers, can provide in-person training and coaching.

- Service notifications

  Business Central online will support app and PTE publishers with extra warnings about potential technical incompatibility. If publishers respond to these notifications in due timing and avoid incompatibilities

repeatedly, Microsoft will stand with these publishers to help where needed. If a publisher includes a telemetry key in their app, then, starting with 2020 release wave 2, Business Central also provides publishers with telemetry about upgrade failures that happen in production because of issues in the publisher's upgrade code.

If publishers lack to keep their code updatable, they risk that ultimately their apps or PTEs will be removed from the customer's tenant, and this will most likely result in important data not being captured as it should. For apps, this also means removal from the marketplace.

Since resellers are the first line contact point for customers, they carry responsibility to explain what it means to load code in a customer's environment. The best way is to explain this is with terms.

We advise these terms include topics like intellectual property rights, upgrade responsibilities, associated costs to keep code updatable, support options, data privacy, and so on.

Pre-release publisher support to keep apps and PTEs compatible and up-to date

Publishers have several tools available for them to keep their code in good shape. Not least, a Public Preview release is made available approximately one month before the announced release date for a major release. In that Public Preview release time frame, Business Central will automatically test and notify publishers of existing apps and PTEs running in production on technical incompatibility with the upcoming release.

> **IMPORTANT**
> Microsoft tests code based on technical compatibility. As the publisher, you are still responsible for all functional and logical validation.

When apps or PTEs cannot be updated by Microsoft

This section describes the processes that are initiated during and after upgrade attempts of code provided by publishers of apps or PTEs.

- **T1 – T30**: Microsoft alerts administrators, resellers, and ISVs

  Shortly after a service update of Business Central online (Day **T**), Microsoft will initiate daily updates attempts on all tenants. In these update attempts, the publisher's provided upgrade code is triggered and run. These attempts run repeatedly in a time frame of approximately one month until the upgrade is successful. For more information, see Major Updates of Business Central Online.

  With every unsuccessful upgrade attempt, stakeholders will receive notifications. Customers and their reselling partners can follow these notifications in the Business Central administration center.

  ISVs who provide third-party AppSource apps might not be listed in the customer's admin center. The reseller will in most cases have worked with the ISV to test compatibility, but after two weeks (Day **T+15**) of failed upgrade attempts, the Microsoft AppSource team will also send the app provider a warning message that action within the next few days is required.

  This message will explain that if they fail to respond correctly, their app will be removed from AppSource at Day **T+30**.

- **T30 – T60**: Microsoft alerts the customer

  After one month of failed upgrade attempts (Day **T+30**), the customers will be notified again that apps or PTEs are incompatible with the new version of Business Central, and that no further automatic upgrade attempts will be planned until further notice. Although the publisher's code continues to run on an

outdated version of Business Central online, the customer must work with their reseller to resolve these issues immediately so that the tenant can be updated. Next to messages in the Business Central administration center, all users in the customer's tenant will also get more active warning about the incompatibilities when they use the product in the browser or their mobile device.

For AppSource apps, if no appropriate action or follow-up was taken by the publisher since the release (**T - T+30days**), the app will be removed from AppSource. This means no new customer will be able to install the app in a new tenant. The main reason for removing an app from the marketplace is to ensure that no new customers will be affected by incompatibilities with the latest version of the base product, Business Central.

If the publisher wants to have their app available again, they must mitigate all existing incompatibility issues and go through the full validation process again.

If the source of the incompatibility has been resolved by the publisher, they'll have to submit a support request to schedule a new set of upgrade attempts for any tenants that are blocked because of this incompatibility. They'll also have to work with their resellers to inform them about the compatibility resolution.

- **T60 – T150**: Microsoft initiates the customer wind-down period

  If the incompatibility issues are not resolved at **T+60**, and the publisher remained unresponsive to the request to resolve the incompatibility, Microsoft may choose to send out a wind-down communication to the customer about removal of the publisher's code.

  This communication will share that the code from the publisher will be removed in *90* days (**T+150**).

  During this wind-down time, the customer and their reselling partner are fully responsible for finding a solution on how to proceed in this situation. If the customer decides to leave Business Central, or decides to use another publisher, they can access their data by exporting the database, use RapidStart Services, or copy data to Excel. For more information, see Exporting Your Business Data to Excel in the business functionality content.

  Microsoft may also choose to remove all existing apps by this publisher from AppSource and block the publisher from publishing new apps for Business Central.

  If this wind-down period is initiated, and the customer was able to fix the incompatible issues with their reseller and potential publishers, it will be at Microsoft discretion if the publisher's code will be removed from Business Central or not.

Get notified about incompatibilities by Microsoft

It is crucial for you to keep contact details correctly up to date. We advise you to use global team aliases instead of individual mail addresses. Here are the mail addresses that we'll use in the above process:

- PTE publishers

  The mail addresses specified in the Business Central administration center; this could be both a customer user and a partner user.

- App publishers

  The mail addresses specified during App publication in the partner center during app publication as support and engineering contact details.

- Customers

  The mail addresses specified in the Business Central administration center; this could be both a customer and a partner user.

See also

The Lifecycle of Apps and Extensions
Update Lifecycle for Customizations
Microsoft Responsibilities for Apps on Business Central online
Technical Support for Business Central online
Sending Extension Telemetry to Azure Application Insights

Update Lifecycle for AppSource Apps FAQ

2/17/2021 • 5 minutes to read • Edit Online

Please see the following sections for frequently asked questions regarding updating apps on AppSource.

I want to submit an updated version of my app. What is the process for that?

For any updated version of your app (small or large changes), you follow the same submission as your original version. It must go through the same validation process. The following are the steps that you must follow:

- Increase the version number of your app within it's json/manifest file.
- Do not change the app's AppID within it's json/manifest file. That needs to remain the same for life of the app
- Include an upgrade codeunit and ensure that it works.
- Upload the app file to your existing Partner Center offer.
- In Partner Center, edit the version field to match what is now in your updated app file.
- Make any other edits in Partner Center as needed.
- Submit for validation.
- Validation takes place as normal (against current production version of Business Central at time of submission):
  - Code review is needed for avoiding violations and ensuring requirements are met.
  - Test validation is scaled back for updated versions.

What happens when my updated app passes validation?

The app is checked into our service and becomes the active version for that current production release of Business Central. If a tenant is already upgraded to the current release, they can install this updated version of your app.

Does Microsoft now update my app (to this latest updated version) on all tenants that already have a previous version?

Microsoft will only force update apps during the 2 major releases (release wave 1 and 2). For these releases, each tenant will have their AppSource apps force updated to the latest available versions in our service.

How do I update the apps on my tenant for minor releases then?

If a tenant wants to update the apps on their tenant during the minor releases, the tenant admin needs to handle this. Here are the steps you follow to update your apps:

- Login to your Business Central Web client instance.
- Navigate to the **Extension Management** page.
- Find the app and uninstall it.
- Reinstall the app.

That gives you the latest available version in our service.

How often should I submit updates of my app?

Our recommendation is to pack more bug fixes and features into less frequent updates. Try to avoid frequent submissions containing very few changes. Being on a more frequent cadence than Business Central (monthly) is not advised. This leads to lower churn to production tenants.

What if a customer reports a critical bug in my app and needs an immediate hotfix version of my app?

We do have a fast track validation process for situations like this. These types of situations become top priority in our queue. A very quick validation takes place with the goal of having the hotfix in our service that same day. The following is the hotfix process:

- Fully test the hotfix version of your app to ensure it fixes the problem and to make sure no other issues are introduced.
- Submit the app via Partner Center per the normal process.
- Email rweigel@microsoft.com to notify him of the hotfix situation.
- Provide justification as to why this is critical. Some definitions of critical being:
  - App is causing the tenant to be unresponsive or unusable with no workaround
  - App is serving a critical business process and that process cannot be executed without a fix
- If justification proves to be critical, hotfix process can proceed.
- Quick Validation takes place ASAP and app is uploaded to our service.
- Tenants can now install this version of your app.

> **NOTE**
> Please ensure that you are only using this process for critical situations. Do not try to use it for minor bug fixes.

Do you have any tips for us when submitting updates of our app?

Yes, we have some valuable tips we would like to share. These are tips that can save you time in the validation process. They will help lead to fewer (and possibly zero) failures during validation. Most importantly; they will lead to fewer issues being found in production by customers.

- Follow the checklist, for more information see Technical Validation Checklist. The checklist is ever evolving and requirements might change or be added. You might miss something from the checklist, leading to validation failure and delaying the passing of your updated app.
- Use AppSourceCop, for more information see Using the Code Analysis Tool. This helps to catch any missing prefix/suffix and DataClassification. Too often we see these fail the updated versions of apps.
- Sign your app. This fails many app validations. We try to publish the app during validation and it is not properly code-signed leading to failure to publish.
- Publish and install your app. This is another big validation failure we see too often.
- Test your app's functionality with 100% coverage. You are the expert on the app and know it best. If you are only testing a small percentage of your app, customers will most likely find issues resulting in you having to update your app more often. And if customers are the ones finding your app issues, they may decide to uninstall it. You should have a vested interest in providing a quality app.
- Test the upgrade of your app. upgrade from the previous version to this latest. Your updated app will not pass validation until the upgrade works. If it fails in our validation, we will return it to you, leading to a delay in it going to our service.

NOTE

The tips that we provide above are for your benefit to pass validation the first time through each submission. And to emphasize these points, think of the delays that can arise when you do not follow these tips. If you submit, it can take a couple days before we validate the app. Once we validate it, if all is good, it should pass quickly. If we find issues that lead us to fail the validation, we send the app back to you as you have to make fixes. It could then take you days to properly fix. Next time you submit, your app does not go to the front of the queue. It begins at the bottom again which means it could be another couple days before we validate it again. By following our tips above, you can avoid those delays. Spend a bit more time up front finding those issues yourself, leading to a quicker path to our service.

See Also

Retaining table data after publishing
Checklist for Submitting Your App
Upgrading Extensions
Using the Code Analysis Tool

FAQ about Managing and Submitting your Business Central Offer

2/17/2021 • 2 minutes to read • Edit Online

This section contains answers to frequently asked questions about managing the offer in Partner Center when you submit an app for Business Central.

If I click the Go Live button, does that mean my app will go live to AppSource?

No. This button is deceiving, and I wish it were worded differently. When you click this button, it triggers our validation process and puts your app into our validation queue.

What should I know about the Review and publish button?

This is the button you should click when you want to submit your app for validation each time. Once you click the button, it takes you to a Review and publish page. Make sure all checkboxes are marked on that page. This pulls in all changed data into our validation queuing system properly.

I only made marketing changes. Will my app have to go through technical validation?

No. If you do not change your app in a submission, we skip technical validation.

See also

FAQ about Updating your Business Central App
FAQ about Library & Dependency Apps in Business Central
Update Lifecycle for AppSource Apps FAQ
The Lifecycle of Apps and Extensions for Business Central

FAQ about Library and Dependency Apps in Business Central

2/17/2021 • 2 minutes to read • Edit Online

This section contains answers to frequently asked questions about library apps and dependency apps in Business Central.

What is a library app?

A library app contains common code that other apps depend on. If you are going to have multiple AppSource apps that all share common code, such as licensing, registration,and so on, you can put that common code into a library app and have the AppSource apps depend on the library app.

How does the library app get installed to a tenant?

A library app does not appear on AppSource. It only lives in Business Central. Our service is built to install the library apps behind the scenes. Here is how it works. Customers finds an app they want to install on AppSource. That customer does not know the app even has library app(s). When they click to install the app, our service looks into that apps json/manifest to see if it first needs to install any library/dependency apps it may depend on. If so, it installs them first before installing the main AppSource app.

What is a dependency app?

A dependency app isn't much different from a library app. A dependency app does have its own offer on AppSource. For example, you might have AppSource "App A" that serves a purpose on its own and is listed on AppSource. You then also have "App B" as its own offer on AppSource but it does depend on code within "App A" and needs "App A" to be installed before it can be installed. This is really the only difference though. Behind the scenes, library and dependency apps behave the same when it comes to walking that dependency chain and auto installing any app the top-level app depends on.

Can I have multiple library apps for my AppSource app?

Yes. Just make sure you list all library apps as dependencies within the AppSource app json file/manifest.

When I get the latest updated version of my app, why don't I get the updated library/dependency apps that my AppSource app depends on?

Currently, updating of apps does not handle the library/dependency chain. This work is planned for a future release. For now, you need to uninstall and reinstall the library/dependency apps as well.

What are the validation requirements for library apps

Library apps only get validated technically, but don't go through any type of marketing validation. Also, the library apps are expected to be covered by the user scenarios and tests that you submit for the core app. The technical requirements are described here.

See also

FAQ about Managing and Submitting your Business Central Offer
FAQ about Updating your Business Central App
Update Lifecycle for AppSource Apps FAQ
The Lifecycle of Apps and Extensions for Business Central

FAQ about Testing your Business Central App

2/17/2021 • 4 minutes to read • Edit Online

This section contains answers to frequently asked questions about testing your app when you submit an app for Business Central.

Since my app goes through validation, do I really need to thoroughly test my app?

Yes. It is your app and you are the expert, so you should have a vested interest in high quality. The main goal of validation is to ensure app code is meeting requirements and policy. The testing done during validation is only to ensure good user experience on the most common app scenarios. We do not test every scenario of an app. Using customers to find bugs is not the proper approach.

Is it ok to submit my app for validation before we complete our own testing?

No. Please do not submit your app for validation until it has been 100% tested. This will lead to many delays in the validation process otherwise. Also, it wastes time and resources.

Is it ok to test my AppSource app in an on-premises environment?

No. You must test your app using Business Central online. Although Business Central online and on-premises are very similar, they are not the exact same. If you only test on-premises, invariably issues will be found in our validation testing.

Does it really matter what version of Business Central I test on before submitting my app for validation?

Yes. It is critical that you always test on the latest version at the time when you are ready to submit for validation. Testing on the wrong version usually leads to validation failure. For example, let's say the latest version of Business Central is 15.4 at the time when you submit for validation You tested on 15.0. The product has changed between those versions, and deprecated features or other changes could result in your app behavior changing.

We recommend that you consider using Docker containers. Use the `Get-BcArtifactUrl` function in the BCContainerHelper PowerShell module to give you the artifact URL for the latest sandbox build for the specified country. You don't have to figure out what product version is active at time of submission. That function does it for you.

How thorough should our testing be?

You should always test with 100% coverage, or at least as close to 100% as you can get. The testing should be a combination of automated and manual tests. The apps with good testing infrastructure behind them are the most successful.

Do I have to test on every country that I intend to support with my app?

Yes. If you support multiple countries, test your app on every country. Each country's code base is slightly

different from both the base application and other countries. It is critical to make sure that the app publishes, syncs, and installs on every country you support. Because an app installs fine on one country doesn't mean it will publish/install fine on another. We have seen many times where it passes on one but fails on another during our validation.

Do you have recommendations on maintenance testing of our apps?

Yes. You should be testing your apps against our various build branches. Through Docker, you have access to our latest public sandbox builds and through the "Ready! for Dynamics 365 Business Central" program on Microsoft Collaborate you can also get access to *Next Minor* and *Next Major* builds. Test often, especially against the *Next Minor* build. This allows you to catch any bugs that may arise from core changes in the product.

I only made minor code changes in my updated app. Can I test just these changes?

No. You should always test 100% coverage no matter what. Testing only what you changed is not the correct approach. Even minor changes can lead to breaking changes where you least expect it.

Do I need to do upgrade testing?

Yes, this is a must. If an app fails to upgrade, customers are unable to get your updated app. This is one of the common failures we see today in validations. You should want your app upgrade to work optimally.

Any recommendations on upgrade testing?

Test the upgrade with extensive app data included. Many of the upgrade failures for customers are data related. The upgrade fails because specific data scenarios were not considered for testing. Or worse, the upgrade succeeds, and data is lost.

Do I only need to do upgrade testing from previous version to current?

No. It is very important you test from various previous versions of your app. This is because we do not automatically upgrade apps for minor releases. You could have a tenant back on version 1.0.0.0 of your app and have to jump all the way to version 1.0.0.5. We don't guarantee direct upgrades of apps from their most previous version.

See also

FAQ about Updating your Business Central App
FAQ about Library & Dependency Apps in Business Central
Update Lifecycle for AppSource Apps FAQ
The Lifecycle of Apps and Extensions for Business Central

FAQ about Updating your Business Central App

2/17/2021 • 5 minutes to read • Edit Online

This section contains answers to frequently asked questions about updating your app for Business Central.

Is it the same process for an app update as the first version?

Yes. You upload the updated app file into Partner Center and submit as normal. It goes through a more scaled-back validation process than your original version, but it does get validated. If it fails validation, it comes back to you for fixing. If it passes validation, it gets checked into Business Central.

What are some "need to know" considerations with an updated app?

When you submit an updated version of your app, you must increase the version number in the app's json/manifest files. Business Central doesn't allow overwrites. So we need the version number increased for us to check the updated app into our service upon it passing validation.

Never change the app's App ID in the json/manifest files. This is must stay the same across versions for various reasons, not least for upgrade reasons.

When is my updated app available for tenants to install?

As soon as your updated app passes validation and is checked into our service, it then becomes the active version (for whatever the current Business Central version is at that time). Even though your offer might still show as in progress in Partner Center, the updated app is active and ready to install. Also, even though your version number in AppSource might show as older, tenants will still get this latest updated version. For example, your updated version might be version 1.0.0.5. And in AppSource it might still show 1.0.0.1. Tenants will get the version 1.0.0.5.

What version of Business Central is my updated app compatible with?

When you submit your app for validation, we validate it against the latest version at that point in time. Once the app passes validation, it is then checked into our service and configured for that version of Business Central, unless a newer version has been released in the meanwhile. For example, if you submit your app and the latest version of Business Central at that time is 16.1, your app will then be compatible and configured for 16.1. When versions 16.2, 16.3, and so on roll out, your app is automatically configured for the latest version. Tenants that are on earlier versions of Business Central will not get that updated version. They still however can install the previous versions of your app. They will get the version of your app that is configured for the version of Business Central that their tenant is on.

When tenants have their Business Central version upgraded, do apps ever get automatically updated?

For minor releases, we do not auto-update apps. This is because of customer feedback and them not wanting their apps auto updated. The only exception to this is if an app will be broken in a minor release due to changes in the base product. In this case, we configure that app in our service as required. When tenants then get upgraded to that minor release, if they have that app, it then gets updated automatically. This is to avoid the app being broken for the customer.

For major releases, we do auto-update every app on every tenant to the app's latest available version. We

consider major releases to be our "refresh" releases.

How do tenants that already have an existing version of my app get my latest updated version?

By uninstalling and reinstalling the app. When doing the reinstall, it calls into our service and finds the latest active version of an app. There is a new **Manage Apps** page in the Business Central administration center that will soon allow updating of AppSource apps without having to do uninstall and reinstall.

Do I have to submit an updated version of my app for the major releases?

No. The only reason you would need to submit an updated version of your app for a major release is if your app is going to be broken in that release. If your app will not be broken, then your latest available version of the app will be rolled over to that major release.

How frequent should I submit updated versions of my app?

We recommend that you bundle more bug fixes and features so that your app doesn't have to be updated frequently. This has been voiced by our Business Central customers. They do not want to be constantly updating their apps in their tenants. We recommend a minimum 1-month app update cadence.

What if I have a critical hotfix?

We treat critical hotfixes with the utmost importance. We do have a process around this. Additional information on this hotfix process can be found here.

If I make changes to the library app, must I also submit an update for the AppSource app?

You would upload the updated library app to Partner Center, leave the main app as is, and submit for validation. We then see that the main app has not changed and only validate the library app.

But wouldn’t I need to change the dependency in my main app’s json to reference the updated library app file?

No. The version number in the dependency listing in the json file to an app is a minimum version. The main app is essentially saying, "I need version 1.0.0.1 or greater" of the library app. For example, the AppSource app lists version 1.0.0.1 for the library app, and that means that it can also use version 1.0.0.2.

Why don't I see the updated version of my app in my sandbox tenant?

Your tenant is on an older version of Business Central and has not yet been upgraded to the latest version of Business Central. The latest updated version of your app is only compatible with the latest Business Central version and later. If you upgrade your tenant to the latest version, you can then update your app.

See also

FAQ about Managing and Submitting your Business Central Offer
FAQ about Library & Dependency Apps in Business Central
Update Lifecycle for AppSource Apps FAQ

Writing Extension Install Code

2/17/2021 • 2 minutes to read • Edit Online

There might be certain operations outside of the extension code itself that you want run when an extension is installed. These operations could include, for example, populating empty records with data, service callbacks and telemetry, version checks, and messages to users. To do these types of operations, you write extension install code. Extension install code is run when:

- An extension is installed for the first time.
- An uninstalled version is installed again.

This enables you to write different code for initial installation and reinstallation.

How to write install code

You write install logic in an *install* codeunit. This is a codeunit that has the SubType property set to **Install**. An install codeunit supports two system triggers on which you can add the install code.

TRIGGER	DESCRIPTION
OnInstallAppPerCompany()	Includes code for company-related operations. Runs once for each company in the database.
OnInstallAppPerDatabase()	Includes code for database-related operations. Runs once in the entire install process.

The install codeunit becomes an integral part of the extension version. You can have more than one install codeunit. However, be aware that there is no guarantee on the order of execution of the different codeunits. If you do use multiple install units, make sure that they can run independently of each other.

Install codeunit syntax

The following code illustrates the basic syntax and structure of an install codeunit:

```
codeunit [ID] [NAME]
{
    Subtype=Install;

    trigger OnInstallAppPerCompany()
    begin
        // Code for company related operations
    end;

    trigger OnInstallAppPerDatabase()
    begin
        // Code for database related operations
    end;

}
```

TIP

Use the shortcuts `tcodeunit` and `ttrigger` to create the basic structure for the codeunit and trigger.

Get information about an extension

Each extension version has a set of properties that contain information about the extension, including: AppVersion, DataVersion, Dependencies, Id, Name, and Publisher. This information can be useful when installing. For example, one of the more important properties is the `DataVersion` property, which tells you what version of data you are dealing with. These properties are encapsulated in a `ModuleInfo` data type. You can access these properties by through the `NAVApp.GetCurrentModuleInfo()` and `NAVAPP.GetModuleInfo()` methods. For more information, see NavApp Data Type

Install codeunit example

This example uses the `OnInstallAppPerDatabase()` trigger to check whether the data version of the previous extension version is compatible for the upgrade.

```
codeunit 50100 MyInstallCodeunit
{
    Subtype=Install;

    trigger OnInstallAppPerDatabase();
    var
        myAppInfo : ModuleInfo;
    begin
        // Get info about the currently executing module
        NavApp.GetCurrentModuleInfo(myAppInfo);

        // A 'DataVersion' of 0.0.0.0 indicates a 'fresh/new' install
        if myAppInfo.DataVersion = Version.Create(0,0,0,0) then
            HandleFreshInstall
        else
            // If not a fresh install, then we are Re-installing the same version of the extension
            HandleReinstall;
    end;

    local procedure HandleFreshInstall();
    begin
        // Do work needed the first time this extension is ever installed for this tenant.
        // Some possible usages:
        // - Service callback/telemetry indicating that extension was installed
        // - Initial data setup for use
    end;

    local procedure HandleReinstall();
    begin
        // Do work needed when reinstalling the same version of this extension back on this tenant.
        // Some possible usages:
        // - Service callback/telemetry indicating that extension was reinstalled
        // - Data 'patchup' work, for example, detecting if new 'base' records have been
        //   changed while you have been working 'offline'.
        // - Setup 'welcome back' messaging for next user access.
    end;
}
```

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension
Converting Extensions V1 to Extensions V2
Building Your First Sample Extension With Extension Objects, Install Code, and Upgrade Code

Upgrading Extensions

2/17/2021 • 12 minutes to read • Edit Online

This article provides information about how to make a newer version of extension upgrade available on tenants. The first phase of the process is to develop the extension for upgrading. In this phase, you add code to upgrade data from the previous extension version. Once you have the upgrade code in place, you can publish and synchronize the new version, and the run the data upgrade.

> **NOTE**
>
> An *upgrade* is defined as enabling an extension that has a greater version number, as defined in the app.json file, than the current installed extension version.

Writing upgrade code

When you develop a new extension version, you must consider the data from the previous version. You have to determine the modifications that must be made to the data to make it compatible with the current version. For example, maybe the new version adds a new field that needs default values set for existing records. Or, the new version adds new tables that must be linked to existing records. To address this type of data handling, you must write upgrade code for the extension version. If there are no data changes between the extension versions, you don't need to write upgrade code. All data that isn't modified by upgrade code will automatically be available when the process completes.

You write upgrade logic in an upgrade codeunit, which is a codeunit whose SubType property is set to **Upgrade**. An upgrade codeunit supports several system triggers on which you can add data upgrade code. These triggers are invoked when you run the data upgrade process on the new extension.

The upgrade codeunit becomes an integral part of the extension and may be modified as needed for later versions. You can have more than one upgrade codeunit. There's a set order to the sequence of the upgrade triggers, but the execution order of the different codeunits isn't guaranteed. If you do use multiple upgrade units, make sure that they can run independently of each other.

Upgrade triggers

The following table describes the upgrade triggers and lists them in the order in which they're invoked.

TRIGGER	DESCRIPTION	FAILS THE UPGRADE ON ERROR
OnCheckPreconditionsPerCompany() and OnCheckPreconditionsPerDatabase()	Used to check that certain requirements are met before the upgrade can be run.	Yes
OnUpgradePerCompany() and OnUpgradePerDatabase()	Used to do the actual upgrade.	Yes
OnValidateUpgradePerCompany() and OnValidateUpgradePerDatabase()	Used to check that the upgrade was successful.	Yes

`PerCompany` triggers are run once for each company in the database, where each trigger is executed within its own system session for the company.

`PerDatabase` triggers are run once in the entire upgrade process, in a single system session that doesn't open

any company.

> **NOTE**
> These triggers are also available in upgrade codeunits for the base application, not just for extensions.

Upgrade codeunit syntax

The following code illustrates the basic syntax and structure of an upgrade codeunit:

```
codeunit [ID] [NAME]
{
    Subtype=Upgrade;

    trigger OnCheckPreconditionsPerCompany()
    begin
        // Code to make sure company is OK to upgrade.
    end;

    trigger OnUpgradePerCompany()
    begin
        // Code to perform company related table upgrade tasks
    end;

    trigger OnValidateUpgradePerCompany()
    begin
        // Code to make sure that upgrade was successful for each company
    end;
}
```

> **TIP**
> Use the shortcuts `tcodeunit` and `ttrigger` to create the basic structure for the codeunit and trigger.

Controlling when upgrade code runs

In most cases, it's important, that upgrade code isn't run more than once. There are a couple ways that you can control when upgrade code runs. You can either use extension version data or upgrade tags. These two methods are described in the sections follow. The method you choose will depend on the complexity of your solution. Use the following table as a guideline:

SCENARIO	VERSION NUMBER	UPGRADE TAGS
Large application with many versions		x
Extension version changes frequently, for example more than once a year		x
Version is set manually	x	
When checking whether it's a first-time installation. In this case, comparing with 0.0.0.0	x	
Fixing a broken upgrade		x

Using extension version data to control upgrade code

Each extension version has a set of properties that contain information about the extension, including: `AppVersion`, `DataVersion`, `Dependencies`, `Id`, `Name`, and `Publisher`. This information can be useful when upgrading. For more information, see JSON Files.

The `AppVersion` is one of the available properties and its value differs depending on the context of the code being run:

- Normal operation: `AppVersion` represents the value of the currently installed extension.
- Installation code: `AppVersion` represents the version of the extension you're trying to install.
- Upgrade code: `AppVersion` represents the version of the extension that you're upgrading to (in other words, the 'newer' version).

Another one of the more important properties is the `DataVersion` property, that represents the value of most recently installed/uninstalled/upgraded version of the extension, meaning that it reflects the most recent version of the data on the system, be that from the currently installed, or a previously uninstalled extension. The `DataVersion` property value differs depending on the context of the code being run:

- Normal operation: `DataVersion` represents the version of the currently installed extension, in which case it's identical to the `AppVersion` property.
- Installation code:
  - Reinstallation (applying the same version): `DataVersion` represents the version of the extension you're trying to install (identical to the `AppVersion` property).
  - New installation: `DataVersion` represents the value of '0.0.0.0' that's used to indicate there's no data.
- Upgrade code:
  - The version of the extension you're upgrading from. Either what was last uninstalled, or what is currently installed.

All these properties are encapsulated in a `ModuleInfo` data type. You can access these properties through the `NAVApp.GetCurrentModuleInfo()` and `NAVApp.GetModuleInfo()` methods. For more information, see NavApp Data Type.

Upgrade codeunit example

This example uses the `OnCheckPreconditionsPerDatabase()` trigger to check whether the data version of the previous extension version is compatible for the upgrade before restoring the archived data of the old extension.

```
codeunit 50100 MyUpgradeCodeunit
{
    Subtype=Upgrade;

    trigger OnCheckPreconditionsPerDatabase();
    var
        myInfo : ModuleInfo;
    begin
        if NavApp.GetCurrentModuleInfo(myInfo) then
            if myInfo.DataVersion = Version.Create(1, 0, 0, 1) then
                error('The upgrade isn't compatible');
    end;

    trigger OnUpgradePerDatabase()
    begin
        NavApp.RestoreArchiveData(Database::"TableName");
    end;
}
```

Using upgrade tags to control upgrade code

Although you can control upgrade code by checking versions, this pattern becomes difficult with larger applications that have many versions. The risk of something going wrong increases. If your solution includes the Microsoft system application, another way is to use *upgrade tags*. Upgrade tags provide a more robust and resilient way of controlling the upgrade process. They provide a way to track upgrade methods have been run to prevent executing the same upgrade code twice. Tags can also be used to skip the upgrade methods for a specific company or to fix an upgrade that went wrong.

Implementation behind upgrade tags

Upgrade tags are implemented as part the **Upgrade Tags** module of the system application. The module provides an API that you can code against to control your upgrade code.

> **TIP**
> If you want to implement your own upgrade tagging functionality, or get a better understanding of the code behind implementation in the system application, see Upgrade Tags in the ALAppExtensions repository on GitHub.

The **Upgrade Tags** module consists of several AL objects. In particular, it includes codeunit 9999 "Upgrade Tag" for creating and handling upgrade tags and table 9999 "Upgrade Tags" for storing them.

Codeunit 9999 "Upgrade Tag" includes the following methods:

METHOD	DESCRIPTION
`HasUpgradeTag(Tag: Code[250]; TagCompanyName: Code[30]): Boolean`	Verifies whether a specific upgrade tag exists. `Tag` is the code to check. `TagCompanyName` is the name of the company for which to check the existence of the tag. Returns `true` if the `Tag` with the given code exists.
`SetUpgradeTag(NewTag: Code[250])`	Specifies the tag to save.
`SetAllUpgradeTags()`	Sets all upgrade tags in a new company. This method is called from codeunit 2 "Company Initialize" in the base application.
`SetAllUpgradeTags(NewCompanyName: Code[30])`	Sets all upgrade tags in a new company. This method is called from report 357 "Copy Company" in the base application.

The codeunit also publishes the following events:

EVENT	DESCRIPTION
`OnGetPerCompanyUpgradeTags(var PerCompanyUpgradeTags: List of [Code[250]])`	Subscribe to this event to register an upgrade tag for OnUpgradePerCompany upgrade method for a new company. `PerCompanyUpgradeTags` specifies a list of tags to insert. Tags that already exist won't be inserted.
`OnGetPerDatabaseUpgradeTags(var PerDatabaseUpgradeTags: List of [Code[250]])`	Subscribe to this event to register an upgrade tag for OnUpgradePerDatabase upgrade method for a new company. `PerDatabaseUpgradeTags` specifies a list of tags to insert. Tags that already exist won't be inserted.

How to use upgrade tags

The following steps provide the general pattern for using an upgrade tag on upgrade code.

> **IMPORTANT**
> Use upgrade tags only for upgrade purposes only.

1. Use the following construct around the upgrade code to check for and add an upgrade tag.

```
// Check whether upgrade tag exists
if UpgradeTag.HasUpgradeTag(UpgradeTagValue) then
  exit;

// Upgrade code

// Add the new upgrade tag using SetUpgradeTag or SetAllUpgradeTags
UpgradeTag.SetUpgradeTag(UpgradeTagValue);
```

   You can use any value for the upgrade tag, but we recommend that you use the convention [CompanyPrefix]-[ID]-[Description]-[YYYYMMDD], for example, ABC-1234-MyExtensionUpgrade-20201206.

2. Add code to register the upgrade tag for new companies that might eventually be created.

   This step ensures that the upgrade code isn't run on the first upgrade because of a missing tag.

   To register the tag, subscribe to the `OnGetPerCompanyUpgradeTags` or `OnGetPerDatabaseUpgradeTags` events.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Upgrade Tag", 'OnGetPerCompanyUpgradeTags', '',
false, false)]
local procedure OnGetPerCompanyTags(var PerCompanyUpgradeTags: List of [Code[250]]);
begin
    PerCompanyUpgradeTags.Add(UpgradeTagValue);
end;
```

   This step isn't necessary if the tag is added by calling the `SetAllUpgradeTags` method.

3. Add code to register the upgrade tag for first-time installations of the extension.

   The step ensures that the upgrade code isn't run on first upgrade because of missing tag.

   To register the tag, call the `SetUpgradeTag` method on the `OnInstallAppPerCompany` and `OnInstallAppPerDatabase` triggers in the extension's install codeunit.

```
codeunit 50100 InstallCodeunit
{
    Subtype=Install;

    trigger OnInstallAppPerCompany()
    var
        UpgradeTag: Codeunit "Upgrade Tag";
    begin
        if not UpgradeTag.HasUpgradeTag(UpgradeTagValue) then
            UpgradeTag.SetUpgradeTag(UpgradeTagValue);
    end;
}
```

> **NOTE**
>
> Instead of registering tags individually, call the `SetAllUpgradeTags` method as opposed to `SetUpgradeTag`. This will register automatically register new tags as they're added.

For more information about extension install code, see Writing Extension Install Code.

Design considerations

1. Keep tags simple by limiting nesting tags to two levels. Complicated if statements can lead to problems.
2. Implement additional safety checks to avoid data corruption, even though you're using upgrade tags. For example, when copying obsolete fields to new fields, make sure the new fields have a default or blank. This check adds safety if upgrade tags introduce data issues.

Example

The following code is a simple example of an upgrade codeunit. For this example, the original extension extended the **Customer** table with a **Shoesize** field. In the new version of the extension, the **Shoesize** field has been removed ObsoleteState=removed), and replaced by a new field **ABC - Customer Shoesize**. The upgrade code will copy data from **Shoesize** field to the **ABC - Customer Shoesize**. An upgrade tag ensures that code doesn't run more than once, and data isn't overwritten on future upgrades. The example also uses a separate codeunit to define the upgrade tag so that they aren't hard-coded, but within methods.

```
codeunit 50100 "ABC Upgrade Shoe Size"
{
    Subtype = Upgrade;

    trigger OnUpgradePerCompany()
    var
        ABCUpgradeTagDefinitions: Codeunit "ABC Upgrade Tag Definitions";
        UpgradeTagMgt: Codeunit "Upgrade Tag";
    begin

        // Check whether the tag has been used before, and if so, don't run upgrade code
        if UpgradeTagMgt.HasUpgradeTag(ABCUpgradeTagDefinitions.GetABCShoeSizeUpgradeTag()) then
            exit;

        // Run upgrade code
        UpgradeShoeSize();

        // Insert the upgrade tag in table 9999 "Upgrade Tags" for future reference
        UpgradeTagMgt.SetUpgradeTag(ABCUpgradeTagDefinitions.GetABCShoeSizeUpgradeTag());
    end;

    local procedure UpgradeShoeSize()
    var
        Customer: Record Customer;
    begin

        if not Customer.FindSet() then
            exit;

        repeat
            // Make sure that target field is blank because you're copying obsolete=removed field to new
field
            // Additional safety check
            if Customer."ABC - Customer Shoesize" <> 0 then
                Error('ShoeSize must be blank, the value is already assigned');

            // Avoid blank modifies - it is a performance hit and slows down the upgrade
            if Customer."ABC - Customer Shoesize" <> Customer.Shoesize then begin
                Customer."ABC - Customer Shoesize" := Customer.Shoesize;
                Customer.Modify();
            end;
        until Customer.Next() = 0;
    end;
}

codeunit 50101 "ABC Upgrade Tag Definitions"
{
    // Register the new upgrade tag for new companies when they are created.
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Upgrade Tag", 'OnGetPerCompanyUpgradeTags', '', false,
false)]
    local procedure OnGetPerCompanyTags(var PerCompanyUpgradeTags: List of [Code[250]]);
    begin
        PerCompanyUpgradeTags.Add(GetABCShoeSizeUpgradeTag());
    end;

    // Use methods to avoid hard-coding the tags. It is easy to remove afterwards because it's compiler-
driven.
    procedure GetABCShoeSizeUpgradeTag(): Text
    begin
        exit('ABC-1234-ShoeSizeUpgrade-20201125');
    end;

}
```

Protecting sensitive code from running during upgrade

The extension might initiate code that you don't want to run during upgrade. The changes done to the data stored in the database will be rolled back. However, things like calls to external web services or physical printing can't be rolled back. Also, some code, like scheduling tasks, might throw an error and fail the upgrade.

For example, let's say the extension runs code that prints a check after a purchase invoice is posted for buying shoes. If the upgrade fails, the purchase invoice is rolled back. But the check will still be printed, unless you have implemented a mechanism to prevent printing.

To avoid this situation, use the session `ExecutionContext`. Depending on the scenario, the system runs a session in a special context for a limited time, which can be either `Normal`, `Install`, `Uninstall`, or `Upgrade`. You get the `ExecutionContext` by calling the GetExecutionContext method. For example, referring the example for printing checks, you could add something like the following code to verify the ExecutionContent before printing the check:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Purch.-Post", 'OnAfterPurchInvHeaderInsert', '', false,
false)]
local procedure PrintCheckWhenPurchasingShoes(var PurchHeader: Record "Purchase Header"; var PurchInvHeader:
Record "Purch. Inv. Header")
begin
    // Check whether global context is upgrade or installation of extension
    if Session.GetExecutionContext() <> ExecutionContext::Normal then
        // Check whether code is triggered by the extension
        if Session.GetCurrentModuleExecutionContext() <> ExecutionContext::Normal then
            // Something is wrong, so you want to abort here because the code doesn't raise the
EnqueuePrintingCheck trigger
            Error('Check can't be printed')
        else begin
            // Other code is invoking the upgrade, so use Job queue or similar mechanism to roll back if
upgrade fails
            EnqueuePrintingCheck(PurchInvHeader);
            exit;
        end;

    CallWebServiceToPrintCheck(PurchInvHeader);
end;
```

Running the upgrade for the new extension version

To upgrade to the new extension version, you use the Sync-NavApp and Start-NAVAppDataUpgrade cmdlets of the Business Central Administration Shell. These cmdlets synchronize table schema changes in the extension with the SQL database and run the data upgrade code.

1. Publish the new extension version. For simplicity, this example assumes the extension isn't signed, which isn't allowed with Dynamics 365 and isn't recommended with an on-premise production environment.

   ```
   Publish-NAVApp -ServerInstance BC -Path .\ProswareStuff_1.7.1.0.app -SkipVerification
   ```

   This step validates the extension syntax against server instance, and stages it for synchronizing.

2. Synchronize the new extension version with the database.

   ```
   Sync-NAVApp -ServerInstance BC -Name ProswareStuff -Version 1.7.1.0
   ```

   This step synchronizes the database with any table schema changes in the extension; it adds the tables from the extension to the tenant.

3. Run a data upgrade.

```
Start-NAVAppDataUpgrade -ServerInstance BC -Name ProswareStuff -Version 1.7.1.0
```

This step runs the upgrade logic that is defined by the upgrade codeunits in the extension. This step will uninstall the current extension version and enable the new version instead.

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension
Converting Extensions V1 to Extensions V2
Sample Extension
Analyzing Extension Upgrade Telemetry
Analyzing Extension Lifecycle Telemetry

Publishing and Installing an Extension v2.0

2/17/2021 • 2 minutes to read • Edit Online

To make your extension available to tenant users requires three basic tasks: publish the extension package to the Dynamics 365 Business Central server instance, synchronize the extension with the tenant database, and install the extension on the tenant.

> **NOTE**
> This article describes how to publish and install the first version of an extension. If you want to publish an install newer version of an extension, see Upgrading Extensions.

Publish and synchronize an extension

Publishing an extension to a Dynamics 365 Business Central server instance adds the extension to the application database that is mounted on the server instance, making it available for installation on tenants of the server instance. Publishing updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

Synchronizing an extension updates the database schema of the tenant database with the database schema that is defined by the extension objects. If a table or table extension is included in the extension, the respective full or companion table is created in the tenant database.

To publish and synchronize an extension

1. Start the Dynamics NAV Administration Shell.
2. To publish the extension, run the Publish-NAVApp cmdlet.

   The cmdlet takes as parameters the Dynamics 365 Business Central service instance that you want to install to and the .app package file that contains the extension. The following example publishes the extension **MyExtension.app** to the **YourDynamicsNAVServer** instance.

   ```
   Publish-NAVApp -ServerInstance YourDynamicsNAVServer -Path ".\MyExtension.app"
   ```

   > **NOTE**
   > If you are publishing a version 16.0 Microsoft extension, you'll have to include the `-SkipVerification` parameter. Otherwise, you will get the error "Publish-NAVApp : You cannot publish an extension that has not been code signed.". This issue is fixed in later versions.

3. To synchronize the schema of a tenant database to the extension, run the Sync-NavApp cmdlet.

   The following example synchronizes the extension **MyExtension** with version number 1.0.0.0:

   ```
   Sync-NavApp -ServerInstance YourDynamicsNAVServer -Name ExtensionName -Version 1.0.0.0 -Tenant
   TenantID
   ```

   Replace `TenantID` with the tenant ID of the database. If you don't have a multitenant server instance, use `default` or omit this parameter.

The extension can now be installed on tenants.

Install an extension

After you publish and synchronize an extension, you can install it on tenants. The extension is then available to users in the client. Installing an extension can be done from the Dynamics 365 client or Dynamics NAV Administration Shell.

> **NOTE**
> Installing an extension will run any installation code that is built-in to the extension. Installation code could, for example, perform operations like populating empty records with data, service callbacks and telemetry, version checks, and messages to users. For more information, see Writing Extension Install Code.

To install an extension by using Dynamics NAV Administration Shell

1. Start the Dynamics NAV Administration Shell.
2. To install the extension on one or more tenants, use the `Install-NAVApp` cmdlet.

   The following example installs the extension **My Extension** for Tenant1 and Tenant3. In single-tenant deployments, you either specify `default` as the tenant ID, or you omit the `-Tenant` parameter.

   ```
   Install-NAVApp -ServerInstance YourDynamicsNAVServer -Name "My Extension" -Tenant Tenant1, Tenant3
   ```

To install an extension by using the client

1. In Dynamics 365 Business Central, use search to open the **Extension Management** page.

   In the **Extension Management** window, you can view the extensions that are published to your server. For each extension, you can see the current installation status.
2. Choose an extension to see additional information and to install the extension.
3. Review and accept the license agreement.
4. Choose the **Install** button to install the extension.

See Also

Unpublishing and Uninstalling Extensions
Developing Extensions
Analyzing Extension Lifecycle Telemetry

Upgrading AppSource Apps in Production

2/17/2021 • 2 minutes to read • Edit Online

When an updated version of an AppSource app becomes the active version in the Dynamics 365 Business Central service, tenants do not automatically get this updated version. This upgrade must be done manually by getting the latest version of the app in AppSource.

To upgrade an AppSource app

Follow the steps below to update a tenant to the latest version of any AppSource app.

1. Log in to the Dynamics 365 Business Central browser client.
2. In the **Tell Me** box, enter **Extension Management**, and then choose the related link.
3. Locate the app that you want to update.
4. Under **Manage**, choose **Uninstall**.
5. Go to AppSource and then install the app again.

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension

Generating Delta files

2/17/2021 • 2 minutes to read • Edit Online

You can use the `Compare-NAVApplicationObject` PowerShell cmdlet to generate .delta files from two versions of a set of application objects. For more information about the PowerShell cmdlet, see Compare-NAVApplicationObject.

The cmdlet has a `ExportToNewSyntax` switch that allows generating .delta files that can be used as a starting point for creating extensions. Setting the `ExportToNewSyntax` flag generates .delta files that contain additional information needed to generate the correct structure and layout of extension objects.

> **IMPORTANT**
> The Txt2Al conversion tool will reject .delta files that were generated without using the -ExportToNewSyntax flag.

Using the `ExportToNewSyntax` switch for the `Compare-NAVApplicationObject` cmdlet produces a .delta file that can be converted to an extension.

The ExportToNewSyntax flag

PARAMETER	DESCRIPTION
Type	SwitchParameter
Aliases	None
Position	Named
Default value	None
Accept pipeline input	False
Accept wildcard characters	False

Example

```
Compare-NAVApplicationObject -OriginalPath "C:\PageWith2Controls.txt" -ModifiedPath
"C:\PageWith3Controls.txt" -ExportToNewSyntax
```

See Also

The Txt2Al Conversion Tool
Developing Extensions
Converting Extensions V1 to Extensions V2
Compare-NAVApplicationObject

Exporting data for Extensions

2/17/2021 • 2 minutes to read • Edit Online

For your extension to run properly, configuration and starting data such as permission sets and table data may be needed. An extension can include the following types of data that can be imported for the tenant during the installation of the extension.

- Permission sets
- Web services
- Starting table data
- Custom report layouts

The data must be exported into files to be included in the extension. To use the export functions you must use a container sandbox environment for Dynamics 365 Business Central. For more information, see Get started with the Container Sandbox Development Environment.

To export permission sets

1. Open the Business Central Development Shell.
2. Export the relevant permission set using the `Export-NAVAppPermissionSet` cmdlet to export the permission set to a file. For example, the following command exports the BASIC permission set.

   ```
   Export-NAVAppPermissionSet -ServerInstance DynamicsNAV160 -Path '.\PermissionSet.xml' -PermissionSetId
   BASIC
   ```

   NOTE

   Export each permission set to a separate XML file.

3. Add the exported permission set files to the Visual Studio Code project that contains your extension.

   WARNING

   If you do not include a permission set with your extension, only users with the SUPER permission set will be able to use the extension.

To export web services

1. Open the Business Central Development Shell.
2. Export the relevant web service using the `Export-NAVAppTenantWebService` cmdlet to export the web service to a file. The following command exports the Customer Card page.

   ```
   Export-NAVAppTenantWebService -ServerInstance DynamicsNAV160 -Path TenantWebService.xml -ServiceName
   Customer -ObjectType Page -ObjectId 21
   ```

   NOTE

   Export each web service to a separate XML file.

3. Add the exported web services files to the Visual Studio Code project that contains your extension.

To export table data

1. Open the Business Central Development Shell.
2. Export the relevant data using the `Export-NAVAppTableData` cmdlet to export the data to a file. This includes setting the path to a folder where you want the .navxdata file created. A data file in the format of TAB.navxdata will be created. (Example: TAB10000.navxdata).

   ```
   Export-NAVAppTableData -ServerInstance DynamicsNAV160 -Path 'C:\NAVAppTableData' -TableId 10000
   ```

   > **NOTE**
   > Export the data for each table to a separate XML file.

3. Add the exported table data files to the Visual Studio Code project that contains your extension.
4. Call the procedure in a Codeunit with the Subtype property `Install` or `Upgrade` and specify the table ID in the `NavApp.LoadPackageData` procedure as shown in the following example.

   ```
   codeunit 50100 MyExtensionUpgrade
   {
       Subtype = Upgrade;
       trigger OnUpgradePerDatabase()
       begin
           NavApp.LoadPackageData(50100);
       end;
   }
   ```

   > **WARNING**
   > An extension can only include table data for new tables that are added as part of the extension.

To export custom report layouts

1. Open the Business Central Development Shell.
2. Export the relevant report layouts using the `Export-NAVAppReportLayout` cmdlet to export to a file:

   ```
   Export-NAVAppReportLayout -ServerInstance DynamicsNAV160 -Path .\ReportLayout.xml -LayoutId 1
   ```

   > **NOTE**
   > Export each custom report layout to a separate XML file.

3. Add the exported custom report files to the Visual Studio Code project that contains your extension.

See Also

Developing Extensions in AL
Converting Extensions V1 to Extensions V2
Writing Extension Install Code

The Txt2Al Conversion Tool

2/17/2021 • 4 minutes to read • Edit Online

The Txt2Al conversion tool allows you to take C/AL objects, which were created in Dynamics NAV or Business Central Spring 2019 (version 14), and convert them into the new .al format. The .al format is used when developing extensions for Dynamics 365 Business Central. Converting the objects consists of following two steps:

1. Exporting the objects from C/SIDE in a cleaned .txt format.
2. Converting the objects to the new syntax.

Where to get the Txt2Al conversion tool

The Txt2Al conversion tool (txt2al.exe) is only available with version 14, which is the last version to support C/AL. Use this version no matter what later version you may eventually be upgrading to. The AL objects created by the tool will be compatible with later versions.

You find the txt2al.exe on the installation media (DVD) in the "DVD\RoleTailoredClient\program files\Microsoft Dynamics NAV\140\RoleTailored Client" folder. Or, it's installed locally with Dynamics NAV Development Environment, for example, in the "C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client" folder.

Using the Txt2Al conversion tool

To run the Txt2Al conversion tool, follow the steps outlined below.

1. Start with a clean Dynamics NAV database and compile the database.
   It's **very** important that you compile the database to get the right result in the next step.
2. Make an export of **all the baseline objects** in the command line using the following syntax:
   ```
   finsql.exe Command=ExportToNewSyntax, File=<filename.txt>, Database="<databasename>", ServerName=
   <servername> ,Filter=Type=table;ID=<tableID>
   ```
   The following example exports the table **225** from the Demo Database NAV (13-0) database:
   ```
   finsql.exe Command=ExportToNewSyntax, File=exportedBaselineObjects.txt, Database="Demo Database NAV (13-
   0)", ServerName=.\NAVDEMO ,Filter=Type=table;ID=225
   ```
3. Import your solution using the import option in C/SIDE and compile the database.
   It's **very** important that you compile the database to get the right result in the next step.
4. Export all **new and/or modified** objects using the following syntax:
   ```
   finsql.exe Command=ExportToNewSyntax, File=<filename.txt>, Database="<databasename>", ServerName=
   <servername> ,Filter=Type=table;ID=<tableID>
   ```
   The following example exports the table **231** from the Demo Database NAV (13-0) database:
   ```
   finsql.exe Command=ExportToNewSyntax, File=exportedNewModifiedObjects.txt, Database="Demo Database NAV
   (13-0)", ServerName=.\NAVDEMO ,Filter=Type=table;ID=231
   ```
5. Run the Set-ObjectPropertiesFromMenuSuite cmdlet, which will convert MenuSuite information on pages and reports in the generated AL objects to enable them for search. For more information, see Making Pages and Reports Searchable in the Web client
6. Create .delta files using the Compare-NAVApplicationObject PowerShell script. For more information, see Generating DELTA Files.
7. Go to the *\Program Files(x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client* folder and locate the **txt2al.exe** converter tool.
8. Run the tool from the command line using the following syntax:

```
txt2al --source --target --rename --type --extensionStartId --injectDotNetAddIns --dotNetAddInsPackage --
dotNetTypePrefix --translationFormat --addLegacyTranslationInfo
```

Parameters

PARAMETER	DESCRIPTION
--source=Path	Required. The path of the directory containing the .delta files.
--target=Path	Required. The path of the directory into which the converted AL files will be placed.
--rename	Rename the output files to prevent clashes with the source .txt files.
--type=ObjectType	The type of object to convert. Allowed values: Codeunit, Table, Page, Report, Query, XmlPort
--extensionStartId	The starting numeric ID of the extension objects (Default: 70000000). It will be incremented by 1 for each extension object.
--stacktrace	Display the stack trace of exceptions raised during the conversion.
--help	Show help screen.
--injectDotNetAddIns	Inject the definition of standard .NET add-ins in the resulting .NET package. The standard .NET add-ins are a set of add-ins that are embedded into the platform.
--dotNetAddInsPackage=Path	Specify the path to an AL file that contains a definition for a .NET package containing .NET type declarations that should be included in the .NET package definition that's produced by the conversion. This parameter should be used to inject a custom set of .NET control add-in declarations. The file should contain something similar to the example shown below.
--multithreaded	Run using multiple threads. This parameter improves performance but results in non-repeatable extension numbers and .xlf content.
--dotNetTypePrefix	Specify a prefix to be used for all .NET type aliases created during the conversion.
--translationFormat=ObjectType	Specify the format to use when generating translation files. The allowed values are: Xliff, Lcg.
--addLegacyTranslationInfo	Add information to the translation file that can be used to migrate existing translations/translated resources. During conversion, XLIFF files from all the ML properties in the app are extracted. If this switch is set, a comment is added in the generated XLIFF that specifies what the ID of the translation item would be in C/SIDE. This acts as a mapping that allows you to convert existing translation resources for your app.

PARAMETER	DESCRIPTION
--runtime	Specify the target runtime for the converted AL. The default is the latest supported runtime. The string should be in a format similar to `major.minor`.
--objectFileNamePattern	Specify the pattern for naming AL files that contain AL objects by using the following placeholders: `{name}` - object name, `{type}` - object type, `{id}` - object ID. This parameter only takes effect if `--rename` is specified. The default is `{name}.{type}`.
--extensionObjectFileNamePattern	Specify the pattern for naming AL files containing AL extension objects by using the following placeholders: `{name}` - extension object name, `{type}` - extension object type, `{id}` - object ID, `{targetName}` - the name of the object being extended, `{targetId}` - the ID of the object being extended. This parameter only takes effect if `--rename` is specified. The default is `{name}.{type}-{targetName}`.
--format	Format the converted AL code using the standard formatter.
--dataClassificationDefaulting	Specify the DataClassification property for all table fields that don't have it specified. For more information, see DataClassification Property.
--tableDataOnly	For table objects, specifies to convert only the table and field definitions, including properties. Methods and trigger code isn't included. **Note:** This parameter was first introduced in Business Central version 14.2 (cumulative update 11) and Business Central version 15.5.

NOTE

It's recommended to only use the conversion tool for export. Importing objects that have been exported can damage your application.

TIP

You can use the Dynamics NAV Development Shell cmdlet `Export-NAVApplicationObject` with the `-ExportToNewSyntax` flag set instead of using finsql. From the command prompt in the Dynamics NAV Development Shell, run `Get-Help Export-NAVApplicationObject -full` to see the full syntax.

See Also

Developing Extensions
AL Development Environment
Page Extension Object
Report Object
Page Properties

Converting Extensions V1 to Extensions V2

2/17/2021 • 5 minutes to read • Edit Online

Extensions are a programming model where functionality is defined as an addition to existing objects and defines how they are different or modify the behavior of the solution. This article explains the steps involved in converting V1 extensions, written in C/SIDE. to V2 extensions; written using the AL Language extension for Visual Studio Code. The overall steps for the conversion are:

1. Convert the source code from C/AL to the AL syntax.
2. Complete the development of the extension in AL syntax.
3. Write upgrade code to restore and modify data from the V1 Extension tables.
4. Build the extension.
5. Uninstall the V1 extension, and publish and run upgrade on the V2 extension.

> **IMPORTANT**
> Converting extensions V1 to extensions V2 can only be done on Business Central Spring 2019 (version 14) and earlier versions. It isn't supported on later Business Central versions, because it requires a working C/AL base application and several ArchiveData functions, which are not available in later versions.

Convert the source code from V1 to V2

To convert the source code, you must use the Txt2Al conversion tool. The Txt2Al conversion tool allows you to take existing application objects that have been exported in .txt format and convert them into the new .al format. The .al format is used when developing extensions for Business Central. For more information about converting the source code, see Txt2Al Conversion Tool.

Complete the development of the extension

When the source code has been converted using the Txt2Al conversion tool, open the project folder in Visual Studio Code, and then modify or add code to the new version as needed. For more information about getting started with Visual Studio Code and the AL Language extension, see Getting Started with AL.

You might run into compilation errors, which can typically be caused by:

- Object IDs that have changed. The conversion tool tries to convert your code into the object ID range allowed for Extensions V2.
- Field or control names look different; the AL syntax requires names, this means that no empty or default names are allowed.
- Menu suites do not exist in Extensions V2.
- .NET references are not allowed; there is no support for .NET types. Instead you must use the classes that replace .NET calls. For more information, see Reference.

IMPORTANT

In the `app.json` file, keep the ID the same as in the V1 extension. Also, make sure to increase the version number.

The version number has the format `Major.Minor.Build.Revision`, for example `1.5.0.0`. To increase the version number, you must increase the value of `Major`, `Minor`, or `Build` by at least one, for example `1.5.1.0` or `1.6.0.0`.

To use `NAVAPP.RestoreArchiveData()` method for upgrading, you must not change the IDs of the tables that are being restored; this means that tables from your V1 extension must have the same IDs in the V2 extensions.

Write upgrade code to move data from V1 Extensions

Just like with V1 extensions, you have to write code to handle data in tables during upgrade. Writing code for the V1-to-V2 extension upgrade is very similar to the code that you have been writing for V1 Extensions. The differences are:

- Instead of adding code to normal codeunit, you write code in an upgrade codeunit, which is a codeunit whose SubType property is set to **Upgrade**.
- Instead of adding code to the user-defined methods `OnNavAppUpgradePerDatabase()` or `OnNavAppUpgradePerCompany()`, you add code to one or more of the following system triggers for data upgrade. These triggers are invoked when a data upgrade is started. The following table lists the upgrade triggers in the order in which they run.

TRIGGER	DESCRIPTION
OnCheckPreconditionsPerCompany() or OnCheckPreconditionsPerDatabase()	Used to check that certain requirements are met in order to run.
OnUpgradePerCompany() or OnUpgradePerDatabase()	Used to run the actual upgrade work
OnValidateUpgradePerCompany() or OnValidateUpgradePerDatabase()	Used to check that the upgrade was successful

However, for this one-time conversion, all of the same **NAVAPP** system methods you used in V1 extensions work with V2 extensions and can be called from any of the upgrade triggers.

METHOD	DESCRIPTION
`NAVAPP.DeleteArchiveData(70000000)`	Deletes the archived data from table 70000000.
`NAVAPP.GetArchiveRecordRef(70000000, archRef)`	Gets a record ref to the archived data from table 70000000.
`archVersion := NAVAPP.GetArchiveVersion()`	Gets the version of the archived data from the old extension.
`NAVAPP.RestoreArchiveData(70000000)`	Restores the data from the archive of table 70000000.

By using these methods, you can restore or move all your data from the old V1 extension into the new V2 by running an upgrade.

IMPORTANT

To use `NAVAPP.RestoreArchiveData()`, you must not change the IDs of the tables that are being restored; this means that tables from your V1 extension must have the same IDs in the V2 extensions.

Example

This code illustrates a simple upgrade codeunit for restoring the V1 extension data for extension table `70000000`.

```
codeunit 70000001 MyExtensionUpgrade
{
    Subtype=Upgrade;

    trigger OnUpgradePerDatabase();
    begin
        NAVAPP.RestoreArchiveData(70000000);
    end;
}
```

TIP

Typing the shortcut `ttrigger` in Visual Studio Code will create the basic structure for a trigger.

Build the extension package

Press Ctrl+Shift+B to compile and build the extension complete with the application objects and upgrade codeunit.

Run the upgrade

The final task of the conversion is to publish the V2 extension, and run the data upgrade. The following steps use an example that upgrades a V1 extension that is called 'ProsewareStuff' and has the version '1.5.0.0.'. The V1 extension is published, installed, and populated with data. The V2 extension has the same name (and ID), but it has the version '1.5.1.0'. The Dynamics 365 Business Central service instance is called 'DynamicsNAV', and there is only one tenant.

The steps use the Dynamics NAV Server Administration tool.

1. Uninstall the V1 extension.

   ```
   Uninstall-NAVApp -ServerInstance NAV -Name ProsewareStuff -Version 1.5.0.0
   ```

   This removes the tables from the SQL Server database and archives extension data.

   IMPORTANT

   The V1 extension must be uninstalled before upgrading it to a V2 extension.

2. Publish the V2 extension. This example assumes the extension is not signed.

   ```
   Publish-NAVApp -ServerInstance DynamicsNAV -Path .\ProswareStuff_1.5.1.0.app -SkipVerification
   ```

   This validates the extension syntax against server instance, and stages it for syncing.

3. Synchronize the V2 extension with the database.

```
Sync-NAVApp -ServerInstance NAV -Name ProswareStuff -Version 1.5.1.0
```

   This adds tables from V2 extension to SQL database.

4. Run the upgrade process to handle archived data from the V1 extension.

```
Start-NAVAppDataUpgrade -ServerInstance NAV -Name ProswareStuff -Version 1.5.1.0
```

   This runs the upgrade logic defined by the upgrade codeunit in the extension, and installs the new V2 extension.

5. (optional) Unpublish the V1 extension.

```
Unpublish-NAVApp -ServerInstance NAV -Name ProswareStuff -Version 1.5.0.0
```

   This removes the unused extension package from server.

Going forward

The upgrade code unit becomes an integral part of the extension. The **NAVAPP** methods were mainly be used for the conversion from V1 to V2. After converting the extension, you should begin to write upgrade code as described in Upgrading Extensions.

See Also

Getting Started with AL
Keyboard Shortcuts
AL Development Environment

The Microsoft_Application.app File

2/17/2021 • 3 minutes to read • Edit Online

The Microsoft_Application.app file is included with Business Central and is located in the `\Applications\Application\Source` folder. The `Microsoft_Application.app` file logically encapsulates all of the extensions making up a solution, for example, version `16.0.0.0` of the base and system application package files, and it provides a convenient way to define and refer to this solution identity.

> **NOTE**
>
> In previous versions the references to base and system application were stated explicitly under `dependencies` in the `app.json` file of the extension. Instead you must now use the `Application` version property in the `app.json` file. For more information, see JSON Files.

The file name of the reference is `Microsoft_Application.app` and in the app.json file of the application package file, the name is `Application`. For code-customized base applications that have their own appId, the `Microsoft_Application.app` file can be modified to reference the appId of the code-customized base applications instead. This allows any extensions that are dependent on the `Application` to resolve to the custom appId.

> **NOTE**
>
> The partner who redefines the application must ensure that extensions that are dependent on the `Application` compile and work. This can be ensured, for example, by not introducing any breaking changes.

> **IMPORTANT**
>
> If you have modified the `Microsoft_Application.app` file, you can rename the file name, and change information about the `publisher`, but it is important to keep `"name": "Application"` in the extension, which is what is being checked for in terms of symbols references. It is also important to keep the `propagateDependencies` set to `true`. The `version` must be set to the version of the Microsoft base application with which it is compatible.

Changing the app.json file for a code-customized base application

The `app.json` file of the `Microsoft_Application.app` file looks like the following example for Business Central version 15.3.

```
{
    "id":  "c1335042-3002-4257-bf8a-75c898ccb1b8",
    "name":  "Application",
    "publisher":  "Microsoft",
    "version":  "15.3.40074.40254",
    "propagateDependencies":  true,
    "logo":  "ExtensionLogo.png",
    "privacyStatement":  "https://go.microsoft.com/fwlink/?LinkId=724009",
    "EULA":  "https://go.microsoft.com/fwlink/?linkid=2009120",
    "help":  "https://go.microsoft.com/fwlink/?linkid=2104024",
    "url":  "https://go.microsoft.com/fwlink/?LinkId=724011",
    "dependencies":  [
                         {
                             "appId":  "437dbf0e-84ff-417a-965d-ed2bb9650972",
                             "name":  "Base Application",
                             "publisher":  "Microsoft",
                             "version":  "15.3.0.0"
                         },
                         {
                             "appId":  "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
                             "name":  "System Application",
                             "publisher":  "Microsoft",
                             "version":  "15.3.0.0"
                         }
                     ],
    "screenshots":  [

                    ],
    "platform":  "15.0.0.0",
    "showMyCode":  true,
    "brief":  "Application (W1)"
}
```

If you have a code-customized base application, the file can be edited to reflect the dependency to this instead. To do so, update the `"dependencies": []` section and change the `"appId": "437dbf0e-84ff-417a-965d-ed2bb9650972"` to the `appId` of your code-customized base application. You can update the `"name"` and `"publisher"` information to match too.

```
...
"dependencies":  [
                         {
                             "appId":  "<appId of the code-customized base app>",
                             "name":  "Customized Base Application",
                             "publisher":  "PartnerSolutions",
                             "version":  "15.3.0.0"
                         },
                         ...
                     ],
...
```

Uptaking the Application app

The Application app logically encapsulates apps making up a solution (such as Base Application and System Application), and provides an abstraction to protect the AppSource and PTE extensions running on top of that solution from not being able to resolve dependencies to these apps.

When using it, future refactoring of the referenced solution; like extracting some areas into separate apps, or changes to the identities of the apps which comprise the solution, will not be forcing all other dependent apps to change or add new apps to their dependencies, as these dependencies will be resolved implicitly via the reference to the Application app.

Additionally, it is meant to simplify the on-premises upgrade scenarios, when customizations are extracted from the Base Application into extensions. And finally it makes it possible to make the same apps available not only to the Business Central customers, but also to the customers of the rich, vertical solutions called Embed Apps, which are also running in the Business Central service.

To enable these benefits, all you need to do, as an AppSource or PTE app owner, is to add the `"application"` property in the `app.json` file of your app and provide the minimum Microsoft Base Application version that this app is compatible with. For more information, see JSON Files. Also, you need to remove the direct dependencies to the Base Application and System Application from the `app.json` file. See the following example:

```
{
    "id":  "e5645aaf-74be-453a-ab50-2e34ec3ee53c",
    "name":  "Fabrikam Gadgets Management",
    "publisher":  "Fabrikam",
    "version":  "15.3.41056.29085",
    "logo":  "ExtensionLogo.png",
    "privacyStatement":  "https://go.microsoft.com/fwlink/?LinkId=724009",
    "EULA":  "https://go.microsoft.com/fwlink/?linkid=2009120",
    "help":  "https://go.microsoft.com/fwlink/?linkid=2104024",
    "url":  "https://go.microsoft.com/fwlink/?LinkId=724011",
    "application": "15.3.0.0",
    "dependencies":  [
                     ],
    "screenshots":  [

                    ],
    "platform":  "15.0.0.0",
    "showMyCode":  true,
    "brief":  "Fabrikam Gadgets Mgt."
}
```

IMPORTANT

Soon up-taking the Application app will also be a mandatory requirement for AppSource apps, enforced by the AppSource technical validation. Thus it is highly recommended to change the existing AppSource apps at first convenience, for example with your next planned app update, and adopt the `"application"` property for all new AppSource apps. We also recommend up-taking the Application app for the customized Base Applications on-premise, and per-tenant-extensions (PTEs) that you use in the Business Central online environments.

See Also

JSON Files
Install an Update

Publishing a Code-Customized Base Application for Business Central On-Prem

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

This topic describes the steps and development environment configuration settings that are needed in order to customize the Base Application code in Dynamics 365 Business Central on-premises and publish the code-customized Base Application to the local server.

> **IMPORTANT**
> Instead of code-customizing the Base Application, it is *strongly* recommended to create extensions whenever possible.

> **NOTE**
> The steps in this topic are not validated against a Docker environment. If you are running the on-premises installation and development from Docker, there can be dependencies and other steps that you need to take into consideration.

Prerequisites

Make sure to have the following prerequisites installed to be able to follow the steps in this topic.

- Dynamics 365 Business Central on-premises with the AL Language Development Environment option installed
- Visual Studio Code
- The AL Language extension

To publish a code customization for Business Central on-premises

1. Get the Base Application source from the `/Applications/BaseApp/Source` folder on the DVD.
2. Unzip the *BaseApplication.source.zip* file and open the source folder in Visual Studio Code. This folder contains all of the base application objects and an `app.json` file with the settings enabled for `OnPrem`.
3. Now, you must configure your `launch.json` file settings to the local server. For more information, see JSON Files.
4. Next, download symbols for the Base Application using **Ctrl+Shift+P** and then choose **Download Symbols**.
5. Customize the Base Application. In this example, we will just modify the text in the **Name** field on the **Customer Card** page to be **Strong**. So, in the `CustomerCard.Page.al` file, we specify the following extra line of code:

```
...
field(Name; Name)
{
    ApplicationArea = All;
    Importance = Promoted;
    ShowMandatory = true;

    Style = Strong;       // Show name in bold

    ToolTip = 'Specifies the customer''s name. This name will appear on all sales documents for the
customer.';

    trigger OnValidate()
    begin
        CurrPage.SaveRecord;
    end;
}
...
```

6. Use the Business Central Administration Console to ensure that the settings for developing for on-premises are correctly set. On the **Development** tab these must be:
   - **Allowed Extension Target Level** is set to **OnPrem**.
   - **Enable Developer Service Endpoint** checkbox is selected.
7. In the `app.json` file, in the `dependencies` section, make sure that `version` is set to the version of the System Application found in the project under `.alpackages`. For example:

```
"dependencies": [
    {
    "appId": "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
    "publisher": "Microsoft",
    "name": "System Application",
    "version": "16.0.10037.0"
    }
],
```

8. Configure **User Settings** or **Workspace Settings** to include the following paths for the `"al.assemblyProbingPaths"` setting. For more information, see AL Language Extension Configuration.

```
"al.assemblyProbingPaths": [
    "C:\\Program Files\\Microsoft Dynamics 365 Business Central\\170",
    "C:\\Program Files (x86)\\Microsoft Dynamics 365 Business Central\\170",
    "C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\Framework\\.NETFramework\\v4.8",
    "C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\WindowsPowerShell"
],
```

9. For improved performance when working with a large project like the Base Application, see the Optimize Visual Studio Code Editing and Building Performance topic.
10. Next, you must uninstall all extensions from the command line. If you at this point try to just publish the extension from Visual Studio Code you will get the following error:

```
The request for path /BC170/dev/apps?SchemaUpdateMode=synchronize&DependencyPublishingOption=ignore
failed with code 422. Reason: The extension could not be deployed because it is already deployed on
another tenant.
```

    You get the error because the extensions are installed in the `global` scope. If you want to publish an extension from Visual Studio Code, within the `developer` scope, you will have to first uninstall and then unpublish the extensions from the command line.

Ideally, you should uninstall the application that you want to update and all its dependencies. To uninstall the Base Application use the following cmdlet:

```
Uninstall-NavApp -Name "Base Application" -ServerInstance BC170 -Force
```

Use the `-Force` parameter to uninstall all dependencies.

> **NOTE**
>
> Alternatively, uninstall everything using the following cmdlet.
>
> ```
> Get-NAVAppInfo -ServerInstance BC170 | %{Uninstall-NAVApp -Name $_.Name -ServerInstance BC170 -
> Force}
> ```

11. Having uninstalled, the next step is to unpublish the application and all of its dependencies. This needs to happen before we can publish it. The following command can be used if there are no dependencies to the Base Application:

    ```
    Unpublish-NavApp -Name "Base Application" -ServerInstance BC170
    ```

    But if there are dependencies on the Base Application this will give an error. To solve this, you must unpublish all the applications with dependencies on the Base Application. This is easier to do using a script that recursively removes the dependencies. Use Windows PowerShell ISE to create a new script with the following lines of code:

    ```
    function UnpublishAppAndDependencies($ServerInstance, $ApplicationName)
    {
         Get-NAVAppInfo -ServerInstance $ServerInstance | Where-Object {
        # If the dependencies of this extension include the application that we want to unpublish, it
    means we have to unpublish this application first.
        (Get-NavAppInfo -ServerInstance $ServerInstance -Name $_.Name).Dependencies | Where-Object
    {$_.Name -eq $ApplicationName}
     } | ForEach-Object {
        UnpublishAppAndDependencies $ServerInstance $_.Name
     }

     Unpublish-NavApp -ServerInstance $ServerInstance -Name $ApplicationName
    }

    function UninstallAndUnpublish($ServerInstance, $ApplicationName)
    {
        Uninstall-NavApp -ServerInstance $ServerInstance -Name $ApplicationName -Force
        UnpublishAppAndDependencies $ServerInstance  $ApplicationName

    }

    UninstallAndUnpublish -ServerInstance "BC170" -ApplicationName "Base Application"
    ```

12. Save the script as **unpublish.ps1**.

13. Run the script from the PowerShell commandline to handle the uninstall and unpublishing of the Base Application and its dependencies: `.\unpublish.ps1`

    > **NOTE**
    >
    > Use `"dependencyPublishingOption": "Ignore"` in the `launch.json` file to only publish this extension. For more information, see JSON Files.

14. Import a license with rights to publish the extension. For example:

```
Import-NAVServerLicense -ServerInstance BC170 -LicenseFile "C:\Users\mylicense.flf"
```

15. Press **Ctrl+F5** to publish the modified Base Application as an extension from Visual Studio Code.

The Base Application is now published with the small customization of bolding the text in the **Name** field on the **Customer Card** page.

See Also

Unpublishing and Uninstalling Extensions
Developing Extensions

Extending Application Areas

2/17/2021 • 5 minutes to read • Edit Online

Application areas represent a feature in the system that offers developers, administrators, and users the ability to define differentiated user experiences.

Application areas are mapped to controls to show or hide them on page objects to enable more or fewer business scenarios.

Extending application areas and the experience tier

In this example you will:

- Add a new application area in the **Application Area Setup** table.
- Enable the application area in the **OnInstallAppPerCompany** trigger.
- Extend the experience tier in the **OnGetExperienceAppArea** event.
- Modify the experience tier (optional).
- Validate the application area in the **OnValidateApplicationAreas**.

> **IMPORTANT**
> The code used in this example is still under active development and might be subject to change in the future.

The following example extends the **Customer List** page. The field **ExampleField** is added and it is followed by a series of properties. The **ApplicationArea** property sets the application areas that apply to the control and in this code, **ExampleAppArea** is assigned to it.

> **IMPORTANT**
> If your extension fails to use **ApplicationArea** in any controls or actions, they will not be visible when you use an experience tier.

The **OnOpenPage** trigger will display the message only if **ApplicationArea** is enabled.

```
pageextension 50100 CustomerListExt extends "Customer List"
{
    layout
    {
        addafter(Name)
        {
            field(ExampleField; "Name 2")
            {
                Caption = 'Example Field';
                ApplicationArea = ExampleAppArea;
                ToolTip = 'This is a field added by an example extension';
                Importance = Promoted;
            }
        }
    }

    trigger OnOpenPage()
    var
        EnableExampleExtension : Codeunit "Enable Example Extension";
    begin
        if EnableExampleExtension.IsExampleApplicationAreaEnabled() then
            Message('App published: Example Extension');
    end;
}
```

Adding an application area

To add an application area, the **Application Area Setup** table must be extended. A new boolean field is added and the name of this field will be used in the attribute that you want to be tagged with this application area. This particular case, in the code below, is an exception, because space is used inside it. Usually, spaces are omitted in the application area attribute. At this point, the extension has an application area but it still needs to be enabled.

```
tableextension 50100 "Application Area Setup" extends "Application Area Setup"
{
    fields
    {
        // Spaces in field name are omitted in the ApplicationArea attribute
        // e.g. ApplicationArea = ExampleAppArea;
        field(50100;"Example App Area";Boolean)
        {
        }
    }
}
```

The codeunit **Install Example Extension** is of the subtype Install and it enables the application area inside the **OnInstallAppPerCompany** trigger.

```
codeunit 50101 "Install Example Extension"
{
    Subtype = Install;

    trigger OnInstallAppPerCompany()
    var
        EnableApplicationArea : Codeunit "Enable Example Extension";
    begin
        if(EnableApplicationArea.IsExampleApplicationAreaEnabled()) then
            exit;

        EnableApplicationArea.EnableExampleExtension();

        // Add your code here
    end;
}
```

The registration of the application area inside an experience tier is made inside the **OnGetEssentialExperienceAppArea**. There are different versions of this event, one for each experience tier and in this case, Essential is chosen. This will make the extension visible inside the Essential experience and the event exposes an **Application Area Setup** temporary record; **TempApplicationAreaSetup**, to the **Application Area Setup** table. At this point, to enable the application area, this must be set to true.

> **NOTE**
>
> This event is important because it is called every single time an experience tier is reset, which can happen because of many reasons.

Another thing that is possible inside these methods is to modify the experience tier. You can also modify other application areas, such as creating an extension that extends the Fixed Assets functionality. By subscribing to **OnValidateApplicationAreas**, the application area inside an experience tier is validated. **OnValidateApplicationAreas** is guaranteed to be executed after the events in the **OnGet*ExperienceAppArea** family. The validation is necessary in the presence of extensions concurrently manipulating the same application areas.

In case a needed application area is not enabled, the suggested action is to show an error and turn off the extension to avoid unintended behavior. However, if the functionality controlled by this application area is of secondary importance and its loss does not affect the rest of the extension, it is also appropriate to keep the extension enabled.

```
codeunit 50100 "Enable Example Extension"
{
    // Extend and modify Essential experience tier with "Example App Area"
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Application Area Mgmt.",
'OnGetEssentialExperienceAppAreas', '', false, false)]
    local procedure RegisterExampleExtensionOnGetEssentialExperienceAppAreas(var TempApplicationAreaSetup:
Record "Application Area Setup" temporary)
    begin
        TempApplicationAreaSetup."Example App Area" := true;
        // Modify other application areas here
    end;

    // Validate that application areas needed for the extension are enabled
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Application Area Mgmt.", 'OnValidateApplicationAreas',
'', false, false)]
    local procedure VerifyApplicationAreasOnValidateApplicationAreas(ExperienceTierSetup: Record "Experience
Tier Setup"; TempApplicationAreaSetup: Record "Application Area Setup" temporary)
    begin
        if ExperienceTierSetup.Essential then
            if not TempApplicationAreaSetup."Example App Area" then
                Error('Example App Area should be part of Essential in order for the Example Extension to
work.');
    end;

    // Helpers .....................................................................
    procedure IsExampleApplicationAreaEnabled() : Boolean
    var
        ApplicationAreaSetup: Record "Application Area Setup";
        ApplicationAreaMgmtFacade: Codeunit "Application Area Mgmt. Facade";
    begin
        if ApplicationAreaMgmtFacade.GetApplicationAreaSetupRecFromCompany(ApplicationAreaSetup,
CompanyName()) then
            exit(ApplicationAreaSetup."Example App Area");
    end;

    procedure EnableExampleExtension()
    var
        ApplicationAreaMgmtFacade: Codeunit "Application Area Mgmt. Facade";
    begin
        ApplicationAreaMgmtFacade.RefreshExperienceTierCurrentCompany();
    end;
}
```

Adding Advanced application area to the Essentials and Premium experiences using an extension

If you are familiar with Dynamics NAV you will have noticed that Dynamics 365 Business Central is not exposing all the controls/actions that you find in Dynamics NAV. These controls have been hidden so far by using the application area **Advanced**, which is not assigned to any experiences. For more information, see FAQ.

Most of these fields will become available/visible soon, but until then you will have to create an extension to get (almost) the same experience as you have in Dynamics NAV. See the example below.

> **IMPORTANT**
>
> Adding the application area **Advanced** to the experience will mean that you lose some of the simplification made to pages. For example, you will see more actions duplicated on many pages, compared to Business Central where the experience is intended to be simpler than in Dynamics NAV. You must also consider that we plan to re-tag the **Advanced** actions/controls and add them to the **Essentials** and/or **Premium** experiences in a future release.

To enable Advanced in an extension

Depending on which experience you want to enable **Advanced** for you can subscribe to `OnGetEssentialExperienceAppAreas` or `OnGetPremiumExperienceAppAreas`. If you have defined your own experience you must subscribe to `OnSetExperienceTier`.

The experiences are additive so you only need to subscribe to one of the events. For example, to enable **Essentials** and **Premium** experiences you only need to subscribe to `OnGetEssentialExperienceAppAreas`.

```
codeunit 50102 EnableAdvancedApplicationArea
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Application Area Mgmt. Facade",
'OnGetEssentialExperienceAppAreas','', false, false)]
    local procedure EnableAdvancedApplicationAreaOnGetEssentialExperienceAppAreas(var
TempApplicationAreaSetup : record 9178 temporary)
    begin
        TempApplicationAreaSetup.Advanced := true
    end;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Application Area Mgmt. Facade",
'OnGetPremiumExperienceAppAreas','', false, false)]
    local procedure EnableAdvancedApplicationAreaOnGetPremiumExperienceAppAreas(var TempApplicationAreaSetup
: record 9178 temporary)
    begin
        TempApplicationAreaSetup.Advanced := true
    end;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Application Area Mgmt. Facade",
'OnSetExperienceTier','', false, false)]
    local procedure EnableAdvancedApplicationAreaOnSetExperienceTier(ExperienceTierSetup : record 9176;var
TempApplicationAreaSetup : record 9178 temporary;var ApplicationAreasSet : boolean)
    begin
        TempApplicationAreaSetup.Advanced := true
    end;
}
```

Application areas advantages and disadvantages

If you decide to code application areas as an extension, there are some aspects that must be considered. Application areas enable hiding entire business scenarios and you can have the same code base, which makes it possible to quickly modify the UI for different business scenarios or audiences. However, tagging errors as missing tags or incorrect tags will occur and every single control will need to be tagged.

See Also

ApplicationArea Property
ApplicationArea Method
AccessByPermission Property
Properties

Extending Item Charge Distribution Methods

2/17/2021 • 5 minutes to read • Edit Online

To ensure correct valuation, your inventory items must carry any added costs, such as freight, physical handling, insurance, and transportation that you incur when purchasing or selling the items.

Users can add these costs by adding a Charge (Item) line to the involved purchase or sales document. For more information, see Use Item Charges to Account for Additional Trade Costs in application help.

Item charges are distributed over other item lines in the document according to a distribution method. Dynamics 365 Business Central offers four distribution methods out of the box: **Equally**, **By Amount**, **By Weight**, and **By Volume**. This article explains how to remove or add item charge distribution methods. The article describes the method for purchases. The steps are similar for sales, except the events are located in codeunit 5807, **Item Charge Assgnt. (Sales)**.

To enable extension of item charges distribution methods, two events can be found in codeunit 5805, **Item Charge Assgnt. (Purch.)**. The work consists of the following two tasks:

1. In the **OnBeforeShowSuggestItemChargeAssignStrMenu** event, you manipulate the options that are presented to users. You can remove, add, and change the order of the options.
    - Keep in mind that other extensions may also manipulate the options.
    - You should not assume that an option will exist, nor should you write code that may remove an option added by another extension.
2. In the **OnAssignItemCharges** event, you distribute the item charge amount over the item lines according to your new distribution method.
    - You must verify that the option selected by the user is your new option. If it is not, then exit without taking action.
    - When you have distributed the amount over the lines, you must set the ItemChargesAssigned boolean to true. If you do not set this boolean to true, an error will occur.

The following procedures show how to extend the item charges distribution methods:

1. Add a new option to the item charges distribution methods in the **OnBeforeShowSuggestItemChargeAssignStrMenu** event.
2. Add a new distribution method for item charges.
3. Call the new distribution method in the **OnAssignItemCharges** event.

The procedures are based on an example where the **By Fairy Dust** option is added to the string menu (STRMENU) and added to the CASE statement.

> **NOTE**
> To complete this example you will have to add a new field **Fairy Dust** to the **Purchase Line** table and other relevant tables and pages.

To add a new option to the item charges distribution methods

Create a new codeunit and add an event subscriber to the **OnBeforeShowSuggestItemChargeAssignStrMenu** event.

```
codeunit 50100 "Item Ch. Assign by Fairy Dust"
{
    var
        ByFairyDustTok: Label 'By Fairy Dust';

    local procedure AssignByFairyDustMenuText(): Text
    begin
        exit(ByFairyDustTok)
    end;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Item Charge Assgnt. (Purch.)",
'OnBeforeShowSuggestItemChargeAssignStrMenu', '', false, false)]
    local procedure AddByFairyDustOnBeforeShowSuggestItemChargeAssignStrMenu(PurchLine: Record "Purchase
Line"; var SuggestItemChargeMenuTxt: Text; var SuggestItemChargeMessageTxt: Text; var Selection: Integer)
    begin
        // if 'By Fairy Dust' is not in the menu options, add it at the end
        if StrPos(SuggestItemChargeMenuTxt, AssignByFairyDustMenuText) = 0 then begin
            SuggestItemChargeMenuTxt += ',' + AssignByFairyDustMenuText;
            // make the last option ('By Fairy Dust') the default selection
            Selection := StrLen(DelChr(SuggestItemChargeMenuTxt, '=', DelChr(SuggestItemChargeMenuTxt, '=',
',','))) + 1;
        end;
    end;
}
```

To add a new distribution method for item charges

In the new codeunit, add functions to distribute the charges over the item lines.

```
    local procedure AssignByFairyDust(var ItemChargeAssignmentPurch: Record "Item Charge Assignment
(Purch)"; Currency: Record Currency; TotalQtyToAssign: Decimal; TotalAmtToAssign: Decimal);
    var
        TempItemChargeAssgntPurch: Record "Item Charge Assignment (Purch)" temporary;
        LineArray: array[2] OF Decimal;
        TotalFairyDust: Decimal;
        QtyRemaining: Decimal;
        AmountRemaining: Decimal;
    begin
        // copy lines to temp variable and calculate total Fairy Dust
        repeat
            if not ItemChargeAssignmentPurch.PurchLineInvoiced then begin
                TempItemChargeAssgntPurch.Init();
                TempItemChargeAssgntPurch := ItemChargeAssignmentPurch;
                TempItemChargeAssgntPurch.Insert(false);
                GetItemValues(TempItemChargeAssgntPurch, LineArray);
                TotalFairyDust := TotalFairyDust + (LineArray[2] * LineArray[1]);
            end;
        until ItemChargeAssignmentPurch.Next = 0;

        if TempItemChargeAssgntPurch.Findset(true) then
            repeat
                // Calculate Fairy Dust to assign to the line
                GetItemValues(TempItemChargeAssgntPurch, LineArray);
                if TotalFairyDust <> 0 then
                    TempItemChargeAssgntPurch."Qty. to Assign" :=
                      (TotalQtyToAssign * LineArray[2] * LineArray[1]) / TotalFairyDust + QtyRemaining
                else
                    TempItemChargeAssgntPurch."Qty. to Assign" := 0;

                // Assign Fairy Dust to the line and calculate the remaining Fairy Dust to assign
                ItemChargeAssignmentPurch.Get(
                  TempItemChargeAssgntPurch."Document Type",
                  TempItemChargeAssgntPurch."Document No.",
                  TempItemChargeAssgntPurch."Document Line No.",
                  TempItemChargeAssgntPurch."Line No.");
```

```
                ItemChargeAssignmentPurch."Qty. to Assign" := Round(TempItemChargeAssgntPurch."Qty. to
Assign", 0.00001);
                ItemChargeAssignmentPurch."Amount to Assign" :=
                  ItemChargeAssignmentPurch."Qty. to Assign" * ItemChargeAssignmentPurch."Unit Cost" +
AmountRemaining;
                AmountRemaining := ItemChargeAssignmentPurch."Amount to Assign" -
                  Round(ItemChargeAssignmentPurch."Amount to Assign", Currency."Amount Rounding Precision");
                QtyRemaining := TempItemChargeAssgntPurch."Qty. to Assign" - ItemChargeAssignmentPurch."Qty.
to Assign";
                ItemChargeAssignmentPurch."Amount to Assign" :=
                  Round(ItemChargeAssignmentPurch."Amount to Assign", Currency."Amount Rounding Precision");
                ItemChargeAssignmentPurch.Modify(false);

            until TempItemChargeAssgntPurch.Next = 0;
        TempItemChargeAssgntPurch.DeleteAll(false);
    end;

    procedure GetItemValues(TempItemChargeAssgntPurch: Record "Item Charge Assignment (Purch)" temporary;
var DecimalArray: Array[2] OF Decimal);
    var
        PurchaseLine: Record "Purchase Line";
        PurchRcptLine: Record "Purch. Rcpt. Line";
        ReturnShptLine: Record "Return Shipment Line";
        TransferRcptLine: Record "Transfer Receipt Line";
        SalesShptLine: Record "Sales Shipment Line";
        ReturnRcptLine: Record "Return Receipt Line";
    begin
        // Get the Fairy Dust for the line
        Clear(DecimalArray);
        with TempItemChargeAssgntPurch do
            case "Applies-to Doc. Type" of
                "Applies-to Doc. Type"::Order,
                "Applies-to Doc. Type"::Invoice,
                "Applies-to Doc. Type"::"Return Order",
                "Applies-to Doc. Type"::"Credit Memo":
                    begin
                        PurchaseLine.Get("Applies-to Doc. Type", "Applies-to Doc. No.", "Applies-to Doc.
Line No.");
                        DecimalArray[1] := PurchaseLine.Quantity;
                        DecimalArray[2] := PurchaseLine."Fairy Dust";
                    end;
                "Applies-to Doc. Type"::Receipt:
                    begin
                        PurchRcptLine.Get("Applies-to Doc. No.", "Applies-to Doc. Line No.");
                        DecimalArray[1] := PurchRcptLine.Quantity;
                        DecimalArray[2] := PurchRcptLine."Fairy Dust";
                    end;

                "Applies-to Doc. Type"::"Return Receipt":
                    begin
                        ReturnRcptLine.Get("Applies-to Doc. No.", "Applies-to Doc. Line No.");
                        DecimalArray[1] := ReturnRcptLine.Quantity;
                        DecimalArray[2] := ReturnRcptLine."Fairy Dust";
                    end;
                "Applies-to Doc. Type"::"Return Shipment":
                    begin
                        ReturnShptLine.Get("Applies-to Doc. No.", "Applies-to Doc. Line No.");
                        DecimalArray[1] := ReturnShptLine.Quantity;
                        DecimalArray[2] := ReturnShptLine."Fairy Dust";
                    end;
                "Applies-to Doc. Type"::"Transfer Receipt":
                    begin
                        TransferRcptLine.Get("Applies-to Doc. No.", "Applies-to Doc. Line No.");
                        DecimalArray[1] := TransferRcptLine.Quantity;
                        DecimalArray[2] := TransferRcptLine."Fairy Dust";
                    end;
                "Applies-to Doc. Type"::"Sales Shipment":
                    begin
                        SalesShptLine.Get("Applies-to Doc. No.", "Applies-to Doc. Line No.");
```

```
                    DecimalArray[1] := SalesShptLine.Quantity;
                    DecimalArray[2] := SalesShptLine."Fairy Dust";
                end;
            end;
    end;
```

To call the new distribution method

In the new codeunit, add a subscriber to the **OnAssignItemCharges** event.

```
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Item Charge Assgnt. (Purch.)", 'OnAssignItemCharges',
'', false, false)]
    local procedure AssignByFairyDustOnAssignItemCharges(SelectionTxt: Text; var ItemChargeAssignmentPurch:
Record "Item Charge Assignment (Purch)"; Currency: Record Currency; PurchaseHeader: Record "Purchase
Header"; TotalQtyToAssign: Decimal; TotalAmtToAssign: Decimal; VAR ItemChargesAssigned: Boolean);
    begin
        // if item charges are already assigned, exit
        if ItemChargesAssigned then
            exit;
        // if the user did not choose 'By Fairy Dust', exit
        if not (SelectionTxt = AssignByFairyDustMenuText) then
            exit;
        // assign item charges by fairy dust
        AssignByFairyDust(ItemChargeAssignmentPurch, Currency, TotalQtyToAssign, TotalAmtToAssign);
        // charges have been assigned
        ItemChargesAssigned := true;
    end;
```

See Also

Extending Application Areas

Extending Price Calculations

2/17/2021 • 15 minutes to read • Edit Online

If you record special prices and line discounts for sales and purchases, Dynamics 365 Business Central can automatically calculate prices on sales and purchase documents, and on job and item journal lines. The price is the lowest permissible price with the highest permissible line discount on a given date. Dynamics 365 Business Central automatically calculates the price when it inserts the unit price and the line discount percentage for items on new document and journal lines. For more information, see Price Calculation.

2020 release wave 1 introduces a second implementation of price calculations that will be available as an alternative to the calculations that were available in 2019 release wave 2 and earlier versions. This new implementation has the advantage that it is much easier to extend, for example, with new calculations.

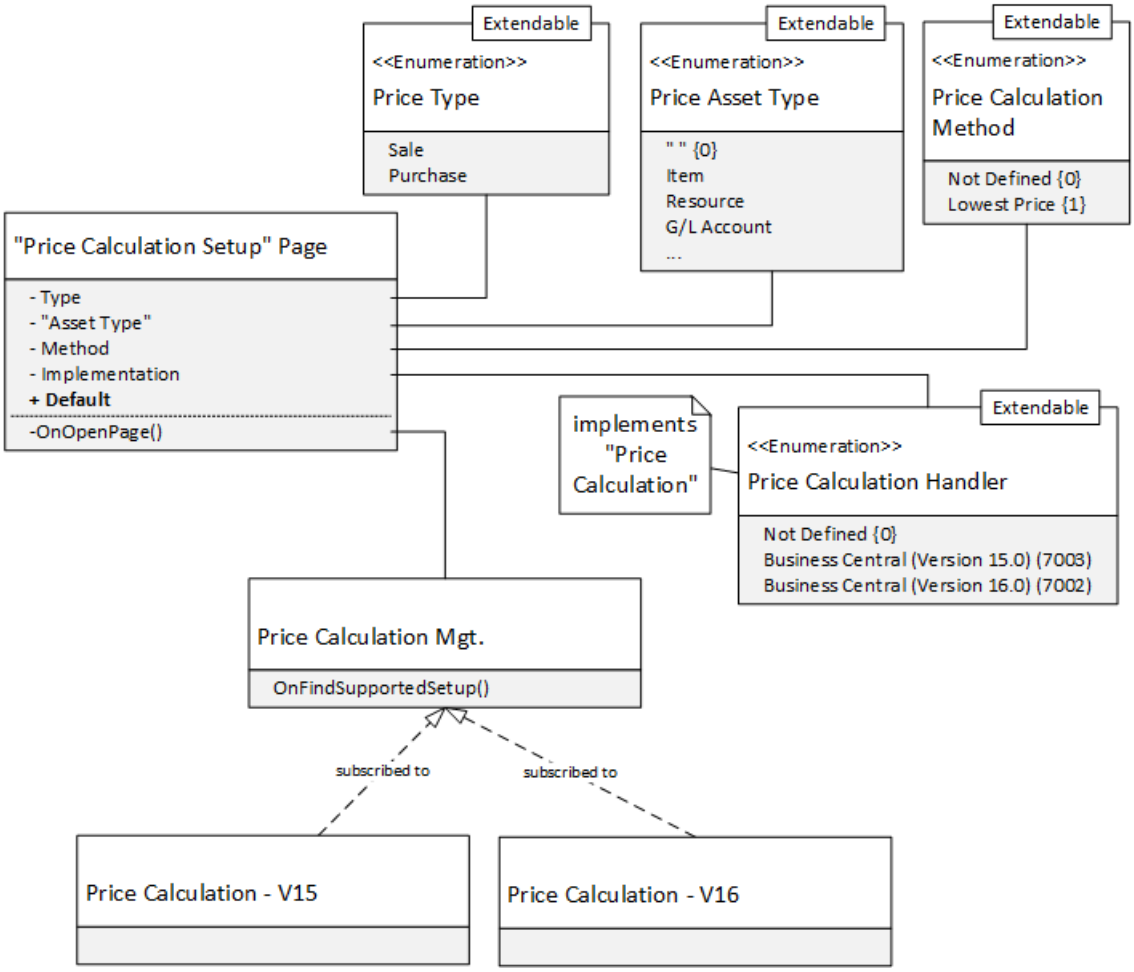
Price calculations that were available in 2019 release wave 2 are unchanged. The new calculations in 2020 release wave 1 are an alternative implementation that you can extend.

> **NOTE**
> The new price calculation capabilities in 2020 release wave 1 exist in code only, and do not include any user experience. We will provide that in an upcoming release. For now, to use the new capability you must create your own page.

This topic describes how price calculations are implemented in 2020 release wave 1 (referred to as "Business Central Version 16" in the illustrations), and provides comparisons with 2019 release wave 2 (referred to as "Business Central Version 15" in the illustrations) to show what we have changed. It also provides some examples of how you can extend price calculations in 2020 release wave 1.

Price Calculation Setup

Price calculation are based on the **Price Calculation Setup** table, where you can choose an existing implementation, as shown in the following image.



The Price Calculation Setup table has the Code field as its primary key. The value of the field is calculated by combining the values in the Method, Type, Asset Type, and Implementation fields. For example, it is '[1-1-0]-7003' for the setup line when the following fields contain the following values:

- Method contains Lowest Price
- Type contains Sale
- Asset Type contains All
- Implementation is Business Central (Version 15.0)"

You can have multiple setups with the same combination of method, type, and asset type. The implementation value should always be different though, because each implementation provides different calculations. The default implementation is defined by the Default field. For example, in 2020 release wave 1 the following setups are available:

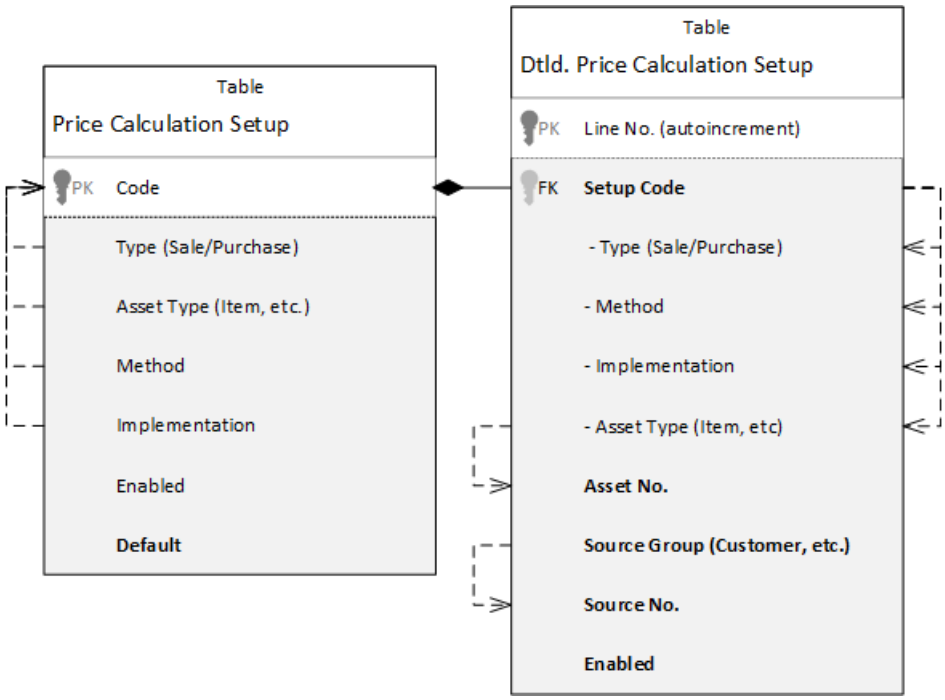
METHOD	TYPE	ASSET TYPE	IMPLEMENTATION	DEFAULT
Lowest Price	Sale	All	Business Central (Version 15.0)	X
Lowest Price	Sale	All	Business Central (Version 16.0)	

By default, all sales lines use the "Business Central (Version 15.0)" implementation to calculate prices, unless the second line has detailed setup lines that define exceptions.

The Price Calculation method on the document line searches for a setup that that has a matching combination of the method, the price type, and asset type on the document line. The method then searches for detailed lines that contain exceptions for the combination of a source group (Customer, Vendor, and Job) and an asset (Item, Resource, and so on) on the document line. If we find a matching setup we use its implementation for price

calculation. If there is no matching setup exception, we use the default implementation.

For example, let's say we have a line on a sales order for Customer 20000 contains item 1000. The default implementation for the sale of any asset is "Business Central (Version 15.0)," but "Business Central (Version 16.0)" implementation contains a detailed setup line for Item 1000. That means that the "Business Central (Version 16.0)" implementation will calculate the price. Detailed setup records are to be entered by users and only make sense for the non-default setup records. The following image shows the relation between the setup line and the detailed setup.



For the Business Central (Version 15.0) implementation, you can only edit the Default field. The records are inserted by the codeunits that subscribe to the OnFindSupportedSetup() event in the Price Calculation Mgt. codeunit. The two price calculation implementation codeunits add pairs of such records, one for the sale of assets and another for purchases. Because the Price Calculation Setup table is extensible, you can add new fields to the key by subscribing to the OnAfterDefineCode() event.

Price Type

The Type field is an extensible Price Type enum that contains the following values:

- Any (0)
- Sale (1)
- Purchase (2)

The values are part of the composite key in the Price Calculation Setup table. Sales and service lines use the Sale type, purchase lines use the Purchase type, and job or item journal lines use both for calculating price and cost. The Any value is the default value, and is used when a line contains both a price and a cost.

Asset Type

The "Asset Type" field is an extensible "Price Asset Type" enum that contains the following values:

- All (0)
- Item (10)
- Item Discount Group (20)
- Resource (30)
- Resource Group (40)
- Service Cost (50)

- G/L Account (60)

The Asset Type is part of the composite key in the Price Calculation Setup table. If the only setup record contains "Asset Type" - All it means that there is no need in special price calculation implementations per asset type. The default implementation will be used regardless of an asset type in the document line. If you need different implementations you must add a setup lines with another asset type. For example, the following table shows a Resource Pricing implementation with a Resource asset type.

METHOD	TYPE	ASSET TYPE	IMPLEMENTATION	DEFAULT
Lowest Price	Sale	All	Business Central (Version 15.0)	X
Lowest Price	Sale	Resource	Resource Pricing	X

Because only one record has the combination of Lowest Price, Sale, and Resource it will be the default. If a sales line sells a resource, its price will be calculated by Resource Pricing implementation. All other assets will use the "Business Central (Version 15.0)" implementation.

Price Calculation Method

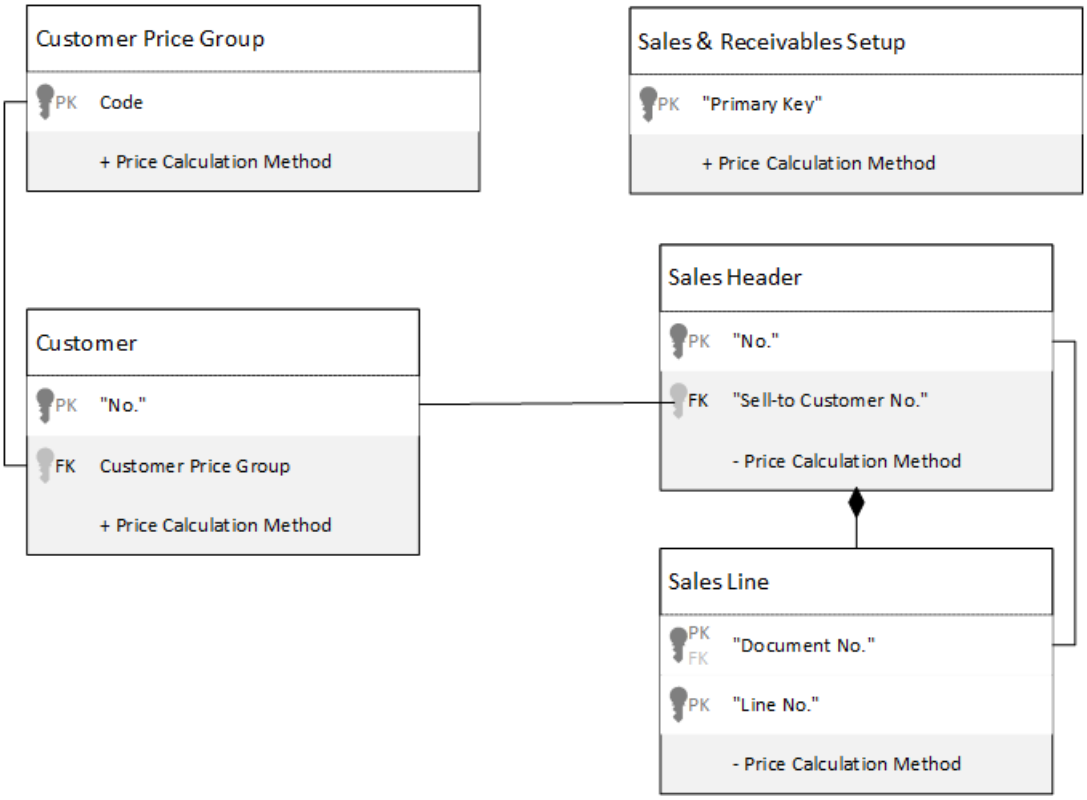
The Method field is an extensible Price Calculation Method enum that contains the following values:

- Not defined (0)
- Lowest Price (1)

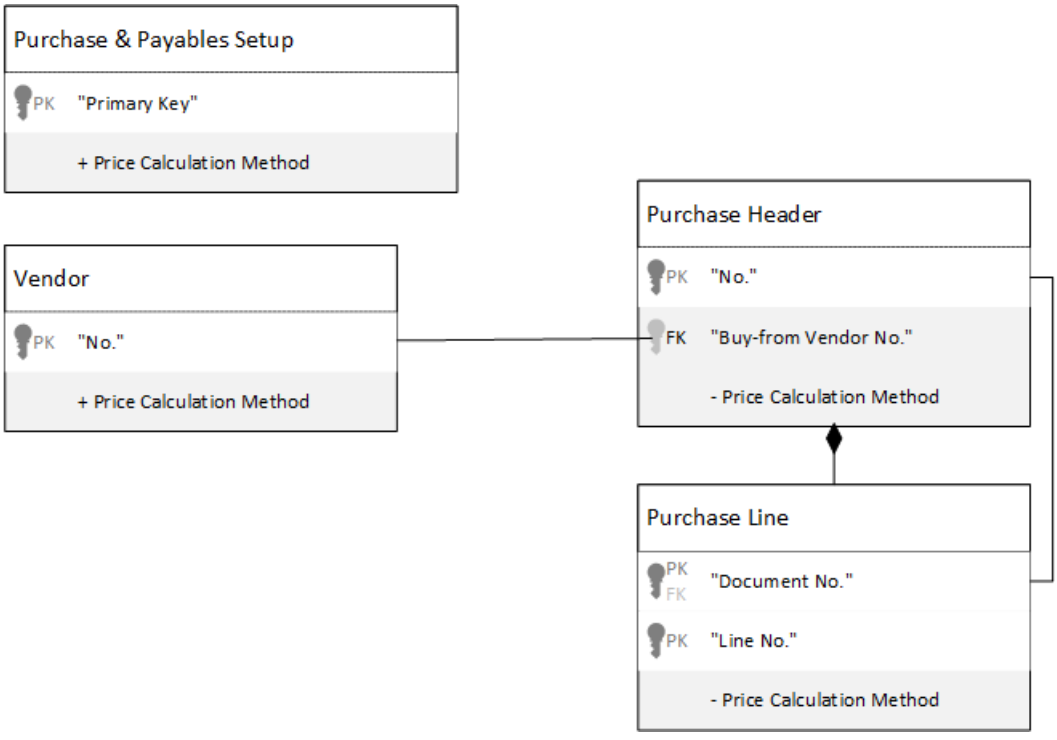
The method value is part of the composite key in the Price Calculation Setup table. A sales document line inherits the method value from the sales header, which in turn inherits it from one of the following, depending on where it's defined:

- Customer
- Customer price group
- Sales & Receivable Setup table

The Sales & Receivable Setup table defines the default method, Lowest Price, for all sales prices. If you want to use a different method, you can specify it on a customer price group or a customer. You cannot edit the method on sales document headers or lines.



The Purchase & Payables Setup table also defines the default method for all purchase prices. You can redefine it for a certain vendor. You cannot edit the method on purchase document headers or lines.



Data Structure Comparison

The Business Central (Version 15.0) calculation uses the following tables that store information about prices, costs, and discounts:

- table7002"SalesPrice"
- table7004"SalesLineDiscount"
- table7012"PurchasePrice"
- table7014"PurchaseLineDiscount"
- table201"ResourcePrice"

- table1014"JobG/LAccountPrice"
- table1013"JobItemPrice"
- table1012"JobResourcePrice"

The Business Central (Version 16.0) calculation uses the following table:

- table7001"PriceListLine"

Table7001"PriceListLine" is compatible with all tables used by the Business Central (Version 15.0) calculation. It contains the set of CopyFrom() methods that convert the data from the tables to the Price List Line table.

If you extended the Business Central (Version 15.0) tables you must also extend the Price List Line table and the CopyFrom() methods by subscribing to special events. Here's are examples that extend the Sales Price table with a Document No. field.

```
tableextension 50010 "Document No in Sales Price" extends "Sales Price"
{
    fields
    {
        field(50000; "Document No."; Code[20])
        { }
    }
}
```

Now we'll extend the Price List Line table with the same field.

```
tableextension 50011 "Doc. No in Price List Line" extends "Price List Line"
{
    fields
    {
        field(50000; "Document No."; Code[20])
        { }
    }
}
```

Now we'll subscribe to the 'OnCopyFromSalesPrice' event to copy data from "Sales Price" to "Price List Line" table.

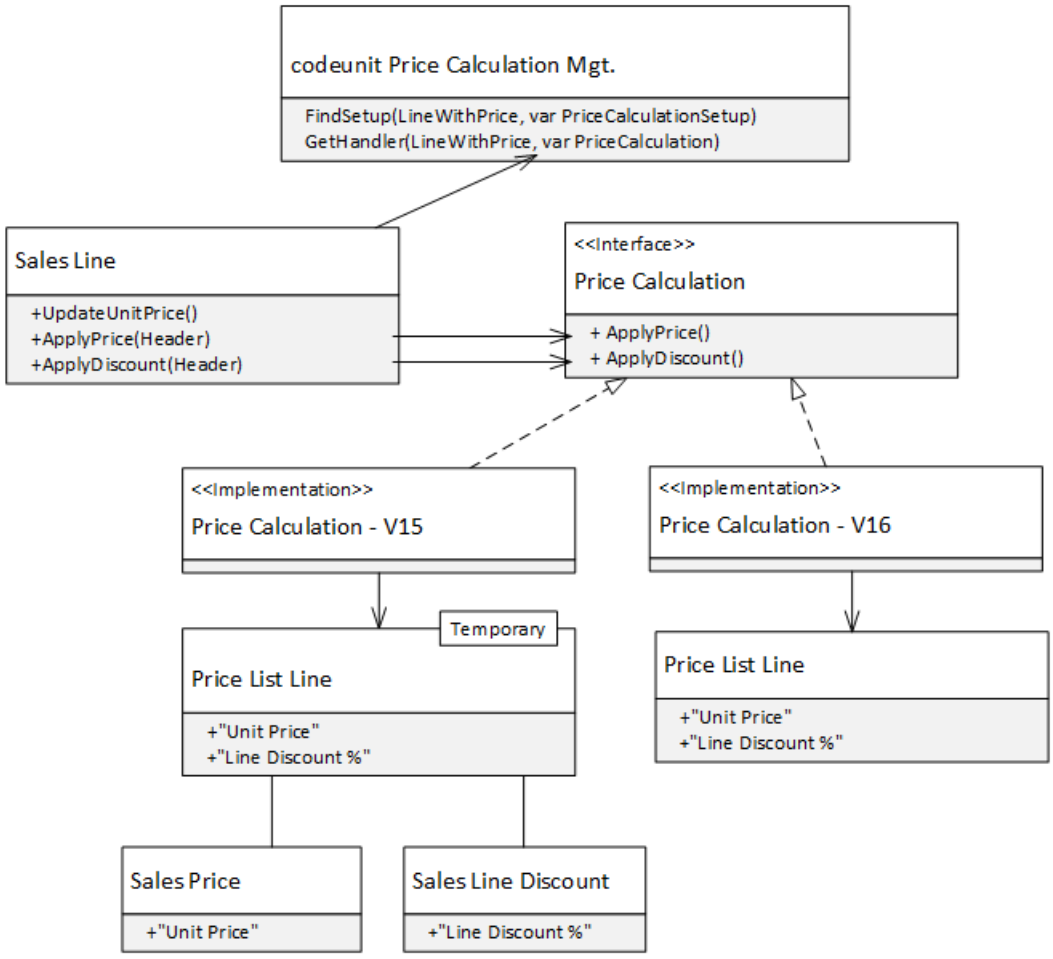
```
codeunit 50012 "Copy DocumentNo to Price List"
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::CopyFromToPriceListLine, 'OnCopyFromSalesPrice', '',
false, false)]
    procedure CopyFromSalesPriceHandler(var SalesPrice: Record "Sales Price"; var PriceListLine: Record
"Price List Line");
    begin
        PriceListLine."Document No." := SalesPrice."Document No.";
    end;
}
```

The Price Calculation Method field is added to all tables that need calculated prices and discounts:

- table 37 "Sales Line"
- table 5902 "Service Line"
- table 39 "Purchase Line"
- table 246 "Requisition Line"
- table 83 "Item Journal Line"
- table 753 "Standard Item Journal Line"

- table 210 "Job Journal Line"
- table 1003 "Job Planning Line"

The following image shows the schema of how the methods called in the Sales Line table get the price and discount amounts from either the Sales Price or Price List Line tables.



Interface Objects

AL interface objects are important for extensibility. They define the capabilities that are available to an object, and allow implementations to differ as long as they comply with the interface requirements. For more information, see Interfaces in AL.

Price calculation uses the following AL interface objects:

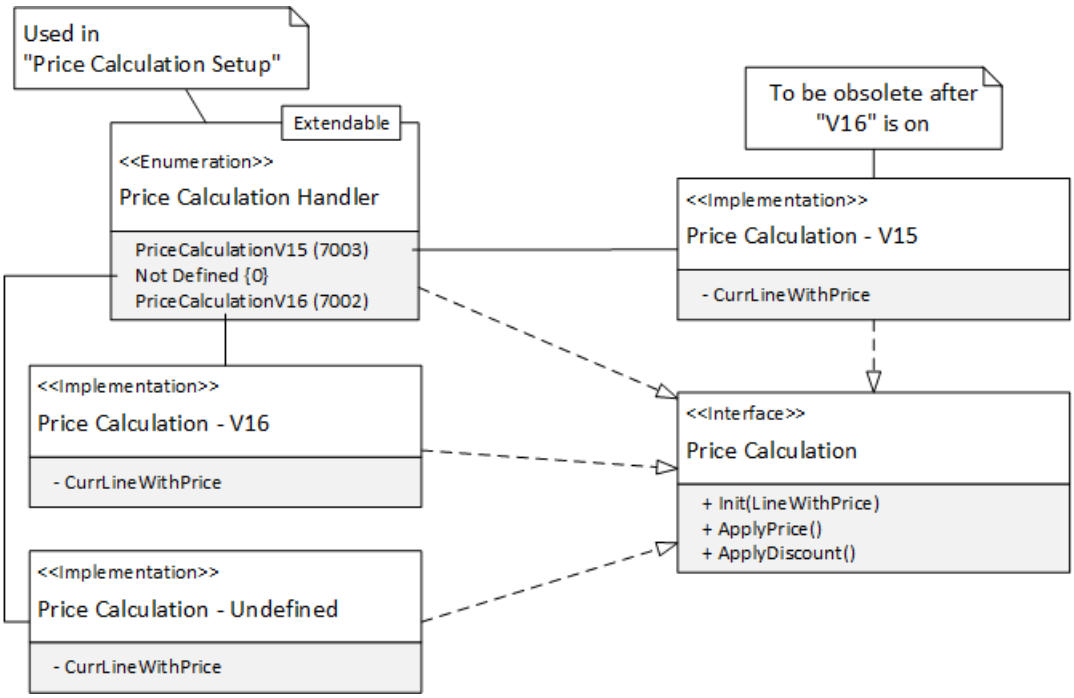
- Price Calculation
- Line With Price
- Price Source Group
- Price Source

Price Calculation

The Price Calculation interface defines methods that calculate amounts and discount percentages on journal and document lines.

In Business Central (Version 15.0) and earlier, the Price Calculation – V15 interface implementation calls the Sales Price Calc. Mgt. and Purch. Price Calc. Mgt. codeunits to calculate prices. In Business Central (Version 15.0), the implementation codeunit is Price Calculation – V16. This codeunit works the same as "Price Calculation – V15" but is based on a different price line table and makes it easier to extend price calculations.

The Price Calculation - Undefined implementation is used when the setup line does not contain a match for the document line. This implementation will display a message that states the combination that is missing.



You can add a new implementation codeunit or reuse one as a starting point and rewrite it as needed. For the new codeunit, you must extend the Price Calculation Handler enum that implements Price Calculation interface and is used in the Price Calculation Setup table.

```
enumextension 50000 "SpecialPriceHandler" extends "Price Calculation Handler"
{
    value(50000; "Special Price")
    {
        Implementation = "Price Calculation" = "Price Calc. - Special Price";
    }
}
```

Afterwards you can insert a record in the Price Calculation Setup table and set the new implementation as the default.

The following code in codeunit 7001 "Price Calculation Mgt." returns a Price Calculation interface that initialized with the instance of the Line With Price interface that depends on the setup record:

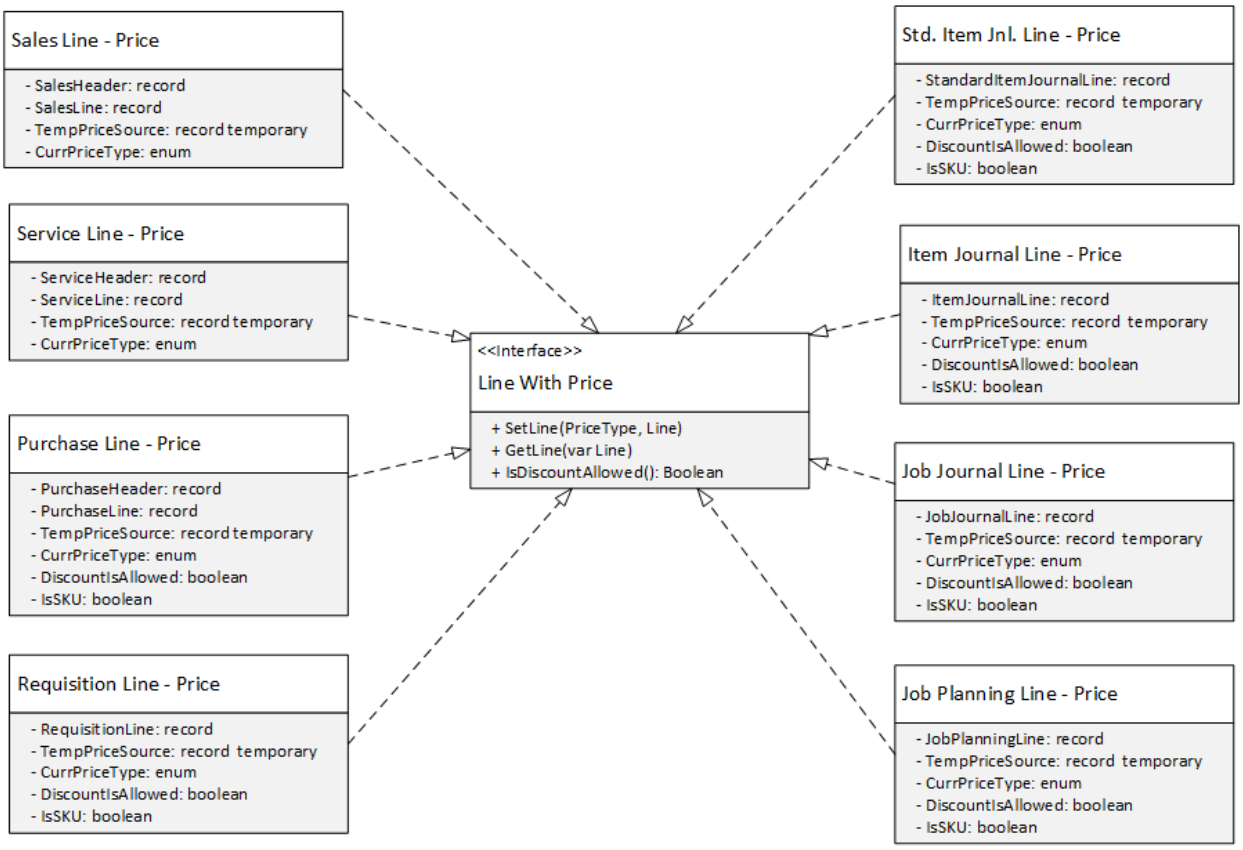
```
procedure GetHandler(
    LineWithPrice: Interface "Line With Price";
    var PriceCalculation: Interface "Price Calculation") Result: Boolean;
    var
        PriceCalculationSetup: Record "Price Calculation Setup";
    begin
        Result := FindSetup(LineWithPrice, PriceCalculationSetup);
        PriceCalculation := PriceCalculationSetup.Implementation;
        PriceCalculation.Init(LineWithPrice, PriceCalculationSetup);
    end;
```

After the price calculation implementation is defined the document line typically calls the methods in the interfrace that calculate the price and discount. The following code is used in the Sales Line table:

```
var
    Line: Variant;
begin
    PriceCalculation.ApplyDiscount();
    PriceCalculation.ApplyPrice(CalledByFieldNo);
    PriceCalculation.GetLine(Line);
    Rec := Line;
end;
```

Line With Price

The Line With Price interface defines methods for lines that require the calculation of a price, cost, and line discount. The following image shows the codeunits that implement this interface.



The following example shows a typical use of the codeunits in the Sales Line table.

```
 var
    PriceCalculationMgt: codeunit "Price Calculation Mgt.";
    PriceCalculation: Interface "Price Calculation";
    SalesLinePrice: Codeunit "Sales Line - Price";
    PriceType: Enum "Price Type";
begin
    SalesLinePrice.SetLine(PriceType::Sale, SalesHeader, Rec);
    PriceCalculationMgt.GetHandler(SalesLinePrice, PriceCalculation);
end;
```

The SalesLinePrice codeunit is declared directly because this is the sales line context. The instance is initialized by the interface's SetLine() method, and then passed to the GetHandler() method for PriceCalculation initialization because all Price Calculation implementation codeunits include an instance of the Line With Price interface, which stores data about document and journal lines. The following example shows how to declare the interface variable.

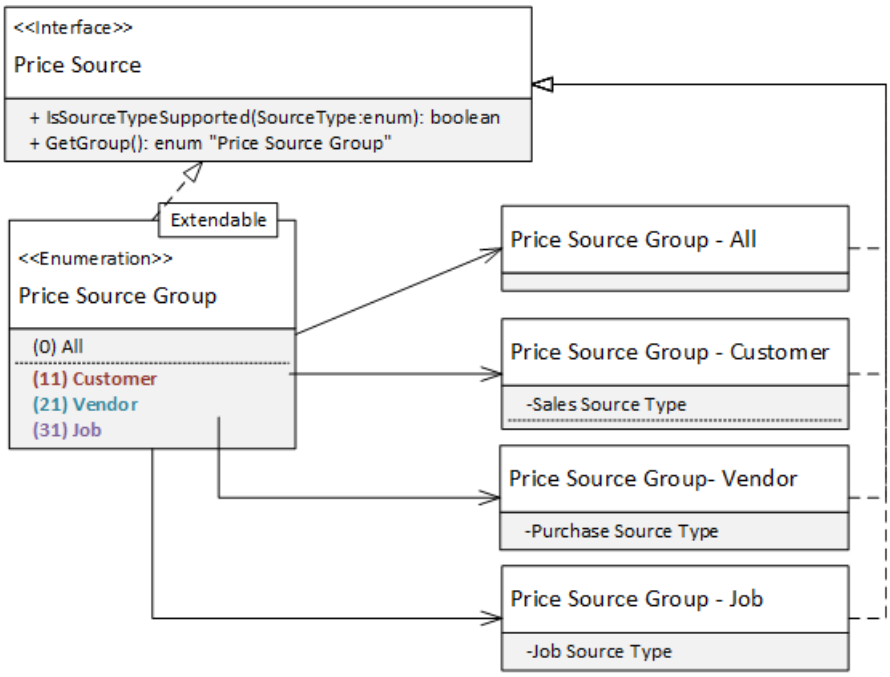
```
    var
        CurrLineWithPrice: Interface "Line With Price";
```

Price Source Group

The Price Source Group interface defines methods for a generic price source group. The Price Source Group enum defines the list of supported source groups, as follows:

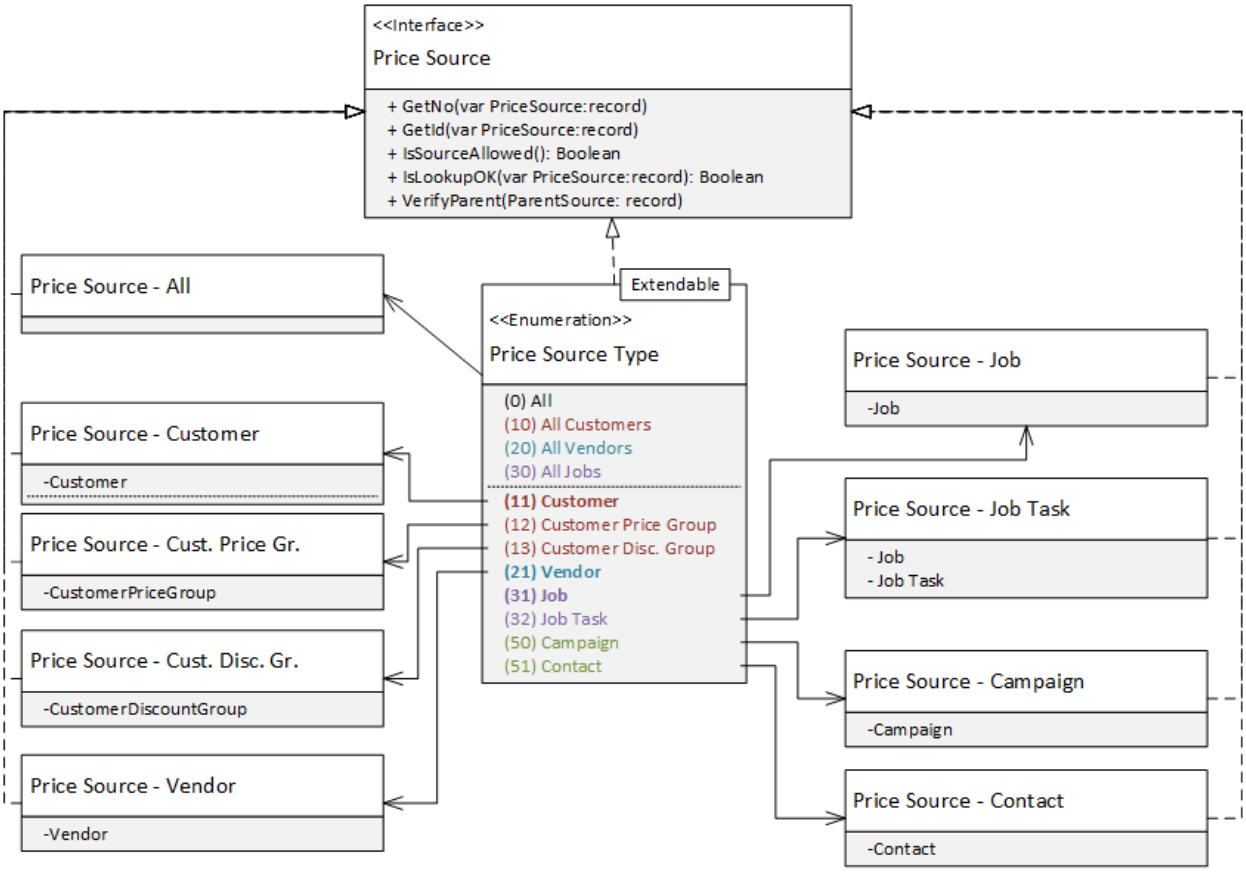
- All (0)
- Customer (11)
- Vendor (21)
- Job (31)

This enum is the subset of the Price Source Type enum. Both enums implement the Price Source Group interface. The interface helps to link the Price Source Type enum with the Sale Price Source Type, Purchase Price Source Type, and Job Price Source Type enums.



Price Source

The Price Source interface defines methods for price sources, such as vendors or customers. The list of supported sources is defined by the Price Source Type enum. The interface is used in the Price Source table to validate the primary key fields and look up respective tables, as shown in the following table.



Example

The Price Source enum implements the Price Source interface and defines Price Source - Customer as the implementation for the value Customer.

```
enum 7003 "Price Source Type" implements "Price Source", "Price Source Group"
    value(11; Customer)
    {
        Implementation = "Price Source" = "Price Source - Customer",
                   "Price Source Group" = "Price Source Group - Customer";
    }
```

The Price Source table has a public method LookupNo() that opens a different page depending on the Source Type value. The PriceSourceInterface variable gets the implementation value from the Source Type enum value and then calls the interface's IsLookupOK(Rec) method.

```
table 7005 "Price Source"
    var
        PriceSourceInterface: Interface "Price Source";

    procedure LookupNo() Result: Boolean;
    begin
        PriceSourceInterface := "Source Type";
        Result := PriceSourceInterface.IsLookupOK(Rec);
    end;
```

Because the source type Customer implementation is the "Price Source - Customer" codeunit, the interface calls its IsLookupOK() method and then opens the Customer List page.

```
codeunit 7032 "Price Source - Customer" implements "Price Source"

    procedure IsLookupOK(var PriceSource: Record "Price Source"): Boolean
    begin
        if Customer.Get(PriceSource."Source No.") then;
        if Page.RunModal(Page::"Customer List", Customer) = ACTION::LookupOK then begin
            PriceSource.Validate("Source No.", Customer."No.");
            exit(true);
        end;
    end;
```

Example of Extended Price Calculations

You can extend price calculations, for example, to include other sources or use calculations that allow for combinations and dependencies. The following sections provide an example.

Example: Change an Item Price When Combined with Another Item

Let's say we want to make the price of one item depend on whether it's sold individually or bundled with one or more other items. We'll use software licenses in this example, but the same principles apply in other scenarios.

> **NOTE**
> The prices, names, and combinations in this example are completely fictional and intended only to support the scenario described here. They do not reflect anything in the real-world.

We have the following licenses in our price list. If you buy 70061 or 70062 alone their prices do not change. However, let's say that we want to offer discounts when one license is purchased along with another. For example, we want to sell 70064 at a reduced monthly rate when it's purchased in combination with 70061 or 70062. Our list would then look like this.

ASSET NUMBER	NAME	MONTHLY BASE PRICE PER USER	BUNDLE PRICE
70061	BC Premium	$100	N/A
70062	BC Essentials	$75	N/A
70063	BC Team Member	$8	N/A
70064	Sales Professional	$65	$15 and $16 when bundled with 70061 or 70062.
70065	Customer Service Pro	$50	$20 and $21 when bundled with 70061 or 70062.

The following image shows an example of a Sales Line page that is extended with the Attached to Line No. field. Notice that the prices are changed based on the combinations of licenses.

Type	No.	Description	Location Code	Quantity	Attach To Line No.	Qty. to Assemble to Order	Reserved Quantity	Unit of Measure Code	Unit Price Excl. VAT
Item	70061	BC Premium	BLUE	2	0		_	PCS	100.00
Item	70062	BC Essential	BLUE	4	0		_	PCS	75.00
Item	70063	BC Team member	BLUE	10	0		_	PCS	8.00
Item	70064	Sales professional	BLUE	9	0		_	PCS	65.00
Item	70065	Customer Service Pro	BLUE	4	0		_	PCS	50.00
→ Item	70064	Sales professional	BLUE	2	10000		_	PCS	20.00
Item	70065	Customer Service Pro	BLUE	1	20000		_	PCS	16.00
Item	70064	Sales professional	BLUE	2	20000		_	PCS	21.00
Item	70065	Customer Service Pro	BLUE	4	10000		_	PCS	15.00

Let's look at some sample extensions that will implement this for us.

The first table extension adds a new field named Attach to Line No. to the Sales Line table and recalculates pricing when we make a change. This field will let us create the combinations that determine our discounts. It also copies the GetPriceCalculationHandler() function from the Sales Line table.

```
tableextension 50001 "Attach Price" extends "Sales Line"
{
    fields
    {
        field(50001; "Attach To Line No."; Integer)
        {
            trigger OnValidate()
            var
                PriceCalculation: Interface "Price Calculation";
            begin
                GetPriceCalculationHandler(PriceCalculation);
                ApplyPrice(FieldNo("Attach To Line No."), PriceCalculation);
            end;
        }
    }

    local procedure GetPriceCalculationHandler(var PriceCalculation: Interface "Price Calculation")
    var
        SalesHeader: Record "Sales Header";
        PriceCalculationMgt: codeunit "Price Calculation Mgt.";
        SalesLinePrice: Codeunit "Sales Line - Price";
        PriceType: Enum "Price Type";
    begin
        SalesHeader.Get("Document Type", "Document No.");
        SalesLinePrice.SetLine(PriceType::Sale, SalesHeader, Rec);
        PriceCalculationMgt.GetHandler(SalesLinePrice, PriceCalculation);
    end;
}
```

The following page extension adds the Attach Line No. field to the Sales order page (subform).

```
pageextension 50001 "Attach Price" extends "Sales Order Subform"
{
    layout
    {
        addAfter(Quantity)
        {
            field("Attach To Line No."; "Attach To Line No.")
            {
                ApplicationArea = Basic, Suite;
            }
        }
    }
}
```

The following table extension adds the Attach to Item No. field to the "Price List line" table.

```
tableextension 50002 "Attach To Price - Line" extends "Price List Line"
{
    fields
    {
        field(50001; "Attach To Item No."; Code[20])
        {
        }
    }
}
```

The following table extension adds the Attach to Item No. field to the "Price Calculation Buffer" table.

```
tableextension 50003 "Attach To Price - Buffer" extends "Price Calculation Buffer"
{
    fields
    {
        field(50001; "Attach To Item No."; Code[20])
        {
        }
    }
}
```

The calculation that links these three new fields is based on the following events:

- OnAfterSetFilters() – Sets the filter on the price list line when searching for the price.
- OnAfterFillBuffer() – Copies the value from the sales line to the buffer.
- FindItemToAttachToInLine() - Defines the value of the item number stored in the sales line that we attach to.

```
codeunit 50004 "Attached Price Mgt."
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Price Calculation Buffer Mgt.", 'OnAfterSetFilters',
'', false, false)]
    procedure OnAfterSetFilters(var PriceListLine: Record "Price List Line"; AmountType: Enum "Price Amount
Type"; var PriceCalculationBuffer: Record "Price Calculation Buffer"; ShowAll: Boolean);
    begin
        PriceListLine.SetRange("Attach To Item No.", PriceCalculationBuffer."Attach To Item No.");
    end;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Sales Line - Price", 'OnAfterFillBuffer', '', false,
false)]
    procedure OnAfterFillBuffer(var PriceCalculationBuffer: Record "Price Calculation Buffer"; SalesHeader:
Record "Sales Header"; SalesLine: Record "Sales Line");
    begin
        PriceCalculationBuffer."Attach To Item No." := FindItemToAttachToInLine(SalesLine);
    end;

    local procedure FindItemToAttachToInLine(CurrSalesLine: Record "Sales Line"): Code[20]
    var
        SalesLine: Record "Sales Line";
    begin
        if SalesLine.Get(CurrSalesLine."Document Type", CurrSalesLine."Document No.", CurrSalesLine."Attach
To Line No.") then
            exit(SalesLine."No.");
    end;
}
```

The new price calculation capabilities are not available in the user interface. When a page does become available, either, from Microsoft or one that you develop yourself, you can use the following sample code to

extend the page with a new control.

```
    field("Attach To Item No."; "Attach To Item No.")
    { }
```

See Also

Extending Application Areas

Extending Pages Previously Based on the Date Virtual Table

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Pages that previously had the **Date** virtual table as their source table have been redesigned so that they are based on buffer tables instead. This way, base application pages such as **Item Availability Lines** and **Res. Availability Lines** can now be extended.

To perform extensions on such pages, you must first extend the underlying buffer table. Then you create a method that calculates and updates the values of the extended fields and subscribe it to the **OnAfterCalcLine** event of the page.

Example

The following example illustrates how to add two new fields, `"Add. -Currency Debit Amount"` and `"Add. -Currency Credit Amount"`, to the **G/L Account Balance Lines** page, which used to be based on the **Date** table.

The first step is to create a Table Extension object and add the two fields, `"Add.-Currency Debit Amount"` and `"Add.-Currency Credit Amount"`, to the **G/L Account Balance Buffer** source table. Then you create a Page Extension object to display the fields in the **G/L Account Balance Lines** page. This is shown in the following code.

```
tableextension 50001 GLAccBalanceBufferExt extends "G/L Acc. Balance Buffer"
{
    fields
    {
        field(50001; "Add.-Currency Debit Amount"; Decimal)
        {

        }
        field(50002; "Add.-Currency Credit Amount"; Decimal)
        {

        }
    }
}

pageextension 50001 GLAccountBalanceLinesExt extends "G/L Account Balance Lines"
{
    layout
    {
        addlast(Control1)
        {

            field(AddCurrencyDebitAmount; "Add.-Currency Debit Amount")
            {

                ApplicationArea = Basic, Suite;
                Caption = 'Debit Amount (ACY)';
            }

            field(AddCurrencyCreditAmount; "Add.-Currency Credit Amount")
            {
                ApplicationArea = Basic, Suite;
                Caption = 'Credit Amount (ACY)';
            }
        }
    }
}
```

The subscriber method `GLAccountBalanceLinesOnAfterCalcLine` computes the values of the `"Add.-Currency Credit Amount"` and `"Add.-Currency Debit Amount"` fields and updates the **G/L Acc. Balance Buffer** table. This method is called when the **OnAfterCalcLine** event in the **G/L Account Balance Lines** page is raised.

```
codeunit 50001 GLAccountBalanceLinesExt
{
    // Subscribe to OnAfterCalcLine event
    [EventSubscriber(ObjectType::Page, Page::"G/L Account Balance Lines", 'OnAfterCalcLine', '', false,
false)]
    local procedure GLAccountBalanceLinesOnAfterCalcLine(var GLAccount: Record "G/L Account"; var
GLAccBalanceBuffer: Record "G/L Acc. Balance Buffer")
    begin

        // Calculate values
        GLAccount.CalcFields("Add.-Currency Credit Amount", "Add.-Currency Debit Amount");

        // Assign calculated values to the new fields of the buffer table
        GLAccBalanceBuffer."Add.-Currency Debit Amount" := GLAccount."Add.-Currency Debit Amount";
        GLAccBalanceBuffer."Add.-Currency Credit Amount" := GLAccount."Add.-Currency Credit Amount";

    end;
}
```

See Also

Page Extension object
Page object
CalcFields Method

Overview of the System Application

2/17/2021 • 3 minutes to read • Edit Online

The System Application contains modules that interact with the Dynamics 365 Business Central platform and online ecosystem to support the business logic in the Base Application. If you are developing extensions or add-ons for Dynamics 365 Business Central, you will probably need to use one or more of the objects in the modules.

This topic provides an overview of the modules in the System Application. For more details about each module, and to get a look at the code, choose the **ReadMe** link for the module to visit our AL AppExtensions repository on GitHub.

> **NOTE**
>
> The modules in the System Application represent a significant change in what's happening under the hood in Dynamics 365 Business Central. We are aware that the changes we have made will introduce breaking changes, so we have made a list of those that we know about, which includes suggestions for solutions. To view the breaking changes list, see Breaking Changes.
>
> We will continue to enhance the System Application in future releases. If you find something you think we should add, visit our Dynamics 365 Application Ideas page. If you want us to improve something, go to the AL AppExtensions repository and submit a pull request for it.

Overview of the Modules in the System Application

The list of modules is growing continuously. The following table lists and describes the modules that are available now.

MODULE	DESCRIPTION	LINK TO README
Assisted Setup	Contains all pages that are used by assisted setup guides.	ReadMe
Auto Format	Formats the appearance decimal data types.	ReadMe
Azure AD Graph	Interface for the Azure AD Graph API.	ReadMe
Azure AD Licensing	Access information about the subscribed SKUs and the corresponding service plans.	ReadMe
Azure AD Plan	Provides methods for retrieving and managing user plans in Azure Active Directory.	ReadMe
Azure AD Tenant	Retrieves information about the Azure Active Directory tenant.	ReadMe
Azure AD User Management	Provides functionality for managing Azure Active Directory users.	ReadMe

MODULE	DESCRIPTION	LINK TO README
Azure AD User	Retrieves and updates a user from the Azure AD Graph API.	ReadMe
Azure Key Vault	Stores Azure Key Vault secrets for deployments.	ReadMe
BLOB Storage	Stores and manages data in a binary format.	ReadMe
Base64 Convert	Converts text and from base 64.	ReadMe
Caption Class	Defines how the CaptionClass property displays captions.	ReadMe
Client Type Management	Allows testing of code that relies on different types of clients.	ReadMe
Confirm Management	Determines whether a confirm dialog displays when logic is run.	ReadMe
Cryptography Management	Contains the capabilities for encryption and hashing.	ReadMe
Cues and KPIs	Provides setup pages and interface methods to manage cues.	ReadMe
Data Classification	Handles data classification for objects that might contain sensitive information.	ReadMe
Data Compression	Compresses and uncompresses data in a binary format.	ReadMe
Date-Time Dialog	Provides helper functions for entering date-time values.	ReadMe
Default Role Center	Supports the default RoleCenter selection.	ReadMe
DotNet Aliases	Defines aliases for .NET classes.	ReadMe
Environment Information	Contains helper methods for getting information about the tenant and general settings.	ReadMe
Extension Management	Provides the tools needed to manage an extension.	ReadMe
Field Selection	Looks up fields.	ReadMe
Filter Tokens	Provides helper functions to manage filter texts.	ReadMe

MODULE	DESCRIPTION	LINK TO README
Headlines	Helps with constructing the text for headlines.	ReadMe
Language	Changes the language for Windows and applications.	ReadMe
Manual Setup	Contains functions and events used by manual setup pages.	ReadMe
Math	Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.	ReadMe
Object Selection	Look up page for all of the application objects, including objects from installed extensions.	ReadMe
Page Summary Provider	Contains functionality for providing summary data for a given page in Microsoft Teams.	ReadMe
Password	Sets and verifies passwords.	ReadMe
Printer Management	Contains functionality that enables a user to manage printers.	ReadMe
Record Link Management	Provides helper functions for RecordLinks.	ReadMe
Recurrence Schedule	Calculates when the next event will occur.	ReadMe
Satisfaction Survey	Shows a satisfaction survey.	ReadMe
Secrets	Contains secret providers for reading secrets from the key vault that is specified by an extension or from other secret providers.	ReadMe
Server Settings	Exposes methods that get settings from the server configuration file.	ReadMe
System Initialization	Runs non-business logic on user log-ins.	ReadMe
Tenant License State	Retrieves the current state of the tenant license.	ReadMe
Translation	Gets and stores language translations.	ReadMe
Upgrade Tags	Provides functionality for ensuring that the upgrade code is run only one time.	ReadMe

MODULE	DESCRIPTION	LINK TO README
User Log-In Times	Keeps track of when users sign in.	ReadMe
User Permissions	Exposes functionality to check and alter User Permission sets.	ReadMe
User Selection	Looks up and selects registered users.	ReadMe
Video	Looks up and plays videos.	ReadMe
Web Service Management	Provides the tools needed to manage web services.	ReadMe

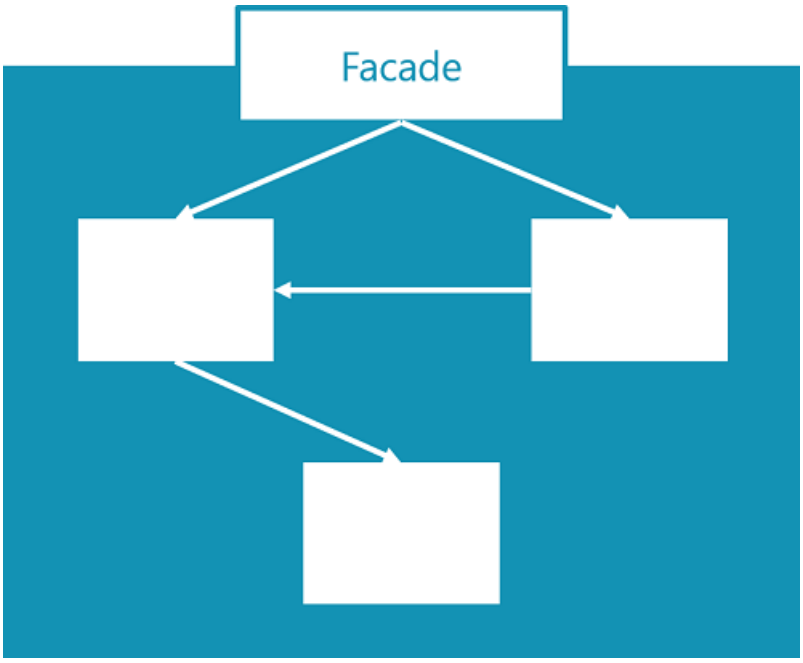
See Also

AL AppExtensions

Module Architecture

2/17/2021 • 5 minutes to read • Edit Online

Though the internal architecture of modules can, and most likely will, differ, there are rules that ensure consistency and reliability in the architecture of application modules. To reduce coupling and increase consistency, every module is a separate entity that has a publicly accessible facade, while the internal implementation is not public, as shown in the following image:



The following sections describe the design principles for modules.

One Module, One Project

Modules are independent projects and have their own app.json files. For details about the contents of the app.json file, see Project Setup.

Packaging and Layering

Modules can belong to a specific package, or functional layer. A module becomes part of a layer when it's added to the folder structure of the layer's package. A module can belong to several packages because the module is a package of its own, and can be bundled into larger functional layering packages.

Dependencies

Dependencies to other modules can only be taken to modules in the same functional layer or to lower layers. For example, a module in the Core Application can only take dependencies to modules in the Core Application or System Application, but never to extensions.

Target Environment

Every module must initially target the most restrictive environment, which is the cloud environment. If unsafe operations are required, those should result in a System Application module where the unsafe operations are wrapped with safe APIs. That is done by setting the **Target** to **OnPrem**. Modules in layers above the System Application must have **Target** set to **Cloud** in the **app.json** file.

Object Accessibility

Only facade codeunits, pages, and tables required in the public API of the module can be accessible. Internal implementation details must be marked as such by setting the access modifier to Internal, i.e. entities can be accessed within the module but cannot be called from outside the module.

Facade Codeunits

Every module must have a facade that meets the following rules:

- Access must be **Public**. It should be set explicitly to emphasize that this is a facade or an API-like codeunit that exposes the core functionality in the module.
- All integration and business event publishers must be in the facade as internal functions. This prevents them from being invoked outside the module.
- Event publishers should be marked as internal. Exceptions must be documented.
- All external methods go in the facade.
- Facades cannot contain logic or local functions.
- Because the facade is the codeunit that other modules reference, it should have a short, meaningful name. Internal implementation codeunits can be suffixed with "Impl," for example, because they are not referenced outside the module.

Business Logic

Implementation codeunits contain the business logic. Pages and tables can contain code, but should do so only when it's absolutely required.

Extensibility

Extensibility must be thought into the internal implementation of every module. If a module should not be extensible, the **Extensibility** property on tables, pages, and enums must be set to **False** to prevent those objects from being extended.

Functions and fields in extensible tables and pages must have an access modifier, as described in the following table.

ACCESS	DESCRIPTION
Local	Accessed only by code in the same table or table extension where the function or field is defined.
Internal	Accessed only by code in the same module, but not from another module.
Protected	Accessed only by code in the same table or table/page extensions of that table.
Public	Accessed by any other code in the same module and in other modules that reference it.

Documentation

Every publicly accessible object must be documented. For more information, see Documentation.

Tests

Every module's public API must be tested according to Microsoft standards. For more information, see Testing.

Single Instance

Use single instance only when it's required, or if the module is expected to be called frequently.

.NET

If a .NET type is an integral part of the module and is not referenced elsewhere, the alias for that type must be defined in the module. If the .NET type is used in different modules, then the alias for it must be defined in the DotNet Aliases module.

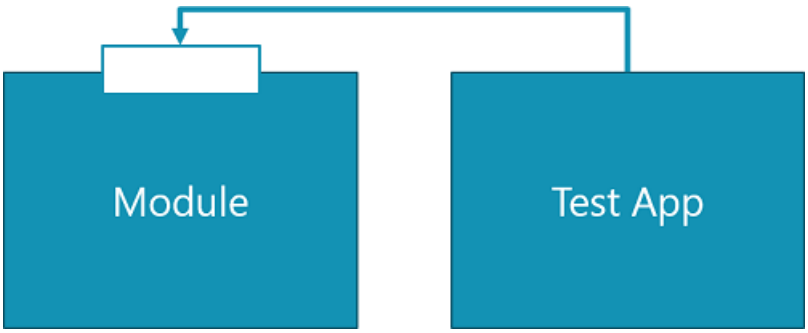
Project Setup

Every module begins with a project setup that includes the following:

- **Module Name**: If the module represents an entity, name the module after the entity. If the module does not represent an entity, give it a name that describes what it does. Module names can be singular or plural, depending on whether they handle one or more entities or tasks.
- **Location**: Determine which layer the module belongs to, and create a subfolder in a folder in that layer. For example, *Modules**System**My Module*).
- **Source Code**: Add the source code of the module in a src subfolder. For example, *Modules**System**My Module**src*).

Testing

Every module must be tested through a separate test module. Only public functions that are exposed through the facade, or other public objects such as pages, tables, xmlports, and queries, are tested by the test module.



The test module should have the same name as the module it tests, but be placed in a separate layer/package that contains tests for all modules in the layer. For example, *Modules****System Tests***My Module*). Test code cannot reside in the same layer folder structure as the module, or within the module, because it must not be executable or part of a production environment.

Documentation Guidelines

It's important that you document your module. The following sections provide some guidelines.

Modules

The app.json file for each module must contain a user-friendly title, a short brief, and a description that states its purpose.

Public API

Every publicly accessible object must have developer facing documentation that follows the XML documentation comments standard.

Public Objects

Public objects, such as codeunits, tables, pages, and so on, must have a summary that states what the object is used for.

```
/// <summary>
/// Lorem ipsum dolor sit amet, consectetur adipiscing elit.
/// </summary>
```

Public Functions

Public functions must have a summary, a parameter description, and a return value description. The following is an example.

```
/// <summary>
/// Maecenas sodales posuere ligula eu maximus.
/// </summary>
/// <param name="Lorem">The current language ID</param>
/// <param name="ipsum ">The CaptionClass expression to resolve</param>
/// <returns> Fusce in tristique massa, tincidunt tempor libero.</returns>
```

See Also

Getting Started with Modules
Create a New Module in the System Application
Module Architecture

Getting Started with Modules in the System Application

2/17/2021 • 2 minutes to read • Edit Online

This topic provides information about requirements for working with modules in the System Application.

AL

Get familiar with development in AL. For more information, see Getting Started with AL.

Git

Familiarize yourself with Git. For a quick introduction, see git - the simple guide.

Setup Environment

For details about how to set up an environment for AL development, see Set up your environment.

Building a new module

Have you decided to build a new module? For more information, see Create a new module.

Modifying an existing module

Want to improve an existing module? For more information, see Change a module.

Have an issue?

Please open an issue.

See Also

Module Architecture
Create a module

Set Up an Environment for Developing a Module

2/17/2021 • 3 minutes to read • Edit Online

This topic describes how to set up an environment for developing a module in the System Application.

Requirements

- You must have a GitHub account.
- You are familiar with the basics of Git, and have the application available. You will use Git to access the GitHub repository.
- Docker is installed. You will use it to run Business Central as a self-contained application.
- BcContainerHelper is installed. You will use it to create a Docker container.
- Demo or Partner license for Business Central

Get the Repository On Your Local Machine

1. Open the ALAppExtensions Repository, and choose Fork to create a fork of the repository.
2. Choose **Code**, and then copy the URL under code to clone the forked repository.

   The URL looks like https://github.com/<*username*>/ALAppExtensions.git, and <*username*> is your GitHub username.
3. Open PowerShell, and then open the directory in which you want to keep the repository files.
4. Run the **git clone** command. Replace <*URL*> with the URL you copied in step 3.
5. Open VS Code, and then go to the **System Modules** folder in the cloned repository.
6. Run the **code ALAppExtensions/Modules/System** command.
7. In the **System** folder, open the **app.json** and note the **version** that is listed. You will need that in step 3 in the process of setting up a Docker container.

Set Up a Business Central Docker Container

1. In PowerShell, import the **BcContainerHelper** module by running the **Import-Module BcContainerHelper** command. This loads the functions from the module.
2. Run the **New-BcContainerWizard** command. This opens a new PowerShell window and allows you to configure your AL Language development environment container.
3. Complete the steps in the wizard. Make the following changes to the default values.
   a. On the **Local Docker Container or Azure VM** step, choose a local docker container.
   b. On the **Authentication, Username/Password Authentication** step, choose options **a**, **b** or **c**.
   c. On the **Container Name** step, enter a name.
   d. On the **Version** step, choose options **e** or **f**, depending on whether you want a sandbox or on-premesis build.
      - Enter the version from `ALAppExtensions/Modules/System/app.json` after selecting option **e** or **f**.
   e. Country, US is fine.

f. Test Toolkit, if you have a partner license, you may choose any option. If you only have a demo license, choose option **d**, **No Test Toolkit**.

g. Premium Plan, not required.

h. Create Test Users, not required.

i. AL Base App Development, we will need this as we will work on the System Application.

j. Export AL Base App, not required as we will not be working on BaseApp.

k. AL Language Extension, not required.

l. License, put in the path to your license file.

m. Database, default option a.

n. Multitenant, non-multitenant setup.

o. We will then put in ! to use the default option choices for the rest of the options.

4. At the end of the configuration, you should be presented with the following script:

```
$containerName = 'mydemo'
$password = 'P@ssw0rd'
$securePassword = ConvertTo-SecureString -String $password -AsPlainText -Force
$credential = New-Object pscredential 'admin', $securePassword
$auth = 'UserPassword'
$artifactUrl = Get-BcArtifactUrl -type 'OnPrem' -country 'w1' -select 'Latest'
$licenseFile = '<Path to license file>'
New-BcContainer -accept_eula `
    -containerName $containerName `
    -credential $credential `
    -auth $auth `
    -artifactUrl $artifactUrl `
    -includeTestToolkit `
    -includeTestLibrariesOnly `
    -licenseFile $licenseFile `
    -includeAL -doNotExportObjectsToText `
    -updateHosts
```

> **NOTE**
>
> If you copy the script, remember to update the path to the license file ($licenseFile). Also, if you are using a demo license, remove the **-includeTestToolkit** and **-includeTestLibrariesOnly** options. 5. Run the script to create the Docker container. This can take some time if you are running it for the first time.

See Also

Getting Started with Modules
Create a New Module in the System Application
Module Architecture

Create a New Module in the System Application

2/17/2021 • 4 minutes to read • Edit Online

This topic provides an overview of how to create a new module in the System Application.

Requirements

1. Familiarity with development in AL. For more information, see AL Development.
2. Your development environment is ready. For more information, see Development Environment.

> **NOTE**
>
> Your environment must have the correct symbols. Go get those, in Visual Studio Code, press **F1**, and then choose **AL: Download Symbols**. Also, make a note of the **server** and **serverInstance** settings. You will add that information to the launch.json file.

Create a New Module

The following sections provide an example of how to contribute a new module. The example is based on the XmlWriter module, which is published in the AlAppExtensions repository. That contribution added a wrapper module to provide support for the XmlWriter, and the steps in this topic will recreate the pull request for the XmlWriter module. If you want to view the original pull request, it's available here: Pull Request 7876.

Set Up Visual Studio Code for Module Development

Open the **launch.json**, file and update the **server**, **serverInstance**, and **authentication** settings, as described in Set Up Your Development Environment.

```
"server": "https://YourDockerContainerName",
"serverInstance": "BC",
"authentication": "Windows",
```

Open the **settings.json** file, and update the **al.assemblyProbingPaths**, as described in Set Up a Development Environment.

Create a Branch

To create a branch, run the **git checkout -b "YourFeatureBranchName"** command. Afterward, you can start creating a new module.

Create a New Module

Before you create a new module, make sure you are familiar with the general architecture of system modules. For more information, see Module Architecture.

We'll start by creating a new folder named **XmlWriter** in the **System** folder, where we will add an app.json file. The app.json file will contain the general details of the module.

```
    {
        "id":  "215b484f-9fbf-437c-bc6e-67e2c0f283b0",
        "name":  "XMLWriter",
        "publisher":  "Microsoft",
        "brief":  "Write XML quickly with System.Xml.XmlTextWriter.",
        "description":  "Provides a fast, non-cached, forward-only way to create streams or files with XML
data that conforms to guidelines for W3C Extensible Markup Language (XML) 1.0 and Namespaces.",
        "version":  "17.0.0.0",
        "privacyStatement":  "https://go.microsoft.com/fwlink/?linkid=724009",
        "EULA":  "https://go.microsoft.com/fwlink/?linkid=2009120",
        "help":  "https://go.microsoft.com/fwlink/?linkid=2103698",
        "url":  "https://go.microsoft.com/fwlink/?linkid=724011",
        "logo":  "",
        "dependencies":  [
            {
                "id":  "7e3b999e-1182-45d2-8b82-d5127ddba9b2",
                "name":  "DotNet Aliases",
                "publisher":  "Microsoft",
                "version":  "17.0.0.0"
            }
        ],
        "screenshots":  [],
        "platform":  "17.0.0.0",
        "idRanges":  [
            {
                "from":  1483,
                "to":  1484
            }
        ],
        "target":  "OnPrem",
        "contextSensitiveHelpUrl":  "https://docs.microsoft.com/dynamics365/business-central/"
    }
```

NOTE

After we finish developing our module, we will need to update the app.json file to ensure that the **versions** and **idRanges** are correct. We can easily verify the version by checking the app.json in other modules in the System Application. The idRanges must reflect the IDs used in the module.

Next, create the **src** folder under **System/XmlWriter**. This folder will contain our source code. We will create an internal implementation codeunit named **XmlWriterImpl.Codeunit.al** in the **src** folder.

After adding the implementation functions, the implementation codeunit will look as follows.

```
codeunit 1484 "XmlWriter Impl"
{
    Access = Internal;

    procedure WriteStartDocument()
    begin
        StringBuilder := StringBuilder.StringBuilder();
        StringWriter := StringWriter.StringWriter(StringBuilder);
        XmlTextWriter := XmlTextWriter.XmlTextWriter(StringWriter);
        XmlTextWriter.WriteStartDocument();
    end;

    procedure WriteEndDocument()
    begin
        XmlTextWriter.WriteEndDocument();
    end;

    procedure ToBigText(var XmlBigText: BigText)
    begin
        XmlTextWriter.WriteString(XmlBigText);
    end;

    var
        StringBuilder: DotNet StringBuilder;
        StringWriter: DotNet StringWriter;
        XmlTextWriter: DotNet XmlTextWriter;
}
```

Now that we have created our implementation codeunit, we must add public functions in a facade codeunit with the functionality that we want to expose. Because the functions are public, we must ensure that these are tested and documented. To do this, we need to create a facade codeunit. In this example, we'll name the codeunit **XmlWriter.Codeunit.al**. The functions call the corresponding functions in the implementation codeunit.

The following is an example of the facade codeunit named **XmlWriter.Codeunit.al** that includes public functions and documentation.

```
/// <summary>
/// Provides helper functions for System.Xml.XmlWriter
/// </summary>
codeunit 1483 "XmlWriter"
{
    Access = Public;

    /// <summary>
    /// Creates the XmlWriter Document
    /// </summary>
    procedure WriteStartDocument()
    begin
        XmlWriterImpl.WriteStartDocument();
    end;

    /// <summary>
    /// Closes any open elements or attributes and puts the writer back in the Start state.
    /// </summary>
    procedure WriteEndDocument()
    begin
        XmlWriterImpl.WriteEndDocument();
    end;

    /// <summary>
    /// Writes the text within XmlWriter to the BigText variable.
    /// </summary>
    /// <param name="XmlBigText">The BigText the XmlWriter has to be write to.</param>
    procedure ToBigText(var XmlBigText: BigText)
    begin
        Clear(XmlBigText);
        XmlWriterImpl.ToBigText(XmlBigText)
    end;

    var
        XmlWriterImpl: Codeunit "XmlWriter Impl";
}
```

Now that we have now exposed the functions, the next step is to add tests. To do that, we'll create a new folder under **System Tests**, **XmlWriter**. We will also create an app.json file for the module details and an **src** folder for our test code.

We will add the following new file under **System Tests/XmlWriter/src**, **XmlWriterTest.Codeunit.al**.

```
    codeunit 139911 "Xml Writer Test"
    {
        Subtype = Test;

        var
            Assert: Codeunit "Library Assert";

        [Test]
        procedure TestWriteStartDocument()
        var
            XmlWriter: Codeunit "XmlWriter";
            XmlBigText: BigText;
        begin
            // [GIVEN] Initialized XmlWriter
            XmlWriter.WriteStartDocument();

            // [THEN] Get XmlDocument to text
            XmlWriter.ToBigText(XmlBigText);
            Assert.AreEqual('<?xml version="1.0" encoding="utf-16"?>', Format(XmlBigText), 'Unexpected text
when creating a Xml Document with XmlWriter');
        end;

        [Test]
        procedure TestWriteEndDocumentNoElement()
        var
            XmlWriter: Codeunit "XmlWriter";
        begin
            // [GIVEN] Initialized XmlWriter with root element
            XmlWriter.WriteStartDocument();

            // [WHEN] Write end document
            // [THEN] Expected error to be thrown
            asserterror XmlWriter.WriteEndDocument();
            Assert.ExpectedError('A call to System.Xml.XmlTextWriter.WriteEndDocument failed with this
message: Document does not have a root element.');
        end;
    }
```

After running the tests successfully, changes are complete.

Commit and push your changes and open a PR

- To commit your changes, run the **git commit -m "Your message"** command.
- To push your changes, run the **git push** command.

You can now go to your Github fork and open a pull request in the AlAppExtensions repository.

See Also

Create a .NET Wrapper Module
Become a Contributor to Business Central
"Git" going with extensions
Walkthrough: Contributing to an extension on GitHub
Getting Started with Modules
Create a New Module in the System Application
Module Architecture

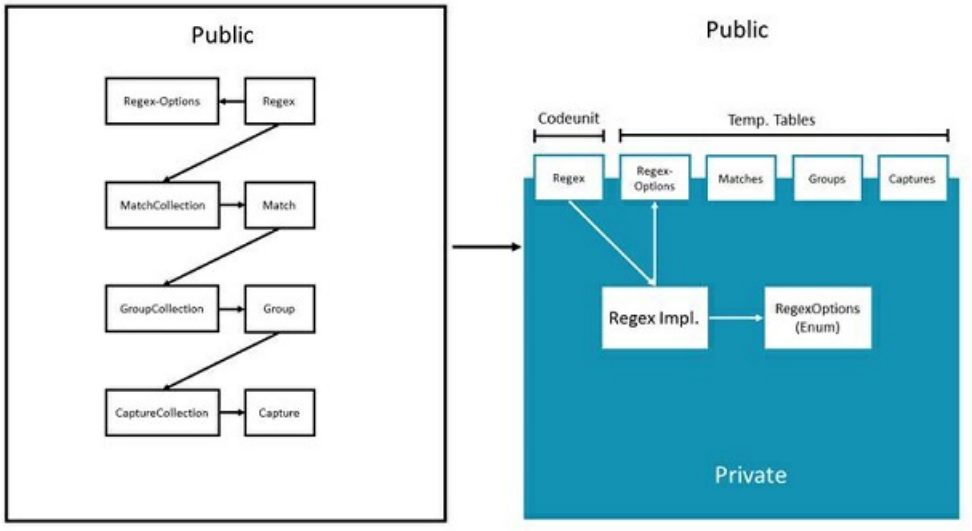
Create a .NET Wrapper Module

2/17/2021 • 4 minutes to read • Edit Online

This topic provides a description of how to contribute a .NET wrapper module to Business Central, using the Regex module as an example. The Regex module is published in the AlAppExtensions repository, and if you aren't already familiar with the Regex class in .NET, see the .NET documentation.

Converting the Regex architecture

In .NET, the Regex class provides functionality for regular expressions. You can implement the functionality in a similar way as a system module by creating a Regex codeunit that provides an interface and an internal Regex implementation that contains the logic.



Wrapping a .NET method

For some methods, you can wrap the .NET method in AL in an internal codeunit and expose the procedure to the facade. For example, you can wrap the IsMatch method as follows.

```
/// <summary>
/// Provides functionality to use regular expressions to match text, split text, replace text etc.
/// </summary>
codeunit 3960 Regex
{
    Access = Public;

    var
        RegexImpl: Codeunit "Regex Impl.";

    /// <summary>
    /// Indicates whether the regular expression finds a match in the input string.
    /// </summary>
    /// <param name="Input">The string to search for a match.</param>
    /// <param name="Pattern">A regular expression pattern to match.</param>
    /// <returns>True if the regular expression finds a match; otherwise, false.</returns>
    procedure IsMatch(Input: Text; Pattern: Text): Boolean
    begin
        exit(RegexImpl.IsMatch(Input, Pattern));
    end;
}
```

```
codeunit 3961 "Regex Impl."
{
    Access = Internal;

    var
        DotNetRegex: DotNet Regex;

    procedure IsMatch(Input: Text; Pattern: Text): Boolean
    begin
        Regex(Pattern);
        exit(DotNetRegex.IsMatch(Input))
    end;
}
```

Codeunits in method signatures

Not all .NET Regex classes map so directly to system modules, however. The .NET Regex class also includes classes such as Match, Group, and Capture, that are used to represent results for regular expression matches. While it's tempting to wrap these classes in codeunits and use those to output results, you should avoid that because procedure signatures should not contain codeunits. Instead, use temporary tables to model these classes. The following code example shows how to implement the Match class in a temporary table.

```
/// <summary>
/// Provides a representation of Regex Matches that models Match objects in .Net
/// </summary>



/// <remark>
/// For more information, visit https://docs.microsoft.com/en-
us/dotnet/api/system.text.regularexpressions.match?view=netcore-3.1.
/// </remark>
table 3965 Matches
{
    TableType = Temporary;
    Extensible = false;

    fields
    {
        field(1; MatchIndex; Integer)
        {
            Description = 'The index of the match in the table.';
        }
        field(2; Index; Integer)
        {
            Description = 'The position in the original string where the first character of the captured
substring is found.';
        }
        field(3; ValueBlob; Blob)
        {
            Access = Internal;
            Description = 'Gets the captured substring from the input string.';
        }
        field(4; Length; Integer)
        {
            Description = 'Gets the length of the captured substring.';
        }
        field(5; Success; Boolean)
        {
            Description = 'Gets a value indicating whether the match is successful.';
        }
    }
}
```

The temporary table has all the normal table procedures, and can be extended with procedures if needed. Now we can write the Match objects, output by .NET, to this table.

```
codeunit 3961 "Regex Impl."
{
    Access = Internal;

    var
        DotNetRegex: DotNet Regex;
        DotNetMatchCollection: DotNet MatchCollection;

    procedure Match(Input: Text; Pattern: Text; var Matches: Record Matches)
    begin
        Regex(Pattern);
        DotNetMatchCollection := DotNetRegex.Matches(Input);
        InsertMatch(Matches);
    end;
}
```

Avoiding Constructors

The .NET Regex class includes multiple constructors, but we should not expose them in the facade. We can, however, use constructors internally, as the code previous code example shows.

Removing constructors from a class can cause overloads. One way to get around that is to use the argument-table pattern. For example, you can construct a temporary table with all optional parameters and internally implement the logic to apply them.

The .NET Regex class contains three constructors that we want to support:

- Regex(Pattern)
- Regex(Pattern, RegexOptions)
- Regex(Pattern, RegexOptions, MatchTimeout)

In .NET, RegexOptions is an enum with options for matching the pattern (case-sensitivity, ignoring whitespace, and so on), and MatchTimeout sets a time-out interval for matching. The following example shows how to add those options to a temporary table.

```
/// <summary>
/// Table with options to use with Regular Expressions
/// </summary>
table 3966 "Regex Options"
{
    Extensible = false;
    TableType = Temporary;

    fields
    {
        field(1; IgnoreCase; Boolean)
        {
            DataClassification = SystemMetadata;
            Description = 'Specifies case-insensitive matching.';
        }
        field(2; Multiline; Boolean)
        {
            DataClassification = SystemMetadata;
            Description = 'Changes the meaning of ^ and $ so they match at the beginning and end
respectively of any line.';
        }

        ...

        field(10; MatchTimeoutInMs; Integer)
        {
            DataClassification = SystemMetadata;
            Description = 'A time-out interval in milliseconds, to indicate when the matching should time
out.';
            InitValue = -1; // Indicates no time-out
        }
    }
}
```

This temporary table makes it straightforward to add overloads to the facade. For each method, there is one procedure without an options parameter, and one overload with an options table. The following example illustrates that for the Match procedure.

```
/// <summary>
    /// Searches the input string for the first occurrence of the specified regular expression, using the
specified matching options.
    /// </summary>
    /// <param name="Input">The string to search for a match.</param>
    /// <param name="Pattern">A regular expression pattern to match.</param>
    /// <param name="Match">The Match object to write information about the match to.</param>
    procedure Match(Input: Text; Pattern: Text; var Matches: Record Matches)
    begin
        RegexImpl.Match(Input, Pattern, Matches);
    end;

    /// <summary>
    /// Searches the input string for the first occurrence of the specified regular expression, using the
specified matching options.
    /// </summary>
    /// <param name="Input">The string to search for a match.</param>
    /// <param name="Pattern">A regular expression pattern to match.</param>
    /// <param name="RegexOptions">A combination of the enumeration values that provide options for
matching.</param>
    /// <param name="Match">The Match object to write information about the match to.</param>
    procedure Match(Input: Text; Pattern: Text; var RegexOptions: Record "Regex Options"; var Matches:
Record Matches)
    begin
        RegexImpl.Match(Input, Pattern, RegexOptions, Matches);
    end;
```

See Also

Module Architecture
Getting Started with Modules in the System Application
Set Up an Environment for Developing a Module
Create a New Module in the System Application
Change a Module in the System Application

Change a Module in the System Application

2/17/2021 • 5 minutes to read • Edit Online

This topic provides an overview of how to change an existing module.

Requirements

1. Familiarity with development in AL. For more information, see AL Development.
2. Your local repository and development environment are ready. For more information, see Set Up an Environment for Developing a Module.

> **NOTE**
>
> Your environment must have the correct symbols. Go get those, in Visual Studio Code, press **F1**, and then choose **AL: Download Symbols**. Also, make a note of the **server** and **serverInstance** settings. You will add that information to the launch.json file.

Your changes must follow the guidelines for module architecture. For more information, see Module Architecture. When changing an existing module, do not introduce breaking changes, that is, make sure that you do not break existing functionality. Existing tests must still pass, and you should add new tests for the functionality that you change or add.

Set Up Visual Studio Code for Module Development

Open the **launch.json**, file and update the **server**, **serverInstance**, and **authentication** settings, as described in Set Up Your Development Environment.

```
    "server": "https://YourDockerContainerName",
    "serverInstance": "BC",
    "authentication": "Windows",
```

Open the **settings.json** file, and update the **al.assemblyProbingPaths**, as described in Set Up Your Development Environment.

Create a Branch

To create a branch, run the **git checkout -b "YourFeatureBranchName"** command. Afterward, you can start creating a new module.

Change an Existing Module

The following sections provide an example of how to contribute to an existing module. The example is based on a previous contribution to the Base64 Convert module, which has been published in the AlAppExtensions repository. The contribution added support for text encodings other than UTF8. If you're interested, you can view the original pull request at Pull Request 7676.

Make Changes to a Module

Before making changes, make sure you are familiar with the general architecture of system modules. For more information, see Module Architecture. You can also check out the article titled How to add a system module for an example of creating a full system module.

We'll start by adding the functions that we need to support different text encodings to the internal

implementation codeunit. We'll add the following functions to the **System/Base64 Convert/src/Base64ConvertImpl.Codeunit.al** implementation codeunit:

```
    procedure ToBase64(String: Text; TextEncoding: TextEncoding): Text
    begin
        exit(ToBase64(String, false, TextEncoding, 0));
    end;

    procedure ToBase64(String: Text; TextEncoding: TextEncoding; Codepage: Integer): Text
    begin
        exit(ToBase64(String, false, TextEncoding, Codepage));
    end;

    procedure ToBase64(String: Text; InsertLineBreaks: Boolean; TextEncoding: TextEncoding): Text
    begin
        exit(ToBase64(String, InsertLineBreaks, TextEncoding, 0));
    end;

    procedure ToBase64(String: Text; InsertLineBreaks: Boolean; TextEncoding: TextEncoding; Codepage:
Integer): Text
    var
        Convert: DotNet Convert;
        Encoding: DotNet Encoding;
        Base64FormattingOptions: DotNet Base64FormattingOptions;
        Base64String: Text;
    begin
        if String = '' then
            exit('');

        if InsertLineBreaks then
            Base64FormattingOptions := Base64FormattingOptions.InsertLineBreaks
        else
            Base64FormattingOptions := Base64FormattingOptions.None;
        case TextEncoding of
            TextEncoding::UTF16:
                Base64String := Convert.ToBase64String(Encoding.Unicode().GetBytes(String),
Base64FormattingOptions);
            TextEncoding::MSDos,
            TextEncoding::Windows:
                Base64String := Convert.ToBase64String(Encoding.GetEncoding(CodePage).GetBytes(String),
Base64FormattingOptions);
            else
                Base64String := Convert.ToBase64String(Encoding.UTF8().GetBytes(String),
Base64FormattingOptions);
        end;

        exit(Base64String);
    end;

    procedure FromBase64(Base64String: Text; TextEncoding: TextEncoding): Text
    begin
        exit(FromBase64(Base64String, TextEncoding, 1252));
    end;

    procedure FromBase64(Base64String: Text; TextEncoding: TextEncoding; CodePage: Integer): Text
    var
        Convert: DotNet Convert;
        Encoding: DotNet Encoding;
        OutputString: Text;
    begin
        if Base64String = '' then
            exit('');

        case TextEncoding of
            TextEncoding::UTF16:
                OutputString := Encoding.Unicode().GetString(Convert.FromBase64String(Base64String));
            TextEncoding::MSDos,
            TextEncoding::Windows:
```

```
            TextEncoding::Windows:
                OutputString :=
Encoding.GetEncoding(CodePage).GetString(Convert.FromBase64String(Base64String));
            else
                OutputString := Encoding.UTF8().GetString(Convert.FromBase64String(Base64String));
        end;
        exit(OutputString);
    end;
```

We also need to update some of the existing functions in **System/Base64 Convert/src/Base64ConvertImpl.Codeunit.al**, while making sure that they keep the same functionality:

```
    procedure ToBase64(String: Text): Text
    begin
        exit(ToBase64(String, false));
    end;

    procedure ToBase64(String: Text; InsertLineBreaks: Boolean): Text
    begin
        exit(ToBase64(String, InsertLineBreaks, TextEncoding::UTF8, 0));
    end;

    procedure FromBase64(Base64String: Text): Text
    begin
        exit(FromBase64(Base64String, TextEncoding::UTF8, 0));
    end;
```

We have changed the implementation codeunit, and avoided breaking existing functionality by keeping the same behavior for existing functions.

Now we'll add public functions in the facade codeunit with the functionality that we want to expose. Because the functions are public, we need to ensure that they are documented and tested. The functions call the corresponding functions in the implementation codeunit. We add the following functions to **System/Base64 Convert/src/Base64Convert.Codeunit.al**:

```
    /// <summary>
    /// Converts the value of the input string to its equivalent string representation that is encoded with
base-64 digits.
    /// </summary>
    /// <param name="String">The string to convert.</param>
    /// <param name="TextEncoding">The TextEncoding for the input string.</param>
    /// <returns>The string representation, in base-64, of the input string.</returns>
    procedure ToBase64(String: Text; TextEncoding: TextEncoding): Text
    begin
        exit(Base64ConvertImpl.ToBase64(String, TextEncoding));
    end;

    /// <summary>
    /// Converts the value of the input string to its equivalent string representation that is encoded with
base-64 digits.
    /// </summary>
    /// <param name="String">The string to convert.</param>
    /// <param name="TextEncoding">The TextEncoding for the input string.</param>
    /// <param name="Codepage">The Codepage if TextEncoding is MsDos or Windows.</param>
    /// <returns>The string representation, in base-64, of the input string.</returns>
    procedure ToBase64(String: Text; TextEncoding: TextEncoding; Codepage: Integer): Text
    begin
        exit(Base64ConvertImpl.ToBase64(String, TextEncoding, Codepage));
    end;

    /// <summary>
    /// Converts the specified string, which encodes binary data as base-64 digits, to an equivalent regular
string.
    /// </summary>
    /// <param name="Base64String">The string to convert.</param>
    /// <param name="TextEncoding">The TextEncoding for the input string.</param>
    /// <returns>Regular string that is equivalent to the input base-64 string.</returns>
    /// <error>The length of Base64String, ignoring white-space characters, is not zero or a multiple of 4.
</error>
    /// <error>The format of Base64String is invalid. Base64String contains a non-base-64 character, more
than two padding characters,
    /// or a non-white space-character among the padding characters.</error>
    procedure FromBase64(Base64String: Text; TextEncoding: TextEncoding): Text
    begin
        exit(Base64ConvertImpl.FromBase64(Base64String, TextEncoding));
    end;

    /// <summary>
    /// Converts the specified string, which encodes binary data as base-64 digits, to an equivalent regular
string.
    /// </summary>
    /// <param name="Base64String">The string to convert.</param>
    /// <param name="TextEncoding">The TextEncoding for the inout string.</param>
    /// <param name="Codepage">The Codepage if TextEncoding is MsDos or Windows.</param>
    /// <returns>Regular string that is equivalent to the input base-64 string.</returns>
    /// <error>The length of Base64String, ignoring white-space characters, is not zero or a multiple of 4.
</error>
    /// <error>The format of Base64String is invalid. Base64String contains a non-base-64 character, more
than two padding characters,
    /// or a non-white space-character among the padding characters.</error>
    procedure FromBase64(Base64String: Text; TextEncoding: TextEncoding; Codepage: Integer): Text
    begin
        exit(Base64ConvertImpl.FromBase64(Base64String, TextEncoding, Codepage));
    end;
```

We have now exposed the functions. The next steps are to ensure that existing tests pass, and then add new tests for the functionality that we added.

After verifying that the tests pass, we'll add the following tests to the **System Tests/Base64 Convert/src/Base64ConvertTest.Codeunit.al** file.

```
    [Test]
    procedure StringToBase64UTF16Test()
    var
        ConvertedText: Text;
    begin
        // [SCENARIO] A string variable is converted to base-64 string, binary representation is kept in
UTF16

        // [WHEN] The string is converted
        ConvertedText := Base64Convert.ToBase64(SampleUTF16Txt, TextEncoding::UTF16);

        // [THEN] The converted value is correct
        Assert.AreEqual(Base64SampleUTF16Txt, ConvertedText, ConvertionToBase64UTF16Err);
    end;

    [Test]
    procedure FromBase64UTF16StringTest()
    var
        ConvertedText: Text;
    begin
        // [SCENARIO] A base-64 string with UTF16 encoding is converted to a regular string

        // [WHEN] The base-64 string is converted
        ConvertedText := Base64Convert.FromBase64(Base64SampleUTF16Txt, TextEncoding::UTF16);

        // [THEN] The converted value is correct
        Assert.AreEqual(SampleUTF16Txt, ConvertedText, ConvertionFromBase64UTF16Err);
    end;
```

Commit and Push Changes, and Open a Pull Request

To submit your changes, follow these steps:

1. To commit your changes, run the **git commit -m "Your message"** command.
2. To push your changes, run the **git push** command.

You can now go to your GitHub fork and open a pull request in the AlAppExtensions repository.

See Also

Become a contributor Git going with extensions Walkthrough: Contributing to an extension on GitHub

Deprecating Explicit and Implicit With Statements

2/17/2021 • 6 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

The extensibility model and the AL programming language is a successor to the C/AL language. And the `with` statement has up until now been supported in AL. While using the `with` statement might make code harder to read, it can also prevent code in Business Central online from being upgraded without changes to the code or even worse - upgraded, but with changed behavior. We distinguish between two types of with statements; the explicit type of `with` using the keyword, and the implicit with which is not expressed directly in code. The next sections describe these one by one.

The explicit with statement

In Business Central online your code is recompiled when the platform and application versions are upgraded. The recompilation ensures that it is working and the recompile regenerates the runtime artifacts to match the new platform. Breaking changes without due warning are not allowed, but the use of the `with` statement makes it impossible, as Microsoft, to make even additive changes in a completely non-breaking way. This problem is not alone isolated to changes made by Microsoft; any additive change has the potential to break a `with` statement in consuming code.

The following example illustrates code written using the `with` statement; referred to in this context as the explicit `with`.

```
codeunit 50140 MyCodeunit
{
    procedure DoStuff()
    var
        Customer: Record Customer;
    begin
        with Customer do begin
            // Do some work on the Customer record.
            Name := 'Foo';

            if IsDirty() then
                Modify();
        end;
    end;

    local procedure IsDirty(): Boolean;
    begin
        exit(false);
    end;
}
```

The `DoStuff()` procedure processes work on the Customer record and calls a local procedure `IsDirty()` to check whether to update the record or not. Looking at the code above it looks like it does nothing since `IsDirty()` returns `false` - assuming that the `IsDirty()` call (line 11) is in fact calling the local `IsDirty()` procedure.

Symbol lookup

Considering the above code sample again, what would happen to that code if `IsDirty` was added to the base

application between two releases? To understand that we need to take a look at how compilers turn syntax into symbols. When the AL compiler meets the `IsDirty` call it must bind the syntax name to a procedure symbol.

When the AL compiler searches for the symbol `IsDirty()` in the sample above it will search in following order:

1. Customer table
   - User-defined members on the Customer table and Customer table extensions
   - Platform-defined members, for example, `FindFirst()` or `Modify()`
2. MyCodeunit codeunit
   - User-defined members, for example, `IsDirty()`
   - Platform-defined members
3. Globally defined members

The first time the search for `IsDirty` finds the name IsDirty, it will not continue to the next top-level group. That means that if a procedure named `IsDirty` is introduced in the Customer table (platform or application) that procedure will be found instead of the procedure in `MyCodeunit`.

The solution for the *explicit* `with` is to stop using it. Using the `with` statement can make your code vulnerable to upstream changes. The example below illustrates how to rewrite the sample in The explicit with statement:

```
// Safe version
codeunit 50140 MyCodeunit
{
    procedure DoStuff()
    var
        Customer: Record Customer;
    begin
        // Do some work on the Customer record.
        Customer.Name := 'Foo';

        if IsDirty() then
            Customer.Modify();
    end;

    local procedure IsDirty(): Boolean;
    begin
        exit(false);
    end;
}
```

The implicit with statement

The implicit with is injected automatically by the compiler in certain situations. The next sections describe, how this works on codeunits and pages.

Codeunits

When a codeunit has the `TableNo` property set, there is an implicit with around the code inside the `OnRun` trigger. This is indicated with the comments in the code example below.

```
codeunit 50140 MyCodeunit
{
    TableNo = Customer;

    trigger OnRun()
    begin
        // with Rec do begin
        SetRange("No.", '10000', '20000');
        if Find() then
            repeat
            until Next() = 0;

        if IsDirty() then
            Error('Something is not clean');
        // end;
    end;

    local procedure IsDirty(): Boolean;
    begin
        exit(false);
    end;
}
```

Similar to the The explicit with statement, the code looks like it will call the local `IsDirty` method, but depending on the Customer table, extensions to the Customer table, and built-in methods that may not be the case. If any of these implement an `IsDirty` method with an identical signature that returns `true`, then the example above will fail with an error. If an `IsDirty` method with a different signature is implemented, then this code will not compile and will fail to upgrade.

Pages

Pages on tables have the same type of implicit with, but around the entire object. Everywhere inside the page object the fields and methods from the source tables are directly available without any prefix.

```
page 50143 ImplicitWith
{
    SourceTable = Customer;

    layout
    {
        area(Content)
        {
            field("No."; "No.") { }
            field(Name; Name)
            {
                trigger OnValidate()
                begin
                    Name := 'test';
                end;
            }
        }
    }

    trigger OnInit()
    begin
        if IsDirty() then Insert()
    end;

    local procedure IsDirty(): Boolean
    begin
        exit(Name <> '');
    end;
}
```

On pages it is not only the code in triggers and procedures that is spanned by the implicit with on the source `Rec`; also the source expressions for the fields are covered.

Warnings and using pragma

From Business Central 2020 release wave 2 we begin to warn about the use of explicit and implicit with for extensions that are targeting the cloud. There are two different warnings: `AL0604` and `AL0606`.

> **NOTE**
> The warnings will become errors with a future release. We will at the earliest remove with statement support from the Business Central 2021 release wave 2.

AL0606 - use of explicit with

The warning has a Quick Fix code action that allows you to convert the statement(s) inside the `with` statement to fully-qualified statements, this means statements as shown in The explicit with statement.

AL0604 - use of implicit with

Just qualifying with `Rec.` will not solve this problem. The `IsDirty()` will still be vulnerable to upstream change. We want to remove the implicit with, but also offer an opt-in model to avoid forcing everyone to upgrade their code at once.

The solution for that is to introduce pragmas in AL. A pragma is an instruction to the compiler on how it should understand the code. The pragma instructs the compiler not to create an implicit with for the `Rec` variable.

The following shows the syntax for the implicit with pragma. The pragma must be used before the beginning of the codeunit or page. For more information, see Pragma Directive and Pragma ImplicitWith.

```
#pragma implicitwith disable|restore
```

The fixed example under Pages will then look like this:

```
#pragma implicitwith disable
page 50143 ImplicitWith
{
    SourceTable = Customer;

    layout
    {
        area(Content)
        {
            field("No."; Rec."No.") { }
            field(Name; Rec.Name)
            {
                trigger OnValidate()
                begin
                    Rec.Name := 'test';
                end;
            }
        }
    }

    trigger OnInit()
    begin
        if IsDirty() then Rec.Insert()
    end;

    local procedure IsDirty(): Boolean
    begin
        exit(Name <> '');
    end;
}
#pragma implicitwith restore
```

The Quick Fix code actions will automatically insert the pragma before and after the fixed object.

> **TIP**
> Remember to **Enable Code Actions** in the settings for the AL Language extension. For more information, see Code Actions.

In the `app.json` file, you can set the `NoImplicitWith` flag to disable implicit with when you have rewritten all code. For more information, see JSON Files.

Suppressing warnings

There are two ways of suppressing warnings to unclutter warnings while working on other issues and then afterwards you can fix the warnings at your own pace.

suppressWarnings setting

Warnings can be suppressed globally in an extension by specifying this in the `app.json` file. For more information, see AL Language Extension Configuration. The syntax is:

```
"suppressWarnings": [ "AL0606", "AL0604" ]
```

Pragmas

It is also possible to use `#pragma` to suppress individual warnings for one or more lines of code. For more information, see Pragma Warning.

```
#pragma warning disable AL0606
    // No AL0606 will be shown for code here.
    with Customer do begin
        Name := 'Foo';
        Insert();
    end;

#pragma warning restore AL0606
// Suppression of AL0606 is restored to global state.
```

See Also

AL Development Environment
Developing Extensions in AL
Directives in AL
Pragma Directive
Pragma ImplicitWith
Pragma Warning
AL Simple Statements

Events in AL

2/17/2021 • 3 minutes to read • Edit Online

The use of events is a proven and established programming concept that can ease application upgrade and limit or even eliminate the need for code modifications in customized applications because of application platform changes.

You can use events to design the application to react to specific actions or behavior that occur. Events enable you to separate customized functionality from the application business logic. By using events in the application where customizations are typically made, you can lower the cost of code modifications and upgrades to the original application.

- Code modifications to customized functionality can be made without having to modify the original application.
- Changes to the original application code can be made with minimal impact on the customizations.

Events can be used for different purposes, such as generating notifications when certain behavior occurs or the state of an entity changes, distributing information, and integrating with external systems and applications. For example, in the CRONUS International Ltd. demonstration database, events are used extensively for workflow and Dynamics 365 for Sales integration.

The following table describes all the different event types:

EVENT TYPES	DESCRIPTION
BusinessEvent	Specifies the method to be business type event publisher.
IntegrationEvent	Specifies the method to be integration type event publisher.
InternalEvent	Specifies the method to be an internal event publisher.
Global	Global events are predefined system events.
Trigger	Trigger events are published by the runtime.

The process for implementing these events is slightly different. To learn about the different types, see Event Types.

How events work

The basic principle is that you program events in the application to run customized behavior when they occur. Events in AL are modeled after Microsoft .NET Framework. There are three major participants involved in events: the *event*, a *publisher*, and a *subscriber*.

- An *event* is the declaration of the occurrence or change in the application. An event is declared by an AL method, which is referred to as an *event publisher function*. An event publisher method is comprised of a signature only and does not execute any code.
- A *publisher* is the object that contains event publisher method that declares the event. The publisher exposes an event in the application to subscribers, essentially providing them with a hook-up point in the application.

Publishing an event does not actually do anything in the application apart from making the event available for subscription. The event must be raised for subscribers to respond. An event is raised by adding logic to the application that calls into the publisher to invoke the event (the event publisher method).

Partners or subsystems can then take advantage of the published event in their solutions. An ISV that delivers vertical solutions, and Microsoft itself, are the typical providers of published events.

Business and integration type events must be explicitly published and raised, which means that you must create event publisher functions and add them to objects manually. On the other hand, trigger events, which occur on table and page operations, are published and raised implicitly by the system at runtime. Therefore, no coding is required to publish them.

- A *subscriber* listens for and handles a published event. A subscriber is an AL method that subscribes to a specific event publisher method and includes the logic for handling the event. When an event is raised, the subscriber method is called and its code is run. A subscriber enables partners to hook into the core application functionality without having to do traditional code modifications. Any Dynamics 365 solution provider, which also includes Microsoft, can use event subscribers.

There can by multiple subscribers to a single event publisher method. However, a publisher has no knowledge of subscribers, if any. Subscribers can reside in different parts of the application than publishers.

How to implement events

Implementing events consists of the following tasks:

1. Publish the event.

   For business and integration events, create and configure a method in an application object to be an event publisher method. For more information, see Publishing Events. This is not required for trigger events because these are automatically published by the system.

2. Raise the event.

   Add code that calls the event publisher method. For more information, see Raising Events. This is not required for trigger events because these are raised automatically by the system.

3. Subscribe to the event.

   At the consumer end, add one or more subscriber methods that subscribe to published events when they are raised. For more information, see Subscribing to Events.

See Also

Publishing Events
Raising Events
Subscribing to Events
Developing Extensions Using the New Development Environment

Event Types

2/17/2021 • 7 minutes to read • Edit Online

Dynamics 365 Business Central supports different types of events for different purposes.

Business events

A business event is a custom event that is raised by AL code. It defines a formal contract that carries an implicit promise not to change in future releases. It is the expectation that business events are published by solution ISVs, including Microsoft.

Business events can be compared with publicly released APIs on which 3rd party solution providers develop integrations and additions. Therefore, the downstream cost of making changes to a business event implementation can be considerable for those who use the event in their applications. There may be some cases where changes are required; however, you should keep these to an absolute minimum.

Development considerations

A typical business event reflects changes in "state" with regards to a process. This makes them very well suited for workflow. An example of a business event could be when a sales order has been posted. It is important to note that business events should not be tied to the implementation-details, such as the tables or fields in which the data is stored. Preferably, the event publisher developer should be free to change the implementation, while still keeping the business event intact. To learn about the syntax and example on how to use the BusinessEvent type, see BusinessEvent Attribute.

Business events should be documented with the solution, including the before-state and after-state of the events.

Integration events

An integration event is also a custom event that is raised by AL code, like a business event, except that it does not carry the same promise of not changing, nor does it have the restriction not to expose implementation details.

The main purpose of integration events is to enable the integration of other solutions with Dynamics 365 Business Central without having to perform traditional code modifications.

Development considerations

An integration event can be changed to a business event later. At which time, it must adhere to the same implied contract and commitment as any business event. It can also simply be designed-in hook points for external add-ons. To learn about the syntax and example on how to use the IntegrationEvent type, see IntegrationEvent Attribute.

Global events

Global events are predefined system events that are automatically raised by various base application codeunits. For example, codeunit 40 **LoginManagement** includes several global method triggers, such as CompanyOpen, CompanyClose, and GetSystemIndicator. For most of these global method triggers, there are one or two global events: a before and after event. For example, there is an OnBeforeCompanyOpen event and an OnAfterCompanyOpen event. The global events are defined as integration event publishers by local methods in the following codeunits.

CODEUNIT ID	CODEUNIT NAME	EVENT
40	LoginInManagement	OnRoleCenterOpen
		OnAfterLogInEnd
		OnBeforeLogInStart
		OnBeforeCompanyOpen
		OnAfterCompanyOpen
		OnBeforeCompanyClose
		OnAfterCompanyClose
42	TextManagement	OnBeforeMakeTextFilter
		OnAfterMakeDateTimeFilter
		OnAfterMakeDateFilter
		OnAfterMakeTextFilter
		OnAfterMakeTimeFilter
42	CaptionManagement	OnAfterCaptionClassTranslate
44	ReportManagement	OnAfterGetPrinterName
		OnAfterDocumentPrintReady
		OnAfterGetPaperTrayForReport
		OnAfterHasCustomLayout
		OnAfterSetupPrinters
		OnAfterSubstituteReport
45	AutoFormatManagement	OnAfterAutoFormatTranslate
49	GlobalTriggerManagement	OnAfterGetGlobalTableTriggerMask
		OnAfterOnGlobalInsert
		OnAfterOnGlobalModify
		OnAfterOnGlobalDelete
		OnAfterOnGlobalRename

CODEUNIT ID	CODEUNIT NAME	EVENT
		OnAfterGetDatabaseTableTriggerSetup
		OnAfterOnDatabaseInsert
		OnAfterOnDatabaseModify
		OnAfterOnDatabaseDelete
		OnAfterOnDatabaseRename
		OnBeforeOnDatabaseInsert
		OnBeforeOnDatabaseModify
		OnBeforeOnDatabaseDelete
		OnBeforeOnDatabaseRename

Trigger events

Unlike business and integration events which must be programmed, trigger events are predefined events. Trigger events are published by the runtime and they cannot be raised programmatically. There are two types of trigger events: database trigger events and page trigger events.

> **NOTE**
> Trigger events do not appear as methods in AL for a table or page object.

Database trigger events

Trigger events are automatically raised by the system when it performs database operations on a table object, such as deleting, inserting, modifying, and renaming a record, as defined in a table. Trigger events are closely associated with the table triggers for database operations: OnDelete, OnInsert, OnModify, OnRename, and OnValidate (for fields). For each database operation, there is a "before" and "after" trigger event with a fixed signature.

Available Database Trigger Events

The following table describes the available database trigger events:

DATABASE TRIGGER EVENT WITH SIGNATURE	DESCRIPTION
`OnBeforeDeleteEvent(VAR Rec: Record; RunTrigger: Boolean)`	Executed before a record is deleted from a table.
`OnAfterDeleteEvent(VAR Rec: Record; RunTrigger: Boolean)`	Executed after a record is deleted from a table.
`OnBeforeInsertEvent(VAR Rec: Record; RunTrigger: Boolean)`	Executed before a record is inserted in a table.

DATABASE TRIGGER EVENT WITH SIGNATURE	DESCRIPTION
`OnAfterInsertEvent(VAR Rec : Record; RunTrigger : Boolean)`	Executed after a record is inserted in a table.
`OnBeforeModifyEvent(VAR Rec : Record; VAR xRec : Record; RunTrigger : Boolean)`	Executed before a record is modified in a table.
`OnAfterModifyEvent(VAR Rec : Record; VAR xRec : Record; RunTrigger : Boolean)`	Executed after a record is modified in a table.
`OnBeforeRenameEvent(VAR Rec : Record; VAR xRec : Record; RunTrigger : Boolean)`	Executed before a record is renamed in a table.
`OnAfterRenameEvent(VAR Rec : Record; VAR xRec: Record; RunTrigger : Boolean)`	Executed after a record is renamed in a table.
`OnBeforeValidateEvent(VAR Rec : Record; VAR xRec : Record; CurrentFieldNo : Integer)`	Executed before a field is validated when its value has been changed.
`OnAfterValidateEvent(VAR Rec : Record; VAR xRec : Record; CurrentFieldNo : Integer)`	Executed after a field is validated when its value has been changed.

The following table describes the parameters of the trigger events:

PARAMETER	TYPE	DESCRIPTION
Rec	Record	The table that raises the event.
xRec	Record	The table that raises the event.
RunTrigger	Boolean	Specifies whether to execute the code in the event trigger when it is invoked. If this parameter is true, the code will be executed. If this parameter is false, then the code is not executed.
CurrentFieldNo	Integer	The number of the field that raises the event.

Order of Event Execution

The relative order of execution of database trigger events, table triggers, and database operations is as follows:

ORDER	ITEM	EXAMPLE
1	Trigger event (before)	OnBeforeDeleteEvent
2	Table trigger	OnDelete
3	Global table trigger in codeunit	OnDatabaseDelete
4	Database operations	Delete the record
5	Trigger event (after)	OnAfterDeleteEvent

Page trigger events

Page Trigger events are raised automatically by the system when it performs certain operations in a page object. Page trigger events are closely associated with the standard page triggers, such as OnOpenPage. OnClosePage, and OnAction.

Available Page Trigger Events

The following table describes the available page trigger events:

TRIGGER EVENT WITH SIGNATURE	DESCRIPTION
`OnBeforeActionEvent(VAR Rec : Record)`	Executed before the OnAction Trigger, which is called when a user selects an action on the page.
`OnAfterActionEvent(VAR Rec : Record)`	Executed after the OnAction Trigger, which is called when a user selects an action on the page.
`OnAfterGetCurrRecordEvent(VAR Rec : Record)`	Executed after the OnAfterGetCurrRecord Trigger, which is called after the current record is retrieved from the table.
`OnAfterGetRecordEvent(VAR Rec : Record)`	Executed after the OnAfterGetRecord Trigger, which is called after the record is retrieved from the table but before it is displayed to the user.
`OnBeforeValidateEvent(VAR Rec : Record, VAR xRec : Record)`	Executed before the OnValidate (Page fields) Trigger, which is called when a field loses focus after its value has been changed.
`OnAfterValidateEvent(VAR Rec : Record, VAR xRec : Record)`	Executed after the OnValidate (Page fields) Trigger, which is called when a field loses focus after its value has been changed.
`OnClosePageEvent(VAR Rec : Record)`	Executed after the OnClosePage Trigger, which is called when page closes after the OnQueryClosePage trigger is executed.
`OnDeleteRecordEvent(VAR Rec : Record, VAR AllowDelete : Boolean)`	Executed after the OnDeleteRecord Trigger, which is called before a record is deleted from a table.
`OnInsertRecordEvent(VAR Rec : Record, BelowxRec : Boolean, VAR xRec : Record, VAR AllowInsert : Boolean)`	Executed after the OnInsertRecord Trigger, which is called before a record is inserted in a table.
`OnModifyRecordEvent(VAR Rec : Record, VAR xRec : Record, VAR AllowModify : Boolean)`	Executed after the OnModifyRecord Trigger, which is called before a record is modified in a table.
`OnNewRecordEvent(VAR Rec: Record, BelowxRec : Boolean, VAR xRec : Record)`	Executed after the OnNewRecord Trigger, which is called before a new record is initialized.
`OnOpenPageEvent(VAR Rec : Record)`	Executed after the OnOpenPage Trigger, which is called after a page is initialized and run.
`OnQueryClosePageEvent(VAR Rec : Record, VAR AllowClose : Boolean)`	Executed after the OnQueryClosePage Trigger, which is called as a page closes and before the OnClosePage Trigger executes.

The following table describes the parameters of the trigger events:

PARAMETER	TYPE	DESCRIPTION
Rec	Record	The table that used page that raises the event.
xRec	Record	The table that used page that raises the event.
AllowDelete	Boolean	Specifies whether the OnDeleteRecord trigger call was successful and the record can be deleted. If this parameter is true, the code will be executed. If this parameter is false, then the code is not executed.
AllowModify	Boolean	Specifies whether the OnModifyRecord trigger call was successful and the record can be modified. If this parameter is true, the code will be executed. If this parameter is false, then the code is not executed.
BelowxRec	Boolean	Specifies whether the new record was inserted after the last record in the table (xRec).
AllowClose	Boolean	Specifies whether to the page can close. If this parameter is true, the code will be executed. If this parameter is false, then the code is not executed.

See Also

Events in AL
Publishing Events
Raising Events
Subscribing to Events

Publishing Events

2/17/2021 • 3 minutes to read • Edit Online

The first phase of implementing an event is publishing the event. Publishing an event exposes it in the application. It provides hook up points for subscribers to register to the event, and eventually handle the event if it's raised. An event is published by adding an AL method that is set up as an event publisher.

- Business and integration events require that you manually create an event publisher method for each event that you want to publish. An event publisher method declares the event in the application and makes it available for subscription. However, it doesn't raise the event. After an event is published, you can raise it in your application from where event subscribers can react and handle the event.
- Trigger events don't require that you create publisher methods. Trigger events are predefined event publisher methods that are called automatically at runtime. So trigger events are readily available to subscribers by default.

Creating an event publisher method to publish business and integration events

You create an event publisher method the same way you create any method in AL. But there are specific attributes that you set to make it an event publisher. Additionally, an event publisher method has the following requirements and restrictions that you must follow, otherwise you can't compile your code changes:

- An event publisher method can't include any code except comments.
- An event publisher method can't have a return value, variables, or text constants.

The following procedure provides an outline of the tasks that are involved in creating an event publisher method for declaring an event.

To create an event publisher method

1. Decide where you want to include the event publisher method.

   You can include an event publisher method in the AL code of any object type, such as codeunit, page, or table. You can create a new object or use an existing object.

   > **IMPORTANT**
   > If you include the event publisher method in a page object, the page must have a source table. Otherwise, you can't successfully create an event subscriber method to subscribe to the event.

2. Add an AL method to the object.

   If you don't want the event publisher to be raised from other objects than the one defining it, make it a local method by affixing it with `local`. The event still remains available to event subscribers from other objects.

   Give the method a name that has the format *On[Event]*, where *[Event]* is text that indicates what occurred, such as `OnAddressLineChanged`.

3. Decorate the method with either the Integration attribute or Business attribute as follows:

```
[IntegrationEvent(IncludeSender : Boolean, GlobalVarAccess : Boolean)]
```

or

```
[BusinessEvent(IncludeSender : Boolean)]
```

> **TIP**
> Use the `teventint` snippet for an integration event or the `teventbus` snippet for a business event to get started.

For more information about integration and business events, see Event Types.

4. Add parameters to the method as needed.

   You can include as many parameters of any type as necessary.

   Make sure to expose enough information. Parameters enable subscriber methods to add value to the application. However, don't expose unnecessary parameters that may constrain you from changing or extending methodically in the future.

You can now add code to the application that raises the event by calling the event publisher method. You can also create subscriber methods that handle the event when it's raised.

Example

This example creates the codeunit **50100 MyPublishers** to publish an integration event. The event is published by adding the global method called `OnAddressLineChanged`. The event takes a single text data type parameter.

> **NOTE**
> This example is part of a larger, simple scenario where when users change the address of a customer on the page **21 Customer Card**, you want to check that the address doesn't include a plus sign (+). If it does, you want to display a message. To accomplish this, you will publish an event that is raised when the **Address** field on **Customer Card** is changed, and add an event subscriber method to that includes logic that checks the address value and returns a message to the user if it contains a plus sign. For a complete description of this scenario and all the code involved, see Event Example.

```
codeunit 50100 MyPublishers
{
    [IntegrationEvent(false, false)]
    procedure OnAddressLineChanged(line : Text[100]);
    begin
    end;
}
```

The next step is to raise this event in the application. To see an example for how this event is raised, go to Raising Event Example.

See Also

Raising Events
Subscribing to Events

Raising Events

2/17/2021 • 2 minutes to read • Edit Online

After an event has been published by an event publisher method, you can modify the application to raise the event where it is needed. Subscribers of an event will not react on the event until it is raised in the application.

To raise an event, you add logic in AL code of the application to call the event publisher method that declares the event. The procedure for calling the event publisher method is the same as calling any other method in AL.

When the code that calls the event publisher method is run, all event subscriber methods that subscribe to the event are run. If there are multiple subscribers, the subscriber methods are run one at a time in no particular order. You cannot specify the order in which the subscriber methods are called.

If there are no subscribers to the published event, then the line of code that calls the event publisher method is ignored and not executed.

Snippet support

Typing the shortcut `teventsub` will create the basic event subscriber syntax when using the AL Language extension in Visual Studio Code.

> **TIP**
>
> Typing the keyboard shortcut `Ctrl + space` displays IntelliSense to help you fill in the attribute arguments and to discover which events are available to use.

Example

This example uses a page extension object **50100 MyCustomerExt** to modify the page **21 Customer Card** so that an event is raised when a user changes the **Address** field. This example assumes that the event has already been published by the event publisher method `OnAddressLineChanged` in a separate codeunit called **50100 MyPublishers**.

> **NOTE**
>
> This example is part of a larger, simple scenario where when users change the address of a customer on the page **21 Customer Card**, you want to check that the address does not include a plus sign (+). If it does, you want to return a message to the user. For a description of this scenario and all the code involved, see Event Example.

In the code that follows, the page extension object modifies the `OnBeforeValidate` trigger of the **Customer Card** page to raise the event `OnAddressLineChanged` which includes the new value of the **Address** field.

```
pageextension 50100 MyCustomerExt extends "Customer Card"
{
    layout
    {
        modify(Address)
        {
            trigger OnBeforeValidate();
            var
                Publisher: Codeunit MyPublishers;
            begin
                Publisher.OnAddressLineChanged(Address);
            end;
        }
    }
}
```

To learn about how the event used in this example is published, see Publishing Events Example.

The next step would be to subscribe to the event to handle to condition. To see an example of how to subscribe to this event, see Subscribing to Events Example.

See Also

Publishing Events
Subscribing to Events
Events Dynamics 365

Subscribing to Events

2/17/2021 • 5 minutes to read • Edit Online

To handle events, you design event subscribers. Event subscribers determine what actions to take in response to an event that has been raised. An event subscriber is a method that listens for a specific event that is raised by an event publisher. The event subscriber includes code that defines the business logic to handle the event. When the published event is raised, the event subscriber is called and its code is run.

Subscribing to an event tells the runtime that the subscriber method must be called whenever the publisher method is run, either by code (as with business and integration events) or by the system (as with trigger events). The runtime establishes the link between an event raised by the publisher and its subscribers, by looking for event subscriber methods.

There can be multiple subscribers to the same event from various locations in the application code. When an event is raised, the subscriber methods are run one at a time in no particular order. You can't specify the order in which the subscriber methods are called.

Creating an event subscriber method

You create an event subscriber method just like other methods except that you specify properties that set up the subscription to an event publisher. The procedure is slightly different for database and page trigger events than business and integration events. Business and integration events are raised by event publisher methods in application code. Trigger events are predefined system events that are raised automatically on tables and pages.

For an explanation about the different types, see Event Types.

To create an event subscriber method

1. Decide which codeunit to use for the event subscriber method.

   You can create a new codeunit or use an existing one.

2. Add an AL method to the codeunit.

   We recommend that you give the method a name that indicates what the subscriber does, and has the format *[Action][Event]*. *[Action]* is text that describes what the method does. *[Event]* is the name of the event publisher method to which it subscribes.

3. Add code to the method for handling the event.

4. Decorate the event subscriber method with the EventSubscriber attribute.

   ```
   [EventSubscriber(ObjectType::<Event Publisher Object Type>, <Event Publisher Object>, '<Published
   Event Name>', '<Published Event Element Name>', <SkipOnMissingLicense>, <SkipOnMissingPermission>)]
   ```

   Set the arguments according to the following table. For optional arguments, if you don't want to set a value, use an empty value (`''`). In this case, the default value, if any, is used.

ARGUMENT	DESCRIPTION	OPTIONAL

ARGUMENT	DESCRIPTION	OPTIONAL
`<Event Publisher Object Type>`	Specify the type of object that publishes the event. This argument can be `Codeunit`, `Page`, `Report`, `Table`, or `XMLPort`.	no
`<Event Publisher Object>`	Specify the object that publishes the event. You can set this argument to the ID, such as `50100`, or the recommended way is to use the object name by using the syntax `<Object Type>::"<Object Name>"`, such as `Codeunit::"MyPublishers"`, or for database triggers `Database::"Customer"`.	no
`<Published Event Name>`	Specify the name of method that publishes the event in the object that is specified by the `<Event Publisher Object>` parameter.	no
`<Published Event Element Name>`	Specifies the table field that the trigger event pertains to. This argument only requires a value for database trigger events, that is, when the `<Event Publisher Object Type>` is set to `Table` and the `<Published Event Name>` argument is a validate trigger event, such as `OnAfterValidateEvent`.	no
`<SkipOnMissingLicense>`	Set to `true` to skip the event subscriber method call if the user's license doesn't cover the event subscriber codeunit. If `false`, an error is thrown and the code execution stops. `false` is the default.	yes
`<SkipOnMissingPermission>`	Set to `true` to skip the event subscriber method call if the user doesn't have the correct permissions the event subscriber codeunit. If `false`, an error is thrown and the code execution stops. `false` is the default.	yes

TIP

There are couple things that can make defining an event subscriber method easier. You can use the `teventsub` snippet to get started. Then, typing the keyboard shortcut `Ctrl + space` displays IntelliSense to help you fill the attribute arguments and discover which events are available. Or, use the Shift+Alt+E keyboard shortcut to look up the event you want to subscribe to and insert the code.

5. Optionally, set the codeunit's **EventSubscriberInstance** property to specify how the event subscriber method will be bound to the instance of this codeunit.

   For more information, see EventSubscriberInstance Property.

Example 1

This example creates the codeunit **50101 MySubscribers** to subscribe to an event that has been published by the event publisher method called `OnAddressLineChanged` in the codeunit **50100 MyPublishers**. The event is raised by a change to the **Address** field on page **21 Customer Card**. This example assumes:

- The codeunit **50100 MyPublishers** with the event publisher method `OnAddressLineChanged` already exists. For an example, see Publishing Event Example.
- The code for raising the `OnAddressLineChanged` event has been added to the **Customer Card** page. For an example, see Raising Event Example.

The following code creates a codeunit called **50101 MySubscribers** that includes an event subscriber method, called `CheckAddressLine`. The method includes code for handling the published event.

```
codeunit 50101 MySubscribers
{
    EventSubscriberInstance = StaticAutomatic;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"MyPublishers", 'OnAddressLineChanged', '', true,
true)]
    procedure CheckAddressLine(line : Text[100]);
    begin
        if (STRPOS(line, '+') > 0) then begin
            MESSAGE('Can''t use a plus sign (+) in the address [' + line + ']');
        end;
    end;
}
```

> **NOTE**
>
> This example is part of a larger, simple scenario where when users change the address of a customer on the page **21 Customer Card**, you want to check that the address does not include a plus sign (+). If it does, you want to return a message to the user. For a description of this scenario and all the code involved, see Event Example.

Example 2

This example achieves the same as example 1, except it subscribes to the page trigger event `OnBeforeValidateEvent` on the `Address` field instead. By using the page trigger, you avoid creating an event publisher and adding code to raise the event. The event is raised automatically by the system.

```
codeunit 50101 MySubscribers
{
    EventSubscriberInstance = StaticAutomatic;

    [EventSubscriber(ObjectType::Page, Page::"Customer Card", 'OnBeforeValidateEvent', 'Address', true,
true)]
    local procedure CheckAddressLine(var Rec : Record Customer)
    begin
        if (STRPOS('Rec.Address', '+') > 0) then begin
            MESSAGE('Can''t use a plus sign (+) in the address [' + 'Address' + ']');
        end;
    end;
}
```

See Also

Publishing Events
Raising Events
Event Types
Events in AL
EventSubscriberInstance Property
EventSubscriber Attribute

Discoverability of Events

2/17/2021 • 2 minutes to read • Edit Online

You subscribe to events to extend application and interact with the base application and other extensions. This topic describes how to discover events that you can subscribe to without writing the code manually. Using the **Event Recorder**, you can record the events that are published and raised while performing the actions of your scenario. For example, record the events raised when you post a purchase order and identify the events that you need for your extension. You can retrieve the events in the form of AL snippet code and use them in Visual Studio Code directly.

Using the Event Recorder

You can launch the event recorder session from Dynamics 365 Business Central. It can be accessed in the following two ways:

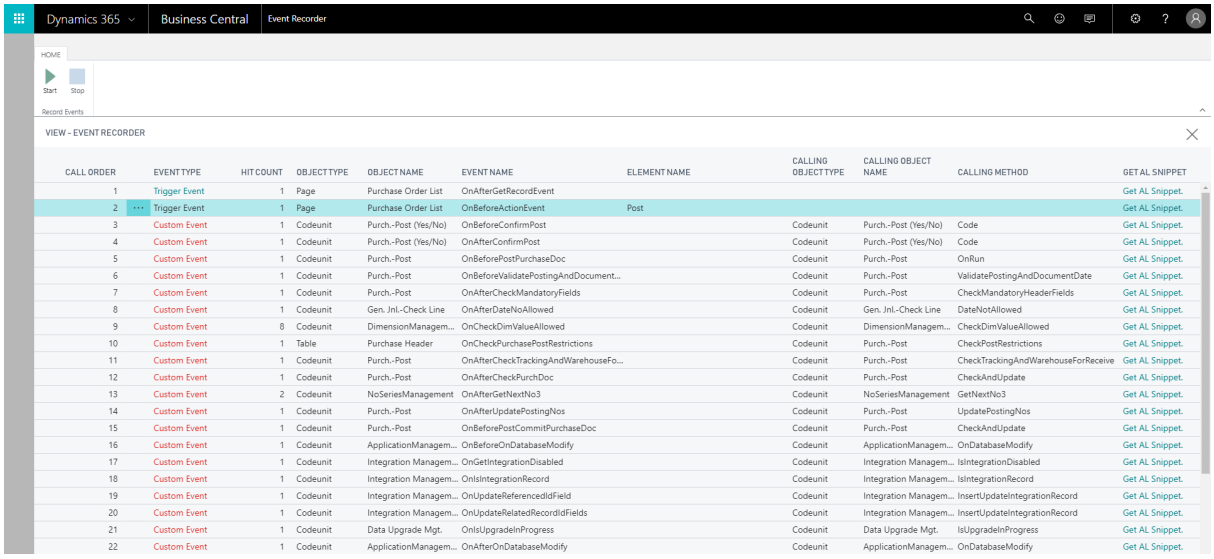
- In Dynamics 365 Business Central, search for **Event Recorder**.
- In Visual Studio Code, use the `Ctrl+Shift+P` keys and select the **Open Event Recorder** command to open the **Event Recorder** page in the Dynamics 365 Business Central web client.

> **NOTE**
>
> The event recorder captures all events that are raised in the same session. If the actions performed by the user are in another session, then the event recorder will not capture them.

How to record events

The following steps describe how to record events when you are on the **Event Recorder** page.

1. On the **Event Recorder** page, in the upper right corner, choose **Open this page in a new window**.
2. Then on the **Event Recorder** page, choose **Record Events**, and then choose the **Start** button.
3. In the original window, perform all the actions that you want to record while the event recorder session is on. For example, post a purchase order.
4. After you have performed the actions of your scenario, navigate back to the window that has the **Event Recorder** page open and click the **Stop** button to finish recording.
   All the events raised while performing the actions of your scenario are recorded and can be viewed on as shown below.



CALL ORDER	EVENT TYPE	HIT COUNT	OBJECT TYPE	OBJECT NAME	EVENT NAME	ELEMENT NAME	CALLING OBJECT TYPE	CALLING OBJECT NAME	CALLING METHOD	GET AL SNIPPET
1	Trigger Event	1	Page	Purchase Order List	OnAfterGetRecordEvent					Get AL Snippet.
2	Trigger Event	1	Page	Purchase Order List	OnBeforeActionEvent	Post				Get AL Snippet.
3	Custom Event	1	Codeunit	Purch.-Post (Yes/No)	OnBeforeConfirmPost		Codeunit	Purch.-Post (Yes/No)	Code	Get AL Snippet.
4	Custom Event	1	Codeunit	Purch.-Post (Yes/No)	OnAfterConfirmPost		Codeunit	Purch.-Post (Yes/No)	Code	Get AL Snippet.
5	Custom Event	1	Codeunit	Purch.-Post	OnBeforePostPurchaseDoc		Codeunit	Purch.-Post	OnRun	Get AL Snippet.
6	Custom Event	1	Codeunit	Purch.-Post	OnBeforeValidatePostingAndDocument...		Codeunit	Purch.-Post	ValidatePostingAndDocumentDate	Get AL Snippet.
7	Custom Event	1	Codeunit	Purch.-Post	OnAfterCheckMandatoryFields		Codeunit	Purch.-Post	CheckMandatoryHeaderFields	Get AL Snippet.
8	Custom Event	1	Codeunit	Gen. Jnl.-Check Line	OnAfterDateNoAllowed		Codeunit	Gen. Jnl.-Check Line	DateNotAllowed	Get AL Snippet.
9	Custom Event	8	Codeunit	DimensionManagem...	OnCheckDimValueAllowed		Codeunit	DimensionManagem...	CheckDimValueAllowed	Get AL Snippet.
10	Custom Event	1	Table	Purchase Header	OnCheckPurchasePostRestrictions		Codeunit	Purch.-Post	CheckPostRestrictions	Get AL Snippet.
11	Custom Event	1	Codeunit	Purch.-Post	OnAfterCheckTrackingAndWarehouseFo...		Codeunit	Purch.-Post	CheckTrackingAndWarehouseForReceive	Get AL Snippet.
12	Custom Event	1	Codeunit	Purch.-Post	OnAfterCheckPurchDoc		Codeunit	Purch.-Post	CheckAndUpdate	Get AL Snippet.
13	Custom Event	2	Codeunit	NoSeriesManagement	OnAfterGetNextNo3		Codeunit	NoSeriesManagement	GetNextNo3	Get AL Snippet.
14	Custom Event	1	Codeunit	Purch.-Post	OnAfterUpdatePostingNos		Codeunit	Purch.-Post	UpdatePostingNos	Get AL Snippet.
15	Custom Event	1	Codeunit	Purch.-Post	OnBeforePostCommitPurchaseDoc		Codeunit	Purch.-Post	CheckAndUpdate	Get AL Snippet.
16	Custom Event	1	Codeunit	ApplicationManagem...	OnBeforeOnDatabaseModify		Codeunit	ApplicationManagem...	OnDatabaseModify	Get AL Snippet.
17	Custom Event	1	Codeunit	Integration Managem...	OnGetIntegrationDisabled		Codeunit	Integration Managem...	IsIntegrationDisabled	Get AL Snippet.
18	Custom Event	1	Codeunit	Integration Managem...	OnIsIntegrationRecord		Codeunit	Integration Managem...	IsIntegrationRecord	Get AL Snippet.
19	Custom Event	1	Codeunit	Integration Managem...	OnUpdateReferencedIdField		Codeunit	Integration Managem...	InsertUpdateIntegrationRecord	Get AL Snippet.
20	Custom Event	1	Codeunit	Integration Managem...	OnUpdateRelatedRecordIdFields		Codeunit	Integration Managem...	InsertUpdateIntegrationRecord	Get AL Snippet.
21	Custom Event	1	Codeunit	Data Upgrade Mgt.	OnIsUpgradeInProgress		Codeunit	Data Upgrade Mgt.	IsUpgradeInProgress	Get AL Snippet.
22	Custom Event	1	Codeunit	ApplicationManagem...	OnAfterOnDatabaseModify		Codeunit	ApplicationManagem...	OnDatabaseModify	Get AL Snippet.

5. Choose **Get AL Snippet** to get the event subscription code in AL. You can use the AL snippet code in Codeunits to subscribe to those events.

> **NOTE**
>
> The recorded events are not saved. When you refresh the page, the recorded events disappear.

For more information on how to subscribe to events, see Subscribing to Events.

Recorded Events

All the recorded events display in the order they were called. The Event Recorder page provides information on the events that were raised including the details whether the raised events were trigger events or custom events. The custom events are either Business Events or Integration Events. For more information, see Event Types.

You can identify the Event types, additionally, you can discover which object types and methods raised the events with the details like calling methods, object types, and object names. For more information about Events, see Events in AL.

See Also

Events in AL
Publishing Events
Raising Events
Subscribing to Events
Debugging in AL
Developing Extensions

Event Example

2/17/2021 • 2 minutes to read • Edit Online

This article includes a simple code example to explain how to use Business Central events. The example uses an event to verify a customer's address. When a user changes the address of a customer on the page **21 Customer Card**, an event is used to check that the address doesn't include a plus sign (+). If it does, a message is displayed in the client. In this example, you'll add code that:

- Publishes an event that is raised when the **Address** field on **Customer Card** is changed.
- Raises the event from the **Customer Card** page
- Subscribes to the event to check the address value and return a message to the user if it contains a plus sign.

```
// The following code creates codeunit that publishes the `OnAddressLineChanged` event.

codeunit 50100 MyPublishers
{
    [IntegrationEvent(false, false)]
    procedure OnAddressLineChanged(line: Text[100])
    begin
    end;
}

// The following code extends the Customer Card page to raise the `OnAddressLineChanged` event
// when the Address field is changed.
pageextension 50100 MyCustomerExt extends "Customer Card"
{
    layout
    {
        modify(Address)
        {
            trigger OnBeforeValidate();
            var
                Publisher: Codeunit MyPublishers;
            begin
                Publisher.OnAddressLineChanged(Address);
            end;
        }
    }
}

// The following code creates an event subscriber codeunit.
// It declares the `CheckAddressLine` event subscriber that listens for OnAddressLineChanged event.
// The event subscriber displays a message in the client when '+' is used in the **Address** field.

codeunit 50101 MySubscribers
{
    //Set the event subscribers to bind automatically to the event
    EventSubscriberInstance = StaticAutomatic;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"MyPublishers", 'OnAddressLineChanged', '', true,
true)]
    procedure CheckAddressLine(line: Text[100]);
    begin
        if (STRPOS(line, '+') > 0) then begin
            MESSAGE('Can''t use a plus sign (+) in the address [' + line + ']');
        end;
    end;
}
```

TIP

This example binds the event subscriber automatically to the event. You can also bind the event subscriber manually, by setting the EventSubscriberInstance property. For more information, including an example, see EventSubscriberInstance property.

See Also

Publishing Events
Raising Events
Subscribing to Events
Events Dynamics 365

Walkthrough: Implementing New Workflow Events and Responses

2/17/2021 • 12 minutes to read • Edit Online

If a business scenario requires a workflow event or a workflow response that is not supported in a Business Central solution, you must implement it by extending the application code.

In the **Workflow** page, the workflow administrator creates a workflow by listing the involved steps on the lines. Each step consists of a workflow event, moderated by event conditions, and a workflow response, customized by response options. You define workflow steps by filling fields on workflow lines from fixed lists of event and response values representing scenarios that are supported by the application code. For more information, see Set Up Workflows in the business functionality content.

The following procedure describes how to add a new workflow event and a new workflow response and then register the involved object relations, so that the new elements can be used in workflows. You can then share your code as an app or a per-tenant extension, for example. The workflow administrator can then select the new workflow event and response from the **Workflow** page to incorporate them in new or existing workflow steps.

> **IMPORTANT**
>
> To ensure that custom workflow records are upgraded correctly, you must add new workflow events, workflow responses, and workflow table relations to dedicated extension points, as described in this procedure. During an upgrade to the next version, the libraries of workflow events, responses, and table relations are removed and then recreated with the latest content from Microsoft. By adding your custom workflow records using subscriptions to the Microsoft-provided extension points, you ensure that your custom record library gets recreated after an upgrade.

> **NOTE**
>
> This topic refers to two types of events:
>
> - *Workflow event*: An occurrence in the application that users in the client can select from the **Workflow** page to define workflow steps. For more information, see Workflows in Dynamics 365 Business Central in the business functionality content.
> - *Event*: The declaration of the occurrence or change in the application. Workflow events typically subscribe to events. For more information, see Events in AL.

The development work involved in creating a new workflow event and a related workflow response consists of the following tasks, as a minimum:

1. Create a workflow event
    a. Create a workflow event code that identifies the workflow event
    b. Add the workflow event code to the **Workflow Event** table
    c. Create and publish an event that the workflow event subscribes to
    d. Raise the event
    e. Subscribe to the event and implement the workflow event
2. Create a workflow response

a. Create a workflow response code that identifies the workflow response

b. Add the workflow response code to the **Workflow Response** table

c. Implement the workflow response

d. Enable that the workflow response can be executed

e. Add a new workflow response option

3. Register workflow event/response combinations needed for the new workflow response
4. Register workflow event hierarchies needed for the new workflow event
5. Creating table relations between entities used when the new workflow event and response are used

> **NOTE**
> Data and code samples in this procedure refer loosely to a workflow step of sending a notification when a purchase header is posted. However, the procedure alone does not result in a complete solution. The purpose of the walkthrough is simply to illustrate the process.

Workflow event

In this section, we'll create a code to identify the workflow event, add the workflow event to the library, create an event that the workflow event subscribes to, raise the event, and then subscribe to the event and implement the workflow event.

Each subsection takes you through the discrete steps.

To create a workflow event code that identifies the workflow event

1. Create a new .al file, such as MyWorkflowEvents.codeunit.al, and add a codeunit that will be used for new workflow events. Name it to reflect that it is used to identify the new workflow event, such as `MyWorkflowEvents`.
2. Add a method in the codeunit. Optionally, use the shortcut `teventint`. Name the method to reflect that it is used to identify the workflow event, such as `MyWorkflowEventCode`, and make it take 128 characters of code as a parameter.

> **TIP**
> The terminology can be a bit confusing here. This method is not an *AL event*. It's a method that declares the *workflow event*, and it will subscribe to an *AL event* that, when triggered, will trigger the *workflow event*.

Your MyWorkflowEvents.codeunit.al file now looks like this:

```
codeunit 50101 MyWorkflowEvents
{
    procedure MyWorkflowEventCode(): code[128];
    begin
    end;
}
```

To add the workflow event code to the Workflow Event table

1. Create another method in the codeunit. Name it to reflect that it is used to add the workflow event to the library, such as `AddMyWorkflowEventsToLibrary`.

This method will subscribe to the `OnAddWorkflowEventsToLibrary` method in the `Workflow Event Handling` codeunit in the base application, so you must set the EventSubscriber attribute, and you must add code that handles the event.

The following code illustrates the new method in the `MyWorkflowEvents` codeunit:

```
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Event Handling",
'OnAddWorkflowEventsToLibrary', '', true, true)]
    procedure AddMyWorkflowEventsToLibrary()
    var
        WorkflowEventHandling: Codeunit"Workflow Event Handling";
    begin
        WorkflowEventHandling.AddEventToLibrary(MyWorkflowEventCode(), Database::"Purchase Header", 'My
workflow event description', 0, false);
    end;
```

To publish an event that the workflow event subscribes to

1. Create another codeunit, such as MyEvents.codeunit.al, and add a method that publishes your event. Name the method to reflect that it is used as the publisher event, such as `OnAfterPostPurchaseHeader`.

   ```
   codeunit 50102 MyEvents
   {
       [IntegrationEvent(false, false)]
       procedure OnAfterPostPurchaseHeader(PurchaseHeader: Record "Purchase Header")
       begin
       end;
   }
   ```

   Select the event type that is relevant for the workflow event, such as Integration. For more information, see Event Types.

To raise the event

1. Create an object or an extension object to add the code that will raise the event that triggers the workflow event, such as the `Purch.-Post` codeunit.

   The following code raises the event by extending the `Purchase Order` page object in a new file, `MyPurchOrder.PageExt.al`.

   ```
   pageextension 50100 WFW_PurchaseOrder extends "Purchase Order"
    {
        layout
        {
            modify(Status)
            {
                trigger OnBeforeValidate();
                var
                    MyEvents: Codeunit MyEvents;
                begin
                    MyEvents.OnAfterPostPurchaseHeader(Rec)
                end;
            }
        }
    }
   ```

   For more information, see Raising Events.

To subscribe to the event and implement the workflow event

1. Go back to the MyWorkflowEvents.codeunit.al file file and add another method in the `MyWorkflowEvents` codeunit. Name the new method to reflect that it is used to subscribe to and implement the workflow event, such as `RunWorkflowOnAfterPostPurchaseHeader`.

The following code illustrates the new workflow event that subscribes to your previously created event:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::MyEvents, 'OnAfterPostPurchaseHeader', '', true, true)]
local procedure RunWorkflowOnAfterPostPurchaseHeader(var PurchaseHeader: Record "Purchase Header")
var
    WorkflowManagement: Codeunit "Workflow Management";
begin
    WorkflowManagement.HandleEvent(MyWorkflowEventCode, PurchaseHeader);
end;
```

Another task that you can perform at this point is to specify which filter fields appear in the **Workflow Event Conditions** page.

For more information, see Subscribing to Events.

You have now created a new workflow event. Next, we'll create a new workflow response that relates to the workflow event.

Workflow response

Create a new .al file, such as MyWorkflowResponses.codeunit.al, with code to identify the workflow response, add the workflow response code to the library, implement the workflow response, and then enable that the workflow response can be executed.

To create a workflow response code that identifies the workflow response

1. Add a new codeunit that will be used for the new workflow responses. Name it to reflect that it handles your new responses, such `MyWorkflowResponses`.

2. Add a method in the codeunit. Name the method to reflect that it is used to identify the workflow response, such as `MyWorkflowResponseCode` with a return value of code (128).

To add the workflow response code to the Workflow Response table

1. Add another method in the codeunit that will be the event subscriber. Name it to reflect that it is used to add the workflow response to the library, such as `AddMyWorkflowResponsesToLibrary` and set it to subscribe to the OnAddWorkflowResponsesToLibrary `method in the` Workflow Response Handling` codeunit.

```
 [EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Event Handling",
'OnAddWorkflowEventPredecessorsToLibrary', '', false, false)]
 local procedure AddWorkflowEventHierarchiesToLibrary(EventFunctionName: Code[128])
 begin
 end;
```

2. Add a local variable for the codeunit:

```
[...]
var
WorkflowResponseHandling: Codeunit "Workflow Response Handling";
```

3. In the method, write code that registers the response, so that you end up with something like the following code.

```
 [EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Response Handling",
'OnAddWorkflowResponsesToLibrary', '', true, true)]
 local procedure AddMyWorkflowResponsesToLibrary()
 var
     WorkflowEventHandling: codeunit "Workflow Event Handling";
 begin
     WorkflowResponseHandling.AddResponseToLibrary(MyWorkflowResponseCode, Database::"Purchase
Header", 'Send a notification.', 'GROUP 0');
     End
 end;
```

NOTE

In the To add a new workflow response option section, you will change the GROUP value to *50100*. This way, you'll be able to see the workflow in action.

The codeunit now looks something like this:

```
codeunit 50103 MyWorkflowResponses
{
    procedure MyWorkflowResponseCode(): code[128];
    begin
    end;

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Response Handling",
'OnAddWorkflowResponsesToLibrary', '', true, true)]
    local procedure AddMyWorkflowResponsesToLibrary()
    var
        WorkflowResponseHandling: Codeunit "Workflow Response Handling";
    begin
        WorkflowResponseHandling.AddResponseToLibrary(MyWorkflowResponseCode, Database::"Purchase Header",
'Send a notification.', 'GROUP 0');
    end;
}
```

To implement the workflow response

1. Create another method in the codeunit. Name it to reflect that it is used to implement the workflow response, such as `MyWorkflowResponse`, that takes a parameter based on the Purchase Header table.
2. In the method, write code that handles the response, so that you end up with something like the following code.

```
procedure MyWorkflowResponse(PurchaseHeader: Record "Purchase Header")
begin
    Message('Status change on: %1 %2', PurchaseHeader."Document Type", PurchaseHeader."No.");
end;
```

To enable that the workflow response can be executed

1. Create another method in the codeunit that subscribes to the `OnExecuteWorkflowResponse` event on the `Workflow ResponseHandling` codeunit. Name it to reflect that it is used to enable the new workflow response to be executed alongside existing workflow responses, such as `ExecuteMyWorkflowResponses`.
2. In the event subscriber method, write code that enables the response, so that you end up with something like the following code.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Response Handling",
'OnExecuteWorkflowResponse', '', true, true)]
procedure ExecuteMyWorkflowResponses(ResponseWorkflowStepInstance: Record "Workflow Step Instance";
var ResponseExecuted: Boolean; var Variant: Variant; xVariant: Variant)
var
    WorkflowResponse: record "Workflow Response";
begin
    if WorkflowResponse.GET(ResponseWorkflowStepInstance."Function Name") then
        case WorkflowResponse."Function Name" of
            MyWFResponseCode:
                BEGIN
                    MyWorkflowResponse(Variant);
                    ResponseExecuted := TRUE;
                END;
        END;
end;
```

You'll update this method in step 8 in the next section.

To add a new workflow response option

1. Create a table extension object that extends table 1523, **Workflow Step Argument**, such as MyWorkflowStepArgument.TableExt.al.
2. Add a field that reflects your new response option, such as **My New Response Option**.

```
tableextension 50100 WFW_WorkflowStepArgument extends "Workflow Step Argument"
{
    fields
    {
        field(50100; "My New Response Option"; Text[100])
        {
        }
    }
}
```

3. Create a page extension object that extends page 1523, **Workflow Response Options**, such as MyworkflowStepArgument.PageExt.al.
4. Add a group and a control for the new field.

```
pageextension 50101 WFW_WorkflowResponseOptions extends "Workflow Response Options"
{
    layout
    {
        addlast(content)
        {
            group(Group50100)
            {
                Visible = Rec."Response Option Group" = 'GROUP 50100';
                ShowCaption = false;

                field(MyNewResponseOption; Rec."My New Response Option")
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}
```

Here, the **Visibility** property of the group is set to `"Response Option Group" = 'GROUP 51000'`, but you can

set it to another value.

5. Go back to MyWorkflowResponses.codeunit.al and the ´AddMyWorkflowResponsesToLibrary` method.
6. In the method code, change `'GROUP 0'` to `'GROUP 50100'`.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Response Handling",
'OnAddWorkflowResponsesToLibrary', '', true, true)]
local procedure AddMyWorkflowResponsesToLibrary()
var
    WorkflowResponseHandling: Codeunit "Workflow Response Handling";
begin
    WorkflowResponseHandling.AddResponseToLibrary(MyWorkflowResponseCode, Database::"Purchase
Header", 'Send a notification.', 'GROUP 50100');
end;
```

7. To use the new option in the `MyWorkflowResponse` method, proceed to add a local parameter and a local variable and show a message as the response.

```
 local procedure MyWorkflowResponse(PurchaseHeader: Record "Purchase Header"; WorkflowStepInstance:
Record "Workflow Step Instance")
 var
     WorkflowStepArgument: Record "Workflow Step Argument";
 begin
     if WorkflowStepArgument.Get(WorkflowStepInstance.Argument) then;

     Message('Status change on: %1 %2.\%3', PurchaseHeader."Document Type", PurchaseHeader."No.",
WorkflowStepArgument."My New Response Option")
 end;
```

8. In the `ExecuteMyWorkflowResponses` method, make the following code change:

   Change from this code: `MyWorkflowResponse(Variant);`

   Change to this code: `MyWorkflowResponse(Variant,ResponseWorkflowStepInstance);`

   You have now created the actual workflow event and response. Proceed to perform various tasks that enable them to be used in workflows.

Register workflow event/response combinations

In this section, you'll add new workflow event/response combinations to table 1509 **WF Event/Response Combination** so that they appear correctly in the **Workflow Events** and **Workflow Responses** pages.

To register workflow event/response combinations needed for the new workflow response

1. Open the codeunit that you created in the Workflow response section, `My Workflow Responses`.
2. Create another method in the codeunit that subscribes to the `OnAddWorkflowResponsePredecessorsToLibrary` event on the `Workflow Response Handling` codeunit . Name it to reflect that it is used to add the workflow event/response combinations to table 1509 **WF Event/Response Combination**, such as `AddMyWorkflowEventResponseCombinationsToLibrary`.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Response Handling",
'OnAddWorkflowResponsePredecessorsToLibrary', '', false, false)]
local procedure AddMyworkflowEventOnAddWorkflowResponsePredecessorsToLibrary(ResponseFunctionName:
Code[128])
var
    MyWorkflowEvents: Codeunit MyWorkflowEvents;
    WorkflowResponseHandling: Codeunit "Workflow Response Handling";
begin
    Case ResponseFunctionName of
        MyWorkflowResponseCode():
            WorkflowResponseHandling.AddResponsePredecessor(MyWorkflowResponseCode(),
MyWorkflowEvents.MyWorkflowEventCode());
    End
end;
```

In the method, write code that registers event/response combinations that you want to support in your application, using a CASE statement, such as the code in the example above.

You can also do this work from the user interface on page 1507 **Workflow-Event-Response-Combinations**.

Register workflow event hierarchies

In this section, you'll add new workflow event/event combinations to table 1509 **WF Event/Response Combination** so that they workflow events appear in the correct hierarchy in the **Workflow Events** page.

To register workflow event hierarchies needed for the new workflow event

1. Go back to the MyWorkflowEvents.codeunit.al file with the codeunit that you created in the To create a workflow event code that identifies the workflow event section, `My Workflow Events`.
2. Create another method in the codeunit that subscribes to the `OnAddWorkflowEventPredecessorsToLibrary` event on the `Workflow Event Handling` codeunit. Name it to reflect that it is used to add the workflow event hierarchies to table 1509 **WF Event/Response Combination**, such as `AddWorkflowEventHierarchiesToLibrary`.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Event Handling",
'OnAddWorkflowEventPredecessorsToLibrary', '', false, false)]
local procedure AddWorkflowEventHierarchiesToLibrary(EventFunctionName: Code[128])
var
    WorkflowEventHandling: codeunit "Workflow Event Handling";
begin
    Case EventFunctionName of
        MyWorkflowEventCode():
            //WorkflowEventHandling.AddEventPredecessor(MyWorkflowEventCode(),
WorkflowEventHandling./*[Add your predecessor event code]*/);
            ;
    End
end;
```

In the method, write code that registers event hierarchies that you want to support in your application, using a CASE statement, such as the code in the example above.

You can also do this work from the user interface on page 1506 **Workflow-Event-Hierarchies**.

Create table relations

Workflows events can be executed on different types of records. To keep track of these, you must define relations between the involved records. In this section, you'll create relationships between the entities that are

used when the new workflow event and response are used.

To create table relations between entities that are processed when the new workflow event and response are used in workflows

1. Got back to the MyWorkflowEvents.codeunit.al file with the codeunit that you created in the To create a workflow event code that identifies the workflow event section, `My Workflow Events`.

2. Create another method in the codeunit that subscribes to the `OnAddWorkflowTableRelationsToLibrary` event on the `Workflow Event Handling` codeunit. Name it to reflect that it is used add workflow table relations in table 1505 **Workflow Table Relation**, such as `AddWorkflowTableRelationsToLibrary`.

   ```
   [EventSubscriber(ObjectType::Codeunit, Codeunit::"Workflow Event Handling",
   'OnAddWorkflowTableRelationsToLibrary', '', false, false)]
   local procedure AddWorkflowTableRelationsToLibrary()
   var
       WorkflowSetup: Codeunit "Workflow Setup";
   begin
       WorkflowSetup.InsertTableRelation(Database::"Purchase Header", 1, Database::"Approval Entry", 2);
   end;
   ```

   In the method, write code that registers table relations that you want to support in your application, such as the example above.

   You can also do this work from the user interface on page 1509 **Workflow Table-Relations**.

You have now enabled a new workflow scenario by implementing the required workflow event and response in the application code. The workflow administrator can now select the workflow event and workflow response from the **Workflow** page to define new or edit existing workflows. For more information, see Set Up Workflows in the business functionality content.

See Also

Workflows in Dynamics 365 Business Central
Set Up Workflows
Event Example
Events in AL
Page Extension Object
Table Extension Object
Codeunit Object
Table Object
Getting Started with AL
Development and Administration for Dynamics 365 Business Central

Notifications

2/17/2021 • 5 minutes to read • Edit Online

Notifications provide a programmatic way to send non-intrusive information to the User Interface (UI) in the Web client. Notifications differ from messages initiated by the **Message** method. Messages are modal, which means users are typically required to address the message and take some form of corrective action before they continue working. On the other hand, notifications are non-modal. Their purpose is to give users information about a current situation, but do not require any immediate action or block users from continuing with their current task. For example, you could have a notification that a customer's credit limit is exceeded.

Notifications in the UI

In the UI, notifications appear in the **Notification** bar (similar to validation errors) at the top of the page on which a user is currently working. The user can then choose to dismiss the notification, which clears it. Or, if actions are defined on notification, the user can choose one of the actions.

- There can be multiple notifications. The notifications appear in chronological order from top to bottom.
- Notifications remain for the duration of the page instance or until the user dismisses them or takes action on them.
- Notifications that are defined on sub-pages, for example in parts and FactBoxes, appear in the same **Notification** bar.
- Validation errors on the page will be shown first.

Notifications in the development environment

By using the **Notification** and **NotificationScope** data types and methods in AL, you can add code to send notifications to users. The following table provides an overview of the available methods. The sections that follow provide additional information about how to create notifications.

METHOD	DESCRIPTION
Message	Specifies the content of the notification that appears in the UI.
Scope	Specifies the scope in which the notification appears.
Send	Sends the notification to be displayed by the client.
AddAction	Adds an action on the notification.
SetData	Sets a data property value for the notification
GetData	Gets a data property value from the notification.
Recall	Recalls a sent notification.

Creating and sending a notification

You create a notification by using the **Message** and **Send** methods. The **Message** method defines the message

part of the notification. When the **Send** method is called, the notification is sent to the client and content of the message is displayed.

```
MyNotification.Message := 'This is a notification';
MyNotification.Send();
```

The **Send** method call should be the last statement in the notification code, after any **AddAction** or **SetData** method calls for the notification instance.

Defining the notification scope

The scope determines where the notification is broadcast in the client. There are two different scopes: *LocalScope* and *GlobalScope*.

- A *LocalScope* notification appears in context of the user's current task, that is, on the page the user is currently working on. *LocalScope* is the default.
- A *GlobalScope* notification is not directly related to the current task, and will appear regardless of which the page the user is viewing.

  > **NOTE**
  > *GlobalScope* is currently not supported. This will be implemented in a future release.

The following code creates a notification in the *LocalScope*:

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.Send();
```

Adding actions on a notification

You add actions on notifications by using the **AddAction** method. This method provides a way for you to create interactive notifications. By default, users have the option to dismiss the notifications. However, there might be cases where you want to provide users with different actions that they can take to address the notification, like opening an associated page for modifying data.

Conceptually, a notification action calls a method in a specified codeunit, passing the notification object in the call. The method includes the business logic for handling the action.

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.AddAction('Action 1',Codeunit::"Action Handler",'RunAction1');
MyNotification.AddAction('Action 2',Codeunit::"Action Handler",'RunAction2');
MyNotification.Send();
```

The basic steps for adding an action are as follows:

1. Create a global method in a new or existing codeunit. The method must have a **Notification** data type parameter for receiving the notification object.
2. Add AL code to the method for handling the action.
3. Specify the codeunit and method in the **AddAction** method call.

IMPORTANT

You can have more than one action on a notification. A LocalScope notification can have up to 3 actions. A GlobalScope notification can have up to 2 actions.

Sending data with a notification

You use the **SetData** and **GetData** methods to add data to a notification, which is typically needed when actions are invoked. The **SetData** method sets, or adds, data to the notification. The data is defined as text in a key-value pair. With the **GetData** method, you can then retrieve the data again.

The following code sets data for a notification:

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.SetData('Created',Format(CurrentDateTime,0,9));
MyNotification.SetData('ID',Format(CreateGuid(),0,9));
MyNotification.AddAction('Action 1',Codeunit::"Action Handler",'RunAction1');
MyNotification.AddAction('Action 2',Codeunit::"Action Handler",'RunAction2');
MyNotification.Send();
```

The following code gets the data for a notification:

```
DataValue := MyNotification.GetData('Created');
DataValue := MyNotification.GetData('ID');
```

Example

This simple example illustrates how notifications work and provides some insight into how you can use them. This example extends page **42 Sales Order** of the CRONUS International Ltd. demonstration database according to the following:

- The code compares a customer's balance with their credit limit. If the balance exceeds the credit limit, a notification is sent to the client.
- The notification includes an action, which has the caption **Change credit limit**, that opens page **21 Customer Card**. This enables the user to increase the credit limit.

To complete the example, follow these steps:

1. Create a page extension object that extends page **42 Sales Order**, and add the notification code on the **OnOpenPage** trigger.

```
pageextension 50100 CreditBalanceNotification extends "Sales Order"
{

    trigger OnOpenPage()
    var
        Customer: Record Customer;
        CreditBalanceNotification: Notification;
        OpenCustomer: Text;
        Text003: Label 'The current balance exceeds the credit limit.';
        Text004: Label 'Change credit limit';
    begin
        Customer.Get("Sell-to Customer No.");
        if Customer."Balance (LCY)" > Customer."Credit Limit (LCY)" then begin
            //Create the notification
            CreditBalanceNotification.Message(Text003);
            CreditBalanceNotification.Scope := NotificationScope::LocalScope;
            //Add a data property for the customer number
            CreditBalanceNotification.SetData('CustNumber', Customer."No.");
            //Add an action that calls the ActionHandler codeunit, which you define in the next step.
            CreditBalanceNotification.AddAction('Text004', Codeunit::"ActionHandler",
'OpenCustomer');
            //Send the notification to the client.
            CreditBalanceNotification.Send();
        end;
    end;
}
```

2. Create a codeunit called **ActionHandler** for handling the notification action. Add a global method called **OpenCustomer** that has a **Notification** data type parameter called **CreditBalanceNotification** for receiving the Notification object, and include the following code on the method:

```
codeunit 50100 ActionHandler
{
    trigger OnRun()
    begin

    end;

    procedure OpenCustomer(CreditBalanceNotification: Notification)
    var
        CustNumber: Text;
        CustNo: Text;
        CustRec: Record Customer;
        CustPage: Page "Customer Card";
    begin
        //Get the customer number data from the SetData() call.
        CustNo := CreditBalanceNotification.GetData(CustNumber);
        // Open the Customer Card page for the customer.
        if CustRec.Get(CustNo) then begin
            CustPage.SetRecord(CustRec);
            CustPage.Run();
        end else begin
            Error('Could not find Customer: ' + CustNo);
        end;
    end;
}
```

See Also

Notification Data Type
Developing Extensions
Getting Started with AL

Task Scheduler

2/17/2021 • 2 minutes to read • Edit Online

The task scheduler enables you to control when certain operations or processes (in other words *tasks*) are run. Basically, a task is a codeunit or report that is scheduled to run at a specific data and time. Tasks run in a background session between the Dynamics 365 Business Central service instance and database. Behind the scenes, the task scheduler is used by the job queue to process job queue entries that are created and managed from the clients.

In AL code, you create and manage tasks by using the AL methods that are available for the **TASKSCHEDULER** data type.

METHOD	DESCRIPTION	FOR MORE INFORMATION, SEE
CreateTask	Adds a task to run a codeunit at a specified date and time.	CreateTask Method
SetTaskReady	Sets a task to the **Ready** state. A task cannot run until it is **Ready**.	SetTaskReady Method
TaskExists	Checks whether a specific task exists.	TaskExists Method
CancelTask	Cancels a scheduled task.	CancelTask Method

How task scheduler works

To set up a task, you create a codeunit that contains the logic that you want to run at a scheduled time. Optionally, you can create a second codeunit that contains the logic to handle the task if an error occurs for any reason. This codeunit is referred to as a *failure codeunit*. Once you have the codeunits, you can add AL code to the application that calls the CREATETASK method to schedule a task to run the codeunits. The CreateTask method can also specify the earliest date to run the task, and whether the task is in the ready state.

Task flow

Here is an overview of the process that a task goes through:

1. After you add a task, the task is recorded in table **2000000175 Scheduled Task** of the database.
2. If the task is in the ready state, when the scheduled time occurs, a new background session is started and the task codeunit is run.

   You can view the session in the table **2000000111 Session Event**.
3. If an error occurs, the following happens:

   a. If a failure codeunit is not specified, then the retry flow is initiated.

   b. If a failure codeunit has been specified, the error is passed in a call to the failure codeunit, and the failure codeunit is run.

      If the failure codeunit does not handle the error or fails itself, then the retry flow is initiated.

Error conditions and retry process

A task can fail under the following conditions:

- The company cannot be opened.
- A SQL connection or transient error occurred with the database.
- The Dynamics 365 Business Central service instance restarted while the task was being run.

When an error occurs, unless the task is interrupted by the failure codeunit, the server instance will rerun the task according to the following retry flow:

1. Two minutes after the first failure.
2. Four minutes after the second failure.
3. Fifteen minutes after the third failure and any subsequent failures up to a maximum of 10 times, after which the task is canceled.

About task sessions and permissions

The task runs in a background session, which means that there is no user interface. The behavior is similar to that of the STARTSESSION method, where any dialog boxes that would normally appear are suppressed. For more information about specific dialog boxes, see StartSession method.

The session runs by using the same user/credentials that are used when calling AL code. The user must have appropriate permissions to the codeunit and any other objects that are associated with the operation of the codeunit.

See Also

Task Scheduler Data Type
Developing Extensions
Getting Started with AL

Using App Key Vaults with Business Central Extensions

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Some Business Central extensions make web service calls to non-Business Central services. For example, one extension might call Azure Storage to read/write blobs. Another extension might call the extension publisher's web service to do an operation.

These web service calls are typically authenticated, which means the extension must provide a credential in the call. The credentials enable the other service to accept or reject the call. You can consider the credentials as a kind of secret to the extension. A secret shouldn't be leaked to customers, partners, or anybody else. So where can the extension get the secret from? Here is where Azure Key Vault is used. Azure Key Vault is a cloud service that works as a secure secrets store. It provides centralized storage for secrets, enabling you to control access and distribution of the secrets.

> **NOTE**
> For Business Central online, the app key vault feature is only supported for AppSource extensions.

Getting started

Getting extensions to use secrets from Azure Key Vault involves two areas of work: setting up and configuring Azure Key Vaults and developing the extensions to use secrets from Azure Key Vault.

Setting up and configuring Azure Key Vaults

An extension can retrieve secrets from one or two different Azure Key Vaults. These key vaults must be created in Azure, and the Business Central service configured to access key vaults. The setup process is different for online and on-premises. For more information, see:

- Setting up App Key Vaults for Business Central online
- Setting up App Key Vaults for Business Central on-premises

Developing the extensions to use secrets from Azure Key Vault

Once you have an Azure Key Vault, you can develop Business Central extensions to retrieve secrets from the key vault. In short, this work involves specifying the key vault's URL and adding code to retrieve a secret from the key vault.

For more information, see Using App Key Vault Secrets in Extensions.

See Also

Security Considerations With App Key Vaults
Monitoring and Troubleshooting App Key Vaults
Configuring Business Central Server - Azure Key Vault Extensions

Setting up App Key Vaults for Business Central Online

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

AppSource apps for Business Central can be developed to get secrets from Azure Keys Vaults. The app key vault feature is readily available for use on the service by all App Source apps. However, there are some onboarding tasks required.

> **IMPORTANT**
> With Business Central online, App key vaults can only be used with AppSource apps. They're not supported with per-tenant extensions.

> **TIP**
> You must also specify secrets in a key vault if you deploy Business Central as part of the Embed App program. Especially if you must support the Outlook add-in, in which case you must specify secrets for TEMPORARYDOCUMENTSTORAGEACCOUNT and TEMPORARYDOCUMENTSTORAGEKEY.

For more information about developing extensions with key vaults, see Using Key Vault Secrets in Business Central Extensions.

Create the Azure Key Vault with secrets

In this task, you create a key vault in Azure, and add the secrets that you want to make available to your extensions. An extension can use up to two key vaults, so you can create more than one.

There are different ways to create an Azure key vault. For example, you can use the Azure portal, Azure CLI, and more.

The easiest way is to use the Azure portal. For instructions, see Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal.

For using other methods, see Azure Key Vault Developer's Guide.

Provision the key reader application in your Azure AD tenant

Your Business Central online solution is configured to use an Azure AD application for reading key vault secrets. The application is called **Dynamics 365 Business Central ISV Key Vault Reader**. Microsoft manages the key vault reader application, however, there are a couple tasks that you have to do to enable it. First, the application must be provisioned on your Azure AD tenant, as described here.

To provision the key vault reader application, use the Azure Active Directory PowerShell module.

1. Open Windows PowerShell as an administrator.
2. Install the Azure Active Directory PowerShell module.

```
Install-Module AzureAD
```

3. Import the Azure AD module.

```
Import-Module AzureAD
```

4. Connect to your Business Central Azure AD tenant.

   a. Run the following command:

   ```
   Connect-AzureAD
   ```

   b. Provide your sign-in name and password when prompted.

5. Create an Azure AD service principal using the following command:

```
New-AzureADServicePrincipal -AppId 7e97dcfb-bcdd-426e-8f0a-96439602627a
```

`7e97dcfb-bcdd-426e-8f0a-96439602627a` is the Application (client) ID of Microsoft's centralized Azure AD application.

This step provisions the application in your Azure AD tenant, where it now "lives" together with your key vaults.

Grant the key vault reader application permission to your key vaults

The next task is to grant the key vault reader application permission to read secrets from your key vaults. The steps in this task are done from the Azure portal.

1. Open the key vault in the portal.
2. Select **Access policies**, then **Add Access Policy**.
3. Set **Secret Permissions** to **Get**.
4. Choose **Select principal**, and on the right, search for either the application (client) ID **7e97dcfb-bcdd-426e-8f0a-96439602627a** or the display name **Dynamics 365 Business Central ISV Key Vault Reader**.
5. Select **Add**, then **Save**.

Contact Microsoft to enable the App Key Vault feature

Send an email to bcappkeyvaultonboard@microsoft.com to start the onboarding process. Do this step before you publish your updated extension to Partner Center.

The onboarding process involves a manual verification step that verifies that you own the AAD tenant that contains the key vaults.

Provide the following information in the email:

- Your AAD tenant ID. Obtain this information from the Azure portal by going to the Azure Active Directory Overview page.
- Your AppSource extensions, including names and App IDs, that should be enabled to read secrets from your key vaults.
- Optionally, a screenshot from the Azure portal showing the key vault and its access policies. The screenshot can help Microsoft catch configuration mistakes early in the process.

See Also

Security Considerations With App Key Vaults
Monitoring and Troubleshooting App Key Vaults
Configuring Business Central Server

Setting up App Key Vaults for Business Central On-premises

2/17/2021 • 7 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Business Central extensions can be developed to get secrets from Azure Keys Vaults. This article describes the tasks required to set up Azure Keys Vaults for storing extension secrets and configure them in your Business Central deployment.

For more information about developing extensions with key vaults, see Using Key Vault Secrets in Business Central Extensions.

Prerequisites

To complete the tasks in this article, you need:

- An Azure subscription with an Active Directory tenant.

  You sign up for an Azure subscription at https://azure.microsoft.com. For information about getting an Azure AD tenant, see How to get an Azure Active Directory tenant.

- A security certificate

  Later, you'll have to register and configure an application in Azure AD for reading key vaults. This step requires a certificate. The certificate is used to prove the application's identity when requesting upon request. For a production environment, obtain a certificate from a certification authority or trusted provider.

  In a test environment, if you don't have a certificate, then you can create your own self-signed certificate. For example, on the Business Central server computer, start Windows PowerShell as an administrator. Then at the prompt, run the following commands:

  ```
  $cert = New-SelfSignedCertificate -Subject "BusinessCentralKeyVaultReader" -Provider "Microsoft
  Strong Cryptographic Provider"
  $cert.Thumbprint
  Export-Certificate -Cert $cert -FilePath c:\certs\BusinessCentralKeyVaultReader.cer
  ```

  These commands add a certificate called BusinessCentralKeyVaultReader to the computer's **LocalMachine** > **Personal (My)** certificate store.

Create the Azure Key Vault with secrets

Now, you create one or more key vaults in Azure, and add the secrets that you want to make available to your extensions. An extension can be set up with one or two key vaults.

There are different ways to create an Azure key vault. For example, you can use the Azure portal, Azure CLI, and more.

- The easiest way is to use the Azure portal. For instructions, see Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal.

  After you create the key vault, add the secrets.

- For using other methods, see Azure Key Vault Developer's Guide.

Register a key vault reader application in Azure AD

Next, register an application on your Azure AD tenant for reading secrets from the key vaults. When Azure AD authentication was set up, an Azure AD tenant was created in Azure. Reading key vaults requires a separate application registration with the Azure AD tenant. You can use an existing application.

The steps in this task are done from the Azure portal.

1. Sign in to Azure portal at portal.azure.com and set the portal to your Azure Active Directory tenant.
2. Register an Azure AD application for the reading key vault.

   You add the new application by using the Azure portal. For guidelines, see Register your application with your Azure Active Directory tenant.

   When you add an application to an Azure AD tenant, you must specify the following information:

SETTING	DESCRIPTION
Name	The name of your application as it will display to your users, such as *Business Central Key Vault Reader*.
Supported account types	Specifies which accounts that you would like your application to support. For purposes of this article, select **Accounts in this organizational directory only**.

   When completed, the **Overview** displays in the portal for the new application.

   Copy the **Application (client) ID**. You'll use this information later.
3. Upload the security certificate to the registered application.

   In this step, you upload the certificate file that you obtained as part of the prerequisites.

   Go to the key vault reader application overview page. Select **Certificates & secrets** > **Upload certificate**. Follow the instructions to locate and upload the certificate.

Grant the key vault reader application permission to key vaults

In this task, you grant the key vault reader application permission to read secrets from your key vaults.

The steps in this task are done from the Azure portal.

1. Open the key vault in the portal.
2. Select **Access policies**, then **Add Access Policy**.
3. Set **Secret Permissions** to **Get**.
4. Choose **Select principal**, and on the right, search for either **Application (client) ID** or display name for the key vault reader application.
5. Select **Add**.
6. Select **Save**.

At this point, the work in Azure is finished.

Configure the Business Central Server to use the Apps Key Vault feature

Next, you configure the Business Central Server instance to use the key vault reader application and its certificate, which you registered in Azure AD, for authenticating to the key vaults.

If you're running a container-based environment, you have two options for configuring the server instance. You can either do it manually or use the Set-BcContainerKeyVaultAadAppAndCertificate script. Using the Set-BcContainerKeyVaultAadAppAndCertificate script is simpler and recommended.

Configure a container-based Business Central Server instance

If you are running a container-based environment, use the Set-BcContainerKeyVaultAadAppAndCertificate.ps1 script that is available in the NAV Container Helper GitHub repository at https://github.com/microsoft/navcontainerhelper/blob/master/ContainerHandling/Set-BcContainerKeyVaultAadAppAndCertificate.ps1.

Manually configure a Business Central Server instance

To complete this task, you'll need the user name of the service account that runs the Business Central Server.

1. If not already done, import your key vault certificate and its private keys to the local certificate store for the Business Central server computer.

   You can import the certificate either using the MMC snap-in or Import-PfxCertificate cmdlet from a Windows PowerShell prompt.

   For example, the following PowerShell command installs a certificate to the local machine's personal store:

   ```
   Import-PfxCertificate -FilePath "C:\certificates\BusinessCentralKeyVaultReader.pfx" -Password
   (ConvertTo-SecureString -String "pfxpassword" -AsPlainText -Force) -CertStoreLocation
   Cert:\LocalMachine\My\
   ```

2. Give the service account used by the Business Central Server instance permission to access the certificates private key.

   To do this using the MMC:

   a. Open the MMC snap-in for certificates. See How to: View Certificates with the MMC Snap-in.

   b. Expand the **Certificates (Local Computer)** node, expand the **Personal** node, and then select the **Certificates** subfolder.

   c. In the right pane, right-click the certificate, select **All Tasks**, and then choose **Manage Private Keys**.

   d. In the **Security** dialog box, choose **Add**.

   e. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, enter the name of the dedicated domain user account that is associated with Business Central Server, for example, NETWORK SERVICE. Then, choose the **OK** button.

   f. In the **Full Control** field, select **Allow**, and then choose the **OK** button.

3. Make a note of the certificate thumbprint because you'll need it in the next step. See How to: Retrieve the Thumbprint of a Certificate.

4. Configure the Business Central Server instance.

   Now, you'll configure App Key Vault settings on the server instance. The following table describes the settings that you must configure:

SETTING (KEY NAME)	VALUE
Client Certificate Store Location (AzureKeyVaultClientCertificateStoreLocation)	Set to the certificate store location where key vault certificate was stored. Example: LocalMachine
Client Certificate Store Name (AzureKeyVaultClientCertificateStoreName)	Set to the certificate store name where key vault certificate was stored. Example: MY
Client Certificate Thumbprint (AzureKeyVaultClientCertificateThumbprint)	Set to the thumbprint for the key vault certificate. Example: 649419e4fbb87340f5a0f995e605b74c5f6d943e
Client ID (AzureKeyVaultClientId)	Set to the **Application (client) ID** of the key vault reader application registered in your Azure AD tenant. Example: ed4129d9-b913-4514-83db-82e305163bec
Enable Publisher Validation (AzureKeyVaultAppSecretsPublisherValidationEnabled)	Specifies whether extensions can only use key vaults that belong to their publishers. Enabling this setting (`true`) blocks attempts in AL to read secrets from another publisher's key vault. When extensions that use key vault secrets are published, you must provide your Azure AD tenant ID, which is done by using the Publish-NAVApp cmdlet with the `-PublisherAzureActiveDirectoryTenantId` parameter. **Important** We recommend that you only set it to `false` if you trust all extensions that will be installed. For more information, see App Key Vaults - Security considerations. Example: true

You can configure the instance using the Business Central Server Administration tool or Set-NAVServerConfiguration cmdlet.

To use the Set-NAVServerConfiguration cmdlet, start the Business Central Administration Shell as an administrator, and run the following commands one at a time. Replace brackets with your own values.

```
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultClientCertificateStoreLocation -KeyValue <certificate store location>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultClientCertificateStoreName -KeyValue <certifcate store>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultClientCertificateThumbprint -KeyValue <certificate thumbprint>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName AzureKeyVaultClientId -KeyValue
<application ID of key vault reader app in Azure>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultAppSecretsPublisherValidationEnabled -KeyValue <true|false>
Restart-NAVServerInstance -ServerInstance <serverInstance>
```

At this point, you can run your extensions that use key vault secrets to read secrets from key vault.

TIP

If your on-premises solution uses the ImportStreamWithUrlAccess method, you must have set up an Azure blob storage account and stored the account name and account keys in the current subscription's Azure KeyVault using the identifiers TEMPORARYDOCUMENTSTORAGEACCOUNT and TEMPORARYDOCUMENTSTORAGEKEY. That way, your users can use the integration with Outlook.

See Also

Using App Key Vaults with Business Central Extensions
Security Considerations With App Key Vaults
Monitoring and Troubleshooting App Key Vaults
Authentication and Credential Types
Configuring Business Central Server

Using Key Vault Secrets in Business Central Extensions

2/17/2021 • 6 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

This article describes how to code an extension to retrieve secrets from Azure Key Vaults. Secrets are typically used when the extensions calls a web service. For an overview of app key vaults and secrets, see Using App Key Vaults with Extensions.

Developing an extension to use secrets from a key vault involves two tasks, as described in this article:

- Specifying the Azure Key Vault in the extension's manifest.
- Adding code to retrieve the secrets from the key vault.

Preparation

- Using secrets requires that you have at least one Azure Key Vault with secrets set up and configured for use by the service. If you don't already have an Azure Key Vault, see Setting up App Key Vaults for Business Central online or Setting up App Key Vaults for Business Central on-premises.
- For coding, you'll need the URI the Azure Key Vault that stores the secret and the name of the secret itself. If you don't have this information, you can get it from Azure portal. For instructions, see Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal.

Specify the Azure Key Vault in extensions

You specify the key vaults for an extension in the extension's manifest file (app.json). To specify a key vault, you add the `"keyVaultUrls"` to the app.json, and set the value to the key vault's URL. For example, the following code snippet specifies a key vault that has the URI `https://mykeyvault.vault.azure.net`:

```
"keyVaultUrls":[
"https://mykeyvault.vault.azure.net"
]
```

You can specify up to two key vaults in the app.json, as shown in the following example code snippet:

```
"keyVaultUrls":[
"https://myfirstkeyvault.vault.azure.net",
"https://mysecondkeyvault.vault.azure.net"
]
```

Specifying two key vaults ensures a higher availability of secrets, especially if created in two different Azure regions. At runtime, the Business Central platform will iterate both key vaults until the secret is successfully retrieved. If one of the key vaults is unavailable for any reason, the extension will continue to execute because the other key vault will most likely be available.

Add code to retrieve secrets from the key vault

Next, you add code to the extension for reading secrets from the key vault at runtime. To read secrets, you use

the **Secrets** module of the System Application. Specifically, you'll use codeunit **3800 "App Key Vault Secret Provider"**. This codeunit includes two methods:

METHOD	DESCRIPTION
`TryInitializeFromCurrentApp(): Boolean`	Identifies the calling extension and initializes the codeunit with the key vaults specified in the extension's manifest.
`GetSecret(SecretName: Text, var SecretValue: Text): Boolean`	Retrieves the value of a specific secret from one of the app's key vaults.

Look at the following example for a simple page object. The code retrieves the value of the secret named **MySecret** in app key vault:

```
page 50100 HelloWorldPage
{
    var
        SecretProvider: Codeunit "App Key Vault Secret Provider";
        SecretValue: Text;

    trigger OnOpenPage();
    begin
        if SecretProvider.TryInitializeFromCurrentApp() then begin
            if SecretProvider.GetSecret('MySecret', SecretValue) then
                Message('Retrievedsecret:' + SecretValue)
            else
                Message('Failed to retrieve secret')
        end
        else
            Message('ERROR:' + GetLastErrorText());
    end;
}
```

The call to the `TryInitializeFromCurrentApp` method determines the extension that is currently being executed, then determines the extension's key vaults as specified in the extension manifest. After initialization, the `GetSecret` call reads secrets from the key vault.

Security considerations

Keep the following information in mind when you use the App Key Vault feature with your extensions.

Mark methods as NonDebuggable

When your code works with secrets, whether from a key vault or from Isolated Storage, block the ability to debug relevant methods by using the NonDebuggable Attribute. It prevents other partners from debugging into your code and seeing the secrets.

Don't pass the App Key Vault Secret Provider to untrusted code

Once the **App Key Vault Secret Provider** codeunit has been initialized, it can be used to get secrets.

- If you pass the codeunit to another method, then that method is also able use it.
- If you pass the codeunit to a method in another extension, then the other extension can also use the secret provider to get secrets.

These conditions may not be what you want, so be careful who you pass the secret provider to.

Enable publisher validation

For on-premises deployments, you can configure Business Central Server to run with or without publisher validation of key vault secret providers. Publisher validation is controlled by the server's **Enable Publisher**

Validation (AzureKeyVaultAppSecretsPublisherValidationEnabled) configuration setting. The validation is a runtime operation that ensures extensions use only key vaults that belong to their publishers. It essentially blocks attempts in AL to read secrets from another publisher's key vault.

How it works

Publisher validation is done by comparing the key vault's Azure AD tenant ID with the extension publisher's Azure AD tenant ID. It works this way:

1. When an extension is published by using the Publish-NAVApp cmdlet, the publisher can provide their Azure AD tenant ID by setting the `-PublisherAzureActiveDirectoryTenantId` parameter:

   ```
   Publish-NavApp -ServerInstance <server instance> -Path <path to extension package> -
   PublisherAzureActiveDirectoryTenantId <Azure AD tenant ID GUID>
   ```

   > **NOTE**
   > An error won't occur if `-PublisherAzureActiveDirectoryTenantId` isn't set. There is nothing preventing you from publishing the extension at this point.

2. When the extension runs, it tries to initialize the **App Key Vault Secret Provider** codeunit.
3. The system compares the key vault's Azure AD tenant ID with the Azure AD tenant ID published with the extension:
   - If they match, initialization succeeds.
   - If they don't match, an error occurs.

Turning off publisher validation

Publisher validation is turned on by default, which is the recommended setting. If it's turned off, the server instance won't do any additional validation to ensure extensions have the right to read secrets from the key vaults that they specify. This condition implies some risk of unauthorized access to key vaults that you should be aware of. So, don't turn off publisher validation unless you trust the extensions that can be potentially installed.

For information about how to turn publisher validation on or off, see Configuring Business Central Server.

Monitoring and troubleshooting

Compiling and publishing

If you get errors when you compile or publish your extension, the most likely reasons are:

- You're using an old Visual Studio Code AL extension. Upgrade to the latest AL extension.
- Your extension targets an older runtime. Make sure that the `"runtime"` value in the app.json file is at least `"6.0"`.
- You're running an old version of Business Central Server. Upgrade to at least version 17.0.

Runtime

For runtime operations, there are two sources that you can use for gathering details:

- Windows Event Log of the machine running the Business Central Server.
- Application Insights.

These sources provide details about retrieving secrets from key vaults, for calls to the `TryInitializeFromCurrentApp` and `GetSecret` methods from an extension.

Using Application Insights

You can set up extensions to emit telemetry to an Application Insights resource in Azure.

1. Create an Application Insights resource in Azure if you don't have one.

   The Application Insights resource will be assigned an instrumentation key. Copy this key because you'll need it to enable Application Insights in the Business Central administration center.

   For more information, see Create an Application Insights resource.

2. In the app.json file of the extension, add the `"applicationInsightsKey"`:

```
"applicationInsightsKey":["<instrumenation key>"]
```

3. Now, you can run your extensions and view data in Application Insights.

   For more information, see Viewing telemetry data in Application Insights and Analyzing App Key Vault Secret Trace Telemetry.

See Also

Getting Started with AL
Publishing and Installing Extensions
Configuring Business Central Server

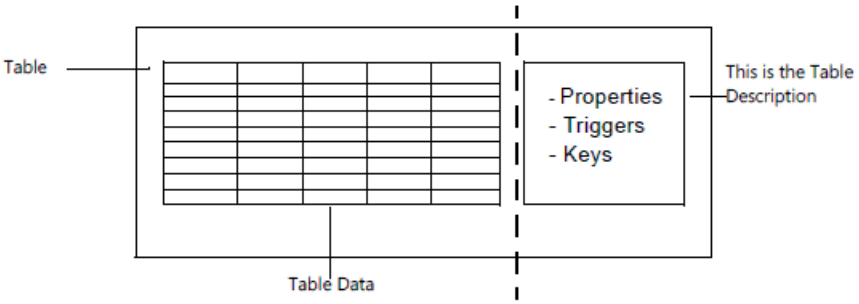
Tables Overview

2/17/2021 • 2 minutes to read • Edit Online

Tables are the fundamental objects in any database. They are the objects in which you store and manipulate data. This is true no matter what kind of data you need to manage. When you create a new database, you begin by building the tables. Later, you create pages and reports in order to access and view the data in the tables.

A table can be visualized as a two-dimensional matrix, consisting of columns and rows. The following illustration shows a table where each row is a record and each column is a field.

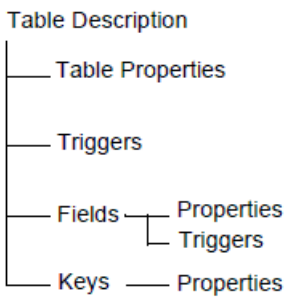
A table consists of two parts: the table data and a table description. The table data is the part users often think of as comprising the database, because it contains the actual records with their data fields. The layout and properties of those fields, however, are specified by the table description. The table description is not directly visible to the user. The following illustration shows how the table data and the table description together form a table.



When you design a table, you assign a number of characteristics to it, such as a name, an ID number, and the fields it contains. You also assign a number of characteristics (such as name, ID number, data type, and initial value) to each field. When you design a new table, you also specify which keys you want the system to maintain. All these characteristics are stored in the table description when you save your table design.

The information in the table description is used by SQL Server and occasionally by database users who need information about the table structure. The table description makes the database flexible, as it lets the system access tables with different structures. The database can extract the definitions of the table structure from the table description and thereby correctly access any table.

The following illustration shows that a table description contains properties, triggers, fields, and keys and shows how these are related.



The table description contains some properties that are related to the table, others that are related to the fields in the table, and other properties related to keys. You can also see that triggers are defined both for the table and for the fields in the table.

Creating tables

In AL code, you can create new tables or modify existing tables. Read more about creating and modifying tables in the following sections.

TO	SEE
Create a new table object	Table Object
Modify an existing table object	Table Extension Object
Decide which field data type you want to apply to your data	Field Data Types
Apply table and field properties	Table and Table Extension Properties
Set primary and secondary table keys	Table Keys

Using triggers in database design

Dynamics 365 Business Central supports setting up actions to take place in response to specific events. These are known as triggers. The following topics help to explain how Dynamics 365 Business Central implements this feature of database design.

TO	SEE
Learn about the set of triggers that Dynamics 365 Business Central supports for tables and fields.	Table and Field Triggers

Creating relationships between tables

In Dynamics 365 Business Central, the primary way to establish a connection between tables is to use the **TableRelation** property. The following topics go into detail about how this works.

TO	SEE
Get a brief introduction to relational database design in Dynamics 365 Business Central.	Setting Relationships Between Tables

See Also

Developing Extensions in AL

Table Object

2/17/2021 • 2 minutes to read • Edit Online

Tables are the core objects used to store data in Dynamics 365 Business Central. No matter how data is registered in the product - from a web service to a finger swipe on the phone app, the results of that transaction will be recorded in a table.

The structure of a table has four sections:

- The first block contains metadata for the overall table, such as the table type.
- The fields section describes the data elements that make up the table, such as their name and the type of data they can store.
- The keys section contains the definitions of the keys that the table needs to support.
- The final section details the triggers and code that can run on the table.

IMPORTANT

Only tables with the Extensible Property set to **true** can be extended.

NOTE

Extension objects can have a name with a maximum length of 30 characters.

IMPORTANT

System and virtual tables can't be extended. System tables are created in the ID range of 2.000.000.000 and above. For more information about object ranges, see Object Ranges.

Snippet support

Typing the shortcut `ttable` will create the basic layout for a table object when using the AL Language extension in Visual Studio Code.

TIP

Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Table example

This table stores address information and it has four fields; `Address`, `Locality`, `Town/City`, and `County`.

```
table 50104 Address
{
    Caption = 'Sample table';
    DataPerCompany = true;

    fields
    {
        field(1; Address; Text[50])
        {
            Description = 'Address retrieved by Service';
        }
        field(2; Locality; Text[30])
        {
            Description = 'Locality retrieved by Service';
        }
        field(3; "Town/City"; Text[30])
        {
            Description = 'Town/City retrieved by Service';
        }
        field(4; County; Text[30])
        {
            Description = 'County retrieved by Service';

            trigger OnValidate();
            begin
                ValidateCounty(County);
            end;

        }
    }
    keys
    {
        key(PrimaryKey; Address)
        {
            Clustered = TRUE;
        }
    }

    var
        Msg: Label 'Hello from my method';

    trigger OnInsert();
    begin

    end;

    procedure MyMethod();
    begin
        Message(Msg);
    end;
}
```

System fields

The Dynamics 365 Business Central platform will automatically add several system fields to tables. For more information, see System Fields.

See Also

AL Development Environment
Table Overview
Table Extension Object
SqlTimestamp Property

Table Keys
Table, Table Fields, and Table Extension Properties

System Fields

2/17/2021 • 5 minutes to read • Edit Online

System fields are fields that are automatically included in every table object by the platform. Dynamics 365 Business Central includes the following system fields:

- SystemId
- Data audit fields
- Timestamp

System fields are assigned numbers in the range 2000000000-2147483647. This range is reserved for system fields. You'll get a design-time error if you give a field a number in this range.

SystemId field

APPLIES TO: Business Central 2019 release wave 2 and later

The **SystemId** field is a GUID data type field that specifies a unique, immutable (read-only) identifier for records in the table. The **SystemId** field has the following characteristics and behavior:

- All records must have a value in the **SystemId** field.
- You can assign your own value when a record is inserted in the database. Otherwise, the platform will automatically generate and assign a value.
- Once the **SystemId** has been set, it can't be changed.
- There's always a unique secondary key on the **SystemId** field to ensure records don't have identical field values.
- The **SystemId** field is given the field number 2000000000.

The **SystemId** field is exposed in the platform code and for AL code, allowing you to code against it. For example:

- The Insert(Boolean, Boolean) lets you specify the **SystemId** value for a record, instead of using one assigned by the platform:

```
myRec.SystemId := '{B6666666-F5A2-E911-8180-001DD8B7338E}';
myRec.Insert(true, true);
```

- The GetBySystemId(Guid) uses the **SystemId** to get a record:

```
id := myRec.GetBySystemId('{B6666666-F5A2-E911-8180-001DD8B7338E}';
```

- The SystemIdNo() gets the field number used by the **SystemId** field in the table:

```
myRec.OPEN(DATABASE::MyTable);
SystemIdFieldNo := myRec.SystemIdNo();
```

- The TableRelation lets you use the **SystemId** field to set up table relationships:

```
field(1; MyField; Guid)
{
    DataClassification = ToBeClassified;
    TableRelation = Customer.SystemId;
}
```

- The **SystemId** field can be used as a binding key to an API:

```
page 50100 "Customer Entity"
{
    PageType = API;
    ApplicationArea = All;
    UsageCategory = Administration;
    SourceTable = Customer;
    ODataKeyFields = SystemId;
    ...
```

APPLIES TO: Business Central 2020 release wave 2 and later

- If the **SystemId** field is specified in a Web Service POST request, the OData stack persists the value in the database.
- The **SystemId** field can be used as part of a (non-primary) key.
- You can show the **SystemId** field as a field on a page.
- You can link FactBoxes/page parts using the **SystemId** field.

Data audit fields

APPLIES TO: Business Central 2020 release wave 2 and later

Every table in Business Central includes the following four system fields, which can be used for auditing records:

FIELD NAME (IN AL)	COLUMN NAME (IN DATABASE)	DATA TYPE	FIELD NUMBER	DESCRIPTION
SystemCreatedAt	$systemCreatedAt	DateTime	2000000001	Specifies the data and time that the record was created.
SystemCreatedBy	$systemCreatedBy	GUID	2000000002	Specifies security ID (SID) of the user that created the record.
SystemModifiedAt	$systemModifiedAt	DateTime	2000000003	Specifies the data and time that the record was last modified.
SystemModifiedBy	$systemModifiedBy	GUID	2000000004	Specifies the SID of the user that last modified the record

Runtime characteristics

At runtime, the data audit fields have the following characteristics and behavior:

The platform will automatically generate and assign values according to the following triggers:

- After all OnBeforeInsert and OnBeforeModify triggers are run

- After the OnInsert and OnModify triggers are run
- Before all OnAfterInsert and OnAfterModify triggers are run

> **NOTE**
> You can assign the values, but the values written to the database are always provided by the platform.

Fields are populated as follows:

- When a new record is created, before calling Insert, the audit fields are given blank GUIDs and blank dates as values.
- When a record is first inserted, the fields are populated with actual values.

  The $systemCreatedBy and $systemModifiedBy fields are given the same value. So are the $systemCreatedAt and $systemModifiedAt fields.

  The $systemCreatedBy and $systemCreatedAt fields won't change after this point.
- When a record is updated, the $systemModifiedBy and $systemModifiedAt fields are changed.

The platform won't populate audit field values in these cases:

- Copying company. The values in the tables of the company being copied stay the same, and the values are copied to the tables of the new company.
- Synchronizing the table schema with the application.

> **NOTE**
> Audit fields can't be imported with configuration packages.

In AL

The data audit fields are exposed in AL code. As a developer, the audit fields give you an easy and performant way to program against historical data. For example, you can write AL queries that return data changes since a specific date and time.

The following methods are available on the RecordRef data type:

METHOD	DESCRIPTION
SystemCreatedAtNo	Gets the field number that is used by the **SystemCreatedAt** field.
SystemCreatedByNo	Gets the field number that is used by the **SystemCreatedBy** field.
SystemModifiedAtNo	Gets the field number that is used by the **SystemModifiedAt** field.
SystemModifiedByNo	Gets the field number that is used by the **SystemModifiedBy** field.

There are a couple points of interest you should know:

- If a record is copied into a temporary table, the data audit field values are copied as well. The values aren't changed by the server when calling a modify or insert method.

- It's possible to use audit fields in a key. The platform doesn't automatically index these fields in any way.

Timestamp field

The **timestamp** field contains row version numbers for records, as maintained in SQL Server. The **timestamp** field is hidden. But, you can expose it by using SqlTimestamp Property. You're then able to write code against it, add filters, and so on, similar to any other field in a table. However, you can't write to the **timestamp** field.

A typical use of the **timestamp** field is for synchronizing data changes in tables. It lets you identify records that have changed since the last synchronization. For example, you can read all the records in a table, then store the highest **timestamp** value. Later, you can query and retrieve records that have a higher **timestamp** value than the stored value.

Expose the timestamp field

1. Add a field that has the data type `BigInteger`.

   Specify a name for the field, such as `Record Time Stamp`. You can specify any valid name for field, but you can't use `timestamp` because it's a reserved name.

2. Set the `SqlTimestamp` property to `true`.

   For example:

```
field(3; "Record Time Stamp"; BigInteger)
{
    SqlTimestamp = true;
}
```

Alternatively, you can use a FieldRef Data Type variable to access the timestamp value of a record, as follows:

1. Create a RecordRef Data Type variable that references the record in a table for which you want to retrieve its timestamp.

2. Use the Field Method on the **RecordRef** variable to get the **FieldRef** for the field that has the number 0. This field contains the timestamp value.

The following example shows how to retrieve the timestamp value for the first record in the `Customer` table. **RecordRef** and **FieldRef** are RecordRef Data Type and FieldRef Data Type variables, respectively.

```
RecordRef.Open(DATABASE::Customer);
RecordRef.FindFirst();
FieldRef := RecordRef.Field(0);
Message(Format(FieldRef.Value()));
```

See Also

Table Object
AL Development Environment
Table Overview
Table Extension Object
SqlTimestamp Property
Table Keys
Table Properties

Table Extension Object

2/17/2021 • 2 minutes to read • Edit Online

The table extension object allows you to add additional fields or to change some properties on a table provided by the Dynamics 365 Business Central service. In this way, you can add data to the same table and treat it as a single table. For example, you may want to create a table extension for a retail winter sports store. In your solution you want to have `ShoeSize` as an additional field on the customer table. Adding this as an extension allows you to write code for the customer record and also include values for the `ShoeSize`.

Along with defining other fields, the table extension is where you write trigger code for your additional fields.

When developing a solution for Dynamics 365 Business Central , you will follow the code layout for a table extension as shown in the example below.

IMPORTANT

Only tables with the Extensible Property set to **true** can be extended.

NOTE

Extension objects can have a name with a maximum length of 30 characters.

IMPORTANT

System and virtual tables cannot be extended. System tables are created in the ID range of 2.000.000.000 and above. For more information about object ranges, see Object Ranges.

IMPORTANT

Extending tables from Dynamics 365 for Sales is currently not supported.

Snippet support

Typing the shortcut `ttableext` will create the basic layout for a table extension object when using the AL Language extension in Visual Studio Code.

Properties

Using a table extension allows you to overwrite some properties on fields in the base table. For a list of Table properties, see Table and Table Extension Properties.

Table extension syntax

```
tableextension Id MyExtension extends MyTargetTable
{
    fields
    {
        // Add changes to table fields here
    }

    var
        myInt: Integer;
}
```

Table extension example

This table extension object extends the Customer table object by adding a field `ShoeSize`, with ID 50116 and the data type `Integer`. It also contains a procedure to check if the `ShoeSize` field is filled in.

```
tableextension 50115 RetailWinterSportsStore extends Customer
{
    fields
    {
        field(50116;ShoeSize;Integer)
        {
            trigger OnValidate();
            begin
                if (rec.ShoeSize < 0) then
                begin
                   message('Shoe size not valid: %1', rec.ShoeSize);
                end;
            end;
        }
    }

    procedure HasShoeSize() : Boolean;
    begin
        exit(ShoeSize <> 0);
    end;

    trigger OnBeforeInsert();
    begin
        if not HasShoeSize then
            ShoeSize := Random(42);
    end;
}
```

See Also

AL Development Environment
Table Overview
Table Object
Table, Table Fields, and Table Extension Properties
Table Keys

Setting Relationships Between Tables

2/17/2021 • 2 minutes to read • Edit Online

It is common to distinguish among the following types of relationships between tables in relational database design:

- One-to-many relationships
- Many-to-many relationships
- One-to-one relationships

The one-to-many relationship is the most common. If your database design model indicates that you need to set up a many-to-many relationship, then your design is probably inefficient. You can typically break down a many-to-many relationship into two one-to-many relationships. A one-to-one relationship is usually not optimal and can often be avoided by combining the two tables.

Using relationships

If your database contains tables with related data, then you can define a relationship between them. You relate tables by specifying one or more fields that contain the same value in related records. These matching fields often have the same name in each table. You can use relationships to:

- Validate data entries
- Perform lookup functions in other tables
- Propagate changes automatically from one table to other tables

Table relationships and the TableRelation property

Table relationships are defined in the AL Language development environment using the **TableRelation** property. This property allows you to define both simple and advanced table relations.

> **NOTE**
> You can define a relationship only to a field that is a member of the primary key group.

Advanced table relations are typically prefixed with a conditional statement and include filters. The following syntax is for table relations.

```
<TableRelation> =
   <TableName>[.<FieldName>] [WHERE(<TableFilters>)] |
   if (<Conditions>) <TableName>[.<FieldName>]
   [WHERE(<TableFilters>)] else <TableRelation>
<Conditions> ::=
   <TableFilters>
<TableFilters> ::=
   [<TableFilter> {,<TableFilter>}]
<TableFilter> ::=
   <DstFieldName>=CONST(<FieldConst>) |
   <DstFieldName>=FILTER(<Filter>)
```

For example:

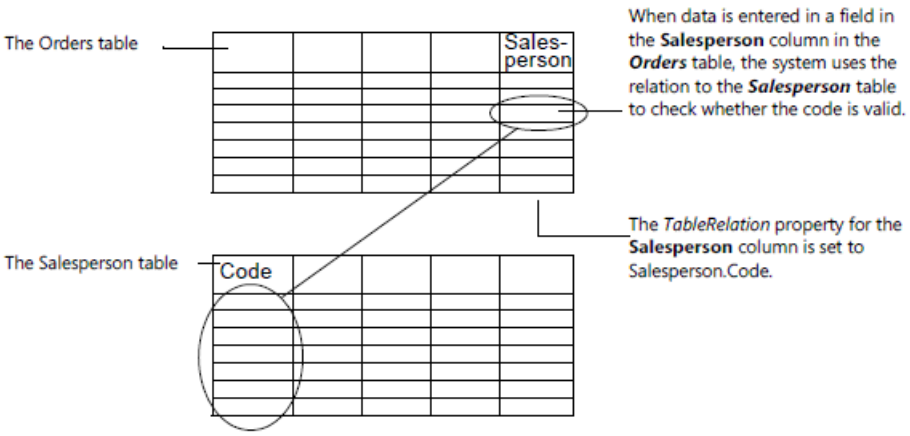
```
table 50120 TableWithRelation
{
  fields
  {
    field(1; Id; Integer) { }
    field(2; Type; enum TypeEnum) { }
    field(3; Relation; Code[20])
    {
      TableRelation =
        if (Type = const (Customer)) Customer
        else if (Type = const (Item)) Item;
    }
  }
}
```

The following table describes each of the symbols.

SYMBOL	DESCRIPTION
TableName	Specifies the related table.
FieldName	Specifies a field in the related table.
Conditions	Table relations can be conditional.
TableFilters	A list of table filters.
TableFilter	A constant expression or a filter expression.
DstFieldName	Specifies the destination field name.
Filter	A filter expression, such as 10|20..30.

Examples of table relationships

For example, you have an **Orders** table that stores orders and a **Salesperson** table that stores the names of all salespeople in your company. In the **Orders** table, you can include a **Salesperson** field that identifies the salesperson. By setting up a relationship between these two tables, you can check whether the **Salesperson** field in the **Orders** table contains a valid code.



For example, you have a **Vendors** table with all your vendors and a **Currency Code** table. You can create a relationship between a **Currency Code** field in the **Vendors** table and the **Currency Code** table. This will allow users to look up information about valid currency codes.

Furthermore, if you change one of the currency codes in the **Currency Code** table, then the change is automatically propagated to all tables that refer to this code.

See Also

Overview of Tables

Viewing Table Data

2/17/2021 • 2 minutes to read • Edit Online

For developers, administrators, and support personnel, it can be useful to inspect table data in the tenant database, particularly when debugging or troubleshooting. To support this need, you can view table objects in the Web client. This lets you to see the data in all rows and columns of a specific table, including any columns that are added by table extensions.

- In a production environment, administrators and support can view a table directly from the Web client.
- From the Business Central administration center, you can launch a list of all tables, sorted by storage size. For more information, see Storage usage by environment.
- In a development environment, in addition to viewing a table directly from the Web client, developers can view a table automatically when they publish/debug an AL project from Visual Studio Code.

> **NOTE**
> The table appears as read-only in the client, so modifications, insertions, and deletions cannot be made.

> **IMPORTANT**
> Data in the tables can be sensitive. Make sure that you follow your organization's guidelines for handling such data.

Required permissions

Whether viewing the table directly from the client or from Visual Studio Code, your Dynamics 365 user account must have the following permissions:

- Read permission on the table that you want to view.
- Execution permission (direct) on the System object **1350 Run table**.

Any end-user that is assigned these permissions will be able to view that table in the browser.

For information about assigning permissions, see Manage Users and Permissions.

View a table object directly from the client

To view a table, you add the `table=<TableID>` parameter to the client's address (URL), replacing `<TableID>` with the ID of the table that you want to view.

For example, if your URL starts with `https://businesscentral.dynamics.com`, then to view table **18 Customer** in your current company, you could use the following URL:

```
https://businesscentral.dynamics.com/?table=18
```

Or for a specific company, such as "CRONUS Inc.":

```
https://businesscentral.dynamics.com/?company=CRONUS%20Inc.&table=18
```

Note the use of `&` when `table=<TableID>` is not located directly after the domain name.

View a table object from an AL project in Visual Studio Code

You can configure an AL project to view a table when you publish or debug the project (pressing **F5** or **Ctrl+F5**).

In the `launch.json` file for the project, set the `"startupObjectType"` parameter to `"table"` and the `"startupObjectId"` parameter to the ID of the table. For example:

```
{
    "version": "1.0.0",
    "configurations": [
        {
            "type": "al",
            "request": "launch",
            "name": "Publish to Microsoft cloud sandbox",
            "serverInstance": "dynamics",
            "startupObjectType": "Table"
            "startupObjectId": 18
        }
    ]
}
```

For more information about the `launch.json` file, see Launch.json file.

Constraints

You cannot view virtual tables or the following system tables:

ID	NAME
2000000170	Configuration Package File
2000000173	Data Sensitivity
2000000100	Debugger Breakpoint
2000000103	Debugger Watch
2000000130	Device
2000000114	Document Service
2000000190	Entitlement Set
2000000191	Entitlement
2000000180	MediaSet
2000000181	Media

ID	NAME
2000000195	Membership Entitlement
2000000162	Nav App Capabilities
2000000152	Nav App Data Archive
2000000161	Application Dependency (*Nav App Dependencies)
2000000150	Application Object Metadata (*Nav App Object Metadata)
2000000163	Nav App Object Prerequisites
2000000142	Application Resource (*Nav App Resource)
2000000151	Installed Application (*Nav App TenantApp)
2000000160	Published Application (*Nav App)
2000000071	Object Metadata
2000000079	Object Tracking
2000000001	Object
2000000198	Page Documentation
2000000186	Profile Page Metadata
2000000082	Report Layout
2000000065	Send To Program
2000000112	Server Instance
2000000066	Style Sheet
2000000197	Token Cache
2000000081	Upgrade Blob Storage
2000000121	User Property
2000000076	Web Service
2000000194	Webhook Notification
2000000199	Webhook Subscription

NOTE

The tables marked with * in the table above changed names with Business Central 2020 release wave 1. For more information, see Deprecated Tables.

See Also

Developing Extensions
Deprecated Tables
Managing Capacity

Insert, Modify, ModifyAll, Delete, and DeleteAll Methods

2/17/2021 • 4 minutes to read • Edit Online

The following methods maintain the database by adding, modifying, and removing records:

- Insert
- Modify
- ModifyAll
- Delete
- DeleteAll

These methods are some of the most frequently used AL methods.

Some of these methods return an optional Boolean value that indicates whether the method succeeded. If you do not handle the return value in your code, a run-time error occurs when a method returns **false**. If you handle the return value by testing its value in an **if** statement, no error will occur, and you must take corrective action in the code.

Insert method

The Insert method inserts a record in a table. Insert has the following syntax.

```
[Ok := ]  Record.Insert([RunTrigger: Boolean[, InsertWithSystemId: Boolean]])
```

A record must be assigned a **SystemId**. You have the option to assign your own value or have the platform assign an auto-generated value. The following example inserts a new record, with the **SystemId**, **No.**, and **Name** fields specified in the assigned values, while other fields will have their default values. If the **No.** field is the primary key of the **Customer** table, then the record will be inserted in the **Customer** table unless the table already contains a record with the same primary key. In this case you receive an error message because the return value is not tested.

```
var
    Customer: Record Customer;
begin
    Customer.Init;
    Customer.SystemId := '{B6666666-F5A2-E911-8180-001DD8B7338E}';
    Customer."No." := '4711';
    Customer.Name := 'Andrew Dixon';
    Customer.Insert(false, true);
end;
```

IMPORTANT

After the **SystemId** has been set on a record, it cannot be changed.

Modify method

Modify modifies a record that already exists. For more information, see Modify Method. Modify has the

following syntax.

```
[Ok :=] Record.Modify([RunTrigger])
```

Modify returns an optional Boolean value. It returns **true** if the record to be modified exists; otherwise, it returns **false**.

The following example changes the name of customer 4711 to Richard Roe. This example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
Customer	Record	Customer

```
Customer.Get('4711');
Customer.Name := 'Richard Roe';
Customer.Modify;
```

ModifyAll method

ModifyAll performs a bulk update of records. For more information, see ModifyAll Method.

ModifyAll has the following syntax.

```
Record.ModifyAll(Field, NewValue [, RunTrigger])
```

ModifyAll uses the current filters. This means that you can perform the update on a specified set of records in a table. ModifyAll returns no value, nor does it cause an error if the set of records to be changed is empty.

In the following example, the `SetRange` statement selects the records where Salesperson Code is PS. The ModifyAll statement changes the Salesperson Code of these records to JR. The example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
Customer	Record	Customer

```
Customer.SetRange("Salesperson Code",'PS','PS');
Customer.ModifyAll("Salesperson Code",'JR');
```

Delete method

Delete deletes a record from the database. For more information, see Delete Method Delete has the following syntax.

```
[Ok :=] Record.Delete([RunTrigger])
```

The record that you want to delete must be specified by using the values in the primary key fields before you call this method. This means that Delete does take filters into consideration.

The following example shows how to use Delete to delete the record for customer number 4711. This example

requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
Customer	Record	Customer

```
Customer."No." := '4711';
Customer.Delete;
```

Delete returns an optional Boolean value. It returns **true** if the record could be found; otherwise, it returns **false**. Unless you test this value in your code, a run-time error occurs when Delete fails.

When you are developing your own applications, you should consider the following scenario:

1. Retrieve a record from the database.
2. Perform various checks to determine whether the record should be deleted.
3. If step 2 indicated that you should delete the record, then delete it.

This can cause problems in a multi-user environment. Another user can modify or delete the same record between your performing steps 2 and 3. If the record is modified, then perhaps the new contents of the record would have changed your decision to delete it. If it has been deleted by the other user, you can get a run-time error if you have just verified that the record existed (in step 1). If the design of your application indicates that you can encounter this problem, you should consider using the LockTable method. LockTable should be used sparingly because this method degrades performance. For more information about the LockTable method, see LockTable Method.

DeleteAll method

DeleteAll deletes all the records that are specified by the filter settings. If no filters are applied, it deletes all the records in the table. For more information, see DeleteAll Method DeleteAll has the following syntax.

```
Record.DeleteAll([RunTrigger])
```

The following example deletes all the records from the **Customer** table where the Salesperson Code is PS. This example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
Customer	Record	Customer

```
Customer.SetRange("Salesperson Code", 'PS', 'PS');
Customer.DeleteAll;
```

NOTE

When you use DeleteAll (true), a copy of the AL variable with its initial values is created. This means that when you use DeleteAll(true) to run the OnDelete trigger, all the changes that were made to the variables in the method or codeunit that is making the call cannot be seen in the OnDelete trigger. If you want to see the changes that you made to the variables, you must use Delete(true) in a loop. There is no difference in performance between using DeleteAll(true) and using Delete(true) in a loop.

See Also

AL Methods
SystemId Field

Get, Find, and Next Methods

2/17/2021 • 3 minutes to read • Edit Online

The following methods are used to search for records:

- `Get`
- `Find`
- `Next`

These methods are some of the most frequently used AL methods. When you search for records, you must know the difference between Get and Find and to know how to use Find and Next in conjunction.

> **TIP**
> When using these methods, consider using the partial records methods to improve performance, especially when looping through several records or when table extensions are defined on the table. For more information, see Using Partial Records.

Get method

The Get Method (Record) retrieves one record based on values of the primary key fields.

Get has the following syntax.

```
[Ok :=] Record.Get([Value],...)
```

For example, if the **No.** field is the primary key of the **Customer** table and if you have created a record variable called **CustomerRec** that has a subtype of Customer, then you can use Get in the following way.

```
CustomerRec.Get('4711');
```

The result is that the record of customer 4711 is retrieved.

Get produces a run-time error if it fails and the return value is not checked by the code. In the previous example, the actual code that you write should resemble the following.

```
if CustomerRec.GET('4711') then
.... // Do some processing.
else
.... // Do some error processing.
```

Get searches for a record, regardless of the current filters, and it does not change any filters. Get always searches through all the records in a table.

GetBySystemId method

APPLIES TO: Business Central 2019 release wave 2 and later

The GetBySystemId(Guid) retrieves a record based on the value of its **SystemId** field.

GetBySystemId has the following syntax:

```
RecordExists :=   Record.GetBySystemId(SystemId: Guid)
```

The following example gets the record that has the SystemId `5286305A-08A3-E911-8180-001DD8B7338E`:

```
var
    Customer: Record Customer;
    Text000: Label 'Customer was found.';
begin
    If Customer.GetBySystemId('{5286305A-08A3-E911-8180-001DD8B7338E}') then
    Message(Text000);
end;
```

Similar to the Get method, GetBySystemId searches for a record, regardless of the current filters, and it does not change any filters. Get always searches through all the records in a table.

Find methods

The Find Method (Record) locates a record in a table that is based on the values stored in the keys.

Find has the following syntax.

```
Ok := Record.Find([Which])
```

The *Which* parameter specifies how to perform the search. You can search for values that are greater than, less than, or equal to the key value, or for the first or last record in a table.

The important differences between Get and Find are as follows:

- Find uses the current filters.
- Find can look for records where the key value is equal to, greater than, or smaller than the search string.
- Find can find the first or the last record, depending on the sort order defined by the current key.

When you are developing applications in a relational database, there are often one-to-many relationships defined between tables. An example could be the relationship between an **Item** table, which registers items, and a **Sales Line** table, which registers the detailed lines from sales orders. One record in the **Sales Line** table can only be related to one item, but each item can be related to any number of sales line records. You would not want an item record to be deleted as long as there are still open sales orders that include the item. You can use Find to check for open sales orders.

The OnDelete trigger of the **Item** table includes the following code that illustrates using Find.

```
SalesOrderLine.SetCurrentKey(Type,"No.");
SalesOrderLine.SetRange(Type,SalesOrderLine.Type::Item);
SalesOrderLine.SetRange("No.","No.");
if SalesOrderLine.Find('-') then
    Error(Text001,TableCaption,"No.",SalesOrderLine."Document Type");
```

If you want to find the first record in a table or set, then use the FindFirst Method (Record). If you want to find the last record in a table or set, then use the FindLast Method (Record).

Next method

The Next Method (Record) is often used with FIND to step through the records of a table.

Next has the following syntax.

```
Steps := Record.Next([Steps])
```

In the following example, Find is used to go to the first record of the table. Next is used to step through every record, until there are no more. When there are no more records, Next returns 0 (zero).

```
if (Rec.FindSet) then
repeat
  // process record
until (Rec.Next = 0);
```

See Also

AL Methods
SystemId Field

Temporary Tables

2/17/2021 • 4 minutes to read • Edit Online

A temporary table is a temporary variable that holds a table. A temporary table is used as a buffer or intermediate storage for table data.

You can use a temporary table just like you use a database table. The differences between a temporary table and a database table are as follows:

- A temporary table data is not stored in the database. It's only held in memory until the table is closed.
- The write transaction principle that applies to a database table does not apply to a temporary table.

Advantage of using a temporary table

The advantage of using a temporary table is that all the interaction with a temporary table occurs on Dynamics 365 Business Central. A temporary table reduces the load on both the network and the SQL database server.

When you want to do many operations on the data in a specific table in the database, you can load the data into a temporary table when you modify it. Loading the data into a temporary table speeds up the process because all the operations are done in memory on the Business Central Server.

Creating and using a temporary table

There are three ways to implement a temporary table:

- Setting the TableType property on the table object to **Temporary**.
- Using a temporary record variable.
- Setting the SourceTableTemporary property on a page.

Whichever way you choose, you must create the table object that defines the fields, like any other table object. The differences are explained in the following sections.

> **TIP**
> Temporary tables retain system fields, like SystemID and data audit fields. For more information, see System Fields.

Setting the TableType to Temporary

APPLIES TO: Business Central 2020 release wave 2 and later

With this implementation, a physical table is not created in the database. In the table object, set the `TableType` property to `Temporary`:

```
table 50100 MyTable
{
    DataClassification = ToBeClassified;
    TableType = Temporary;

    fields

    ...
}
```

This implementation has the same effect as using a temporary record variable or setting the SourceTableTemporary property on a page. But the advantage is that the table schema isn't synchronized with the database. So it does not have restrictions on breaking schema changes, like removing a field, changing its data type or length.

It will also improve the performance of BACPAC generation using the sqlpackage command-line tool, compared to temporary tables based on temporary record variables and pages. For more information, see Performance of BACPAC generation.

Changing the table type

You can change from **Normal** to **Temporary**, and the other way around. When changing **Normal** to **Temporary** the table, you will have to synchronize the extension with the database. This step will remove the table from the database. So if the table contains data, you will have to synchronize the schema using the ForceSync mode.

Using a temporary record variable

With this implementation, a physical table is not created in the database. You create a global or local variable of the type record and set the Temporary Property next to it. The variable that holds a temporary table is defined just like any other global or local variable. The syntax is shown in the following example:

```
var
    TempInvoicePostBuffer: Record "Invoice Post. Buffer" temporary;
```

You manipulate the temporary table variable as you would with any other database table. For example, you can apply filters and do searches. For more information about the operations you can do, see Record Data Type.

Using a SourceTableTemporary property on page objects

Another option for temporary tables is to set the SourceTableTemporary on all pages that use the table. This implementation will also use a physical table in the database.

```
page 50100 MyPage
{
    PageType = Card;
    ApplicationArea = All;
    UsageCategory = Administration;
    SourceTable = TableName;
    SourceTableTemporary = true;

    layout

    ...
}
```

Temporary tables on reports and XMLPorts

You can also use temporary tables for storing the data set that is returned by a report object or that is imported by an XMLPort. To do so, set the **UseTemporary property** to **true** on the relevant report data item or XMLPort table element. For more information, see UseTemporary Property (XMLports) and UseTemporary Property (Reports).

It's useful to use a temporary table for a report when the column data is not mapped directly to a field in the database but is instead the result of a process or operation, such as an aggregation of data from several columns. If a temporary table is not used, the data must be handled by AL code.

For XMLPorts, if the data that you are importing has a different structure than the table in Dynamics 365 Business Central that you want to insert it into, you can import the data into a temporary table. You can then modify the data before inserting it into the database.

See Also

Getting Started with AL
Table Object
Temporary Property
UseTemporary Property (Report)
UseTemporary Property (XMLPort)

Retaining table data after publishing

2/17/2021 • 3 minutes to read • Edit Online

When developing an extension, you debug several times using the F5 shortcut key, and you also test your app by adding some sample data every time. To simplify the extension development process in Business Central, you can synchronize the sample data specified in the extension when you do subsequent publishing from Visual Studio Code.

How data synchronization works

The data synchronization between each publish is controlled by the `schemaUpdateMode` setting, which is specified in the `launch.json` file. This setting consists of three options; **Synchronize**, **Recreate**, and **ForceSync**.

The default value for `schemaUpdateMode` is set to the **Synchronize** mode, which means that every time you publish an extension to the development server, the data you entered previously stays. If you do not want to synchronize the sample data with each publish, you can change the `schemaUpdateMode` setting from `Synchronize` to, for example, `Recreate` with the syntax shown in the example below.

```
{
    "type": "al",
    "request": "launch",
    "name": "your own server",
    "server": "https://localhost",
    "serverInstance": "Nav",
    "authentcation": "UserPassword",
    "startupobjectId": 22,
    "schemaUpdateMode": "Recreate"
}
```

Recreate mode

When you set the schema update mode to **Recreate**, all the tables and table extensions are recreated at every publish, which means that all the data in those tables are lost. This means that you will get empty records when you publish your extension.

ForceSync mode

ForceSync is similar to the existing Synchronize schema update mode, but contains more freedom to make schema changes while retaining data. To enable this mode, set `schemaUpdateMode` to `"ForceSync"` and then set the `"version"` parameter in the app.json file to a fixed number. Data will be preserved in almost all cases with the exception of changing the main table's primary key, in which case the data from the extension tables will be lost. Field renames are allowed and supported in this mode, but the data can only be preserved if you maintain the same ID for the field. If you change both the name and the ID of the field then the data will be lost.

> **IMPORTANT**
> This schema update mode is only meant for testing and development and should never be used in production.

In addition to the `launch.json` file setting, the **ForceSync** switch is available through the PowerShell cmdlet `Sync-NavApp –Mode ForceSync`.

Things to be aware of

Synchronize is the default schema update mode for syncing the database and the extension. There are some key factors to consider when you work with the **Synchronize** mode.

- After publishing, the field data and the primary key information synchronizes with all the tables and the table extensions. This means that you can do additions easily, but not deletions. Breaking changes are never supported in synchronize mode. For example, you can add a field and sync that with the extension just by pressing the F5 shortcut key, but if a field is removed then the table data cannot be synchronized. If you, during development, for example, discover that you no longer want field **X**, and you then mark field **X** as obsolete, you may still want to write an upgrade codeunit to move the data from the obsolete field to a new field **Y** that you introduce. Later, the obsoleted field will not be available. But if you do not want the data, you can choose to use the **Recreate** mode instead.

- When you make changes to the data types, you can only *enlarge* the unit size, and *not decrease* the unit size. For example, you can set a text type from `Code[20]` to `Code[50]` or `Text[32]` to `Text[87]`, and you cannot set a text type from `Code[50]` to `Code[30]` or `Text[87]` to `Text[40]`.

- Making major table structural changes could lead to compilation errors. For example, if you want to update a primary key. In this case, the table data cannot be synchronized, and if you want to publish the extension, you must change the `schemaUpdateMode` to `Recreate`.

- For extensions built on Business Central Spring 2019 or earlier, if a table field has the `SqlDataType` set to a value other than `Varchar` (which is the default), you must delete the `SqlDataType` property on the field, otherwise, you will will not be able to successfully synchronize the extension.

  If the `SqlDataType` property is still needed, you will have to create a new table in the extension that has the same definition as the original table, and write upgrade code that migrates the data from the original table to the new table. For more information, see Writing upgrade code.

  Alternatively, if this is a development scenario, you can synchronize the extension using the ForceSync or Recreate mode.

See Also

AL Development Environment
Upgrading Extensions
Debugging

Classifying Data in Dynamics 365

2/17/2021 • 5 minutes to read • Edit Online

Dynamics 365 includes development features for tagging business data with specific classifications. Specifically, this includes data that is stored in table fields of the database and telemetry data that is emitted from the application.

About Data Classification

Classifying data serves different purposes. It can make data easier and more efficient to locate and retrieve, and also help to add another layer of protection and security for handling private and sensitive data. It can supplement your process for making the application compliant with legislative and regulatory requirements for collecting, storing, and using personal information.

> **IMPORTANT**
>
> You should consider the data classification features offered in Dynamics 365 as the first layer of classification - done by developers (Dynamics 365 and partners) on customizations, add-ons, and extensions. The second layer is to classify the sensitivity of the data itself. For more information, see Classifying Data Sensitivity. It is also important to consider end-users, and how they handle data they provide and that is made available to them.

What are the different data classifications?

The following table describes the different classifications that you can apply to data:

DATA CLASSIFICATION	DESCRIPTION	EXAMPLE
CustomerContent	Content directly provided/created by admins and users.	• Customer generated BLOB or structured storage data • Customer-owned/provided secrets (passwords, certificates, encryption keys, storage keys)
EndUserIdentifiableInformation	(EUII) Data that identifies or could be used to identify the user of a Microsoft service. EUII does not contain Customer content.	• User name or display name (DOMAIN\UserName) • User principle name (name@company.com) • User-specific IP address
AccountData	Customer billing information and payment instrument information, including administrator contact information, such as tenant administrator's name, address, or phone number.	• Tenant administrator contact information (for example, tenant administrator's name, address, e-mail address, phone number) • Customer's provisioning information

DATA CLASSIFICATION	DESCRIPTION	EXAMPLE
EndUserPseudonymousIdentifiers	(EUPI) An identifier created by Microsoft tied to the user of a Microsoft service. When EUPI is combined with other information, such as a mapping table, it identifies the end user. EUPI does not contain information uploaded or created by the customer (Customer content or EUII)	• User GUIDs, PUIDs, or SIDs • Session IDs
OrganizationIdentifiableInformation	(OII) Data that can be used to identify a tenant, generally config or usage data. This data is not linkable to a user and does not contain Customer content.	• Tenant ID (non-GUID) • Domain name in e-mail address (xxx@contoso.com) or other tenant-specific domain information
SystemMetadata	Data generated while running the service or program that is not linkable to a user or tenant.	• Database table names, database column names, entity names

Classifying data in tables and fields

Table objects and field controls include the `DataClassification` property that you can use to tag data with one of the classifications previously described.

Dynamics 365 operates with some standard rules for classification:

- When you add a new field to a table, the field is assigned an initial value of **ToBeClassified**.
- FlowField and FlowFilter fields are automatically set to the **SystemMetadata** data classification. This cannot be changed.
- Existing tables and fields (except for FlowFields and FlowFilters) in an application that has been upgraded from a Dynamics 365 version without the `DataClassification` property, will automatically be assigned the **CustomerContent** classification.

> **IMPORTANT**
> Microsoft is providing this `DataClassification` property as a matter of convenience only. It is your responsibility to classify the data appropriately and comply with any laws and regulations that are applicable to you. Microsoft disclaims all responsibility towards any claims related to your classification of the data.

For more information about this property, see DataClassification Property.

Data classification on upgrade

When you upgrade an application from a Dynamics 365 version that does not contain the `DataClassification` property, existing tables and fields (except for FlowFields and FlowFilters) will automatically be assigned the **CustomerContent** classification. You can then access the `DataClassification` property on these tables and fields, and change the classification as needed. FlowFields and FlowFilters will be assigned the **SystemMetadata** classification automatically.

IMPORTANT

After upgrade or import of objects, using fob files, that introduce new tables and/or fields, make sure to synchronize new tables and/or fields to enable Data Sensitivity Classification by running **SyncAllFields** method in `Data Classification Mgt.` Codeunit (Codeunit 1750). No action is needed when extensions are installed, as installation of extension automatically triggers **SyncAllFields** method. See example below.

Run the script below from Developer Shell:

```
Invoke-NAVCodeunit -Tenant <TenantID> -CompanyName <CompanyName> -CodeunitID 1750 -MethodNAme
'SyncAllFields' -ServerIntance <ServerInstance>
```

Bulk-classifying data

The Field Data Classification report, which is described in the *Viewing current field classifications* section in this topic, provides an overview of the data classifications for fields. The report also lets you assign data classifications for more than one field. For example, this is useful if you are assigning classifications for the first time, or have changed several fields and want to update their classifications. You can bulk-edit classifications only for fields in AL Language development environment. The script does not update fields in extensions.

To bulk-edit classifications, export the report to Excel, update the classifications, and then save your changes. Then, in Windows PowerShell, run the following commands to run the `Import-Module` script and set the classifications on the fields.

To run the script from the default folder on the DVD, run:

```
Import-Module WindowsPowerShellScripts\DataClassification\DataClassification.psm1
```

To update the `DataClassification` property, run the following command. Replace <FilePath> with the full path to the client files. For example, C:\Program Files\Microsoft Dynamics 365 Business Central\160\RoleTailored Client.

```
Set-FieldDataClassificationFromExcelFile -ExcelFilePath "C:\BC\W1 Fields (Main).xlsx" -SheetName 'Field Data
Classification' -RTCFolder "<FilePath>" -DBName BC2 -OutputFolder C:\BC2\Classifications
```

Viewing current field classifications

To view the data classification on all fields, you can do one of the following:

- From the client, search for and open the **Data Classification Worksheet** page.
- Create a page that has the virtual table **Field** (ID 2000000041) as its source, and open the page in the client.

Classifying data in custom telemetry trace events

Custom telemetry trace events are defined by calls to the **SENDTRACETAG** method in the application code. The **SENDTRACETAG** method includes an optional parameter called `DataClassification` that you can use to tag the telemetry trace event with a data classification.

For more information, see SendTraceTag and Instrumenting an Application for Telemetry.

See Also

Data Classification

Integrating Microsoft Dataverse for Extension Development

2/17/2021 • 2 minutes to read • Edit Online

NOTE

Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

Develop extensions and streamline the workflow by synchronizing Microsoft Dataverse data with Dynamics 365 Business Central.

For developing extensions to integrate with sales data, you simply enable the tables used in Microsoft Dataverse. The extension development process includes the following set of properties in Dynamics 365 Business Central to enable field mapping. You can enable the field mapping by using the following properties. The tables are extensible, so that you can update Microsoft Dataverse with data as well.

Associated table and field properties

The following properties are used for integrating with Microsoft Dataverse:

PROPERTIES	APPLIES TO	DESCRIPTION
TableType Property	Tables	Specifies the table type. This enables the table to integrate with the external database. For example, `CDS`.
ExternalName Property	Tables, Fields	Specifies the name of the original table in the external database when used as a table property. Specifies the field name of the corresponding field specified in the external table when used as a field property.
ExternalAccess Property	Fields	Specifies the access to the underlying Microsoft Dataverse table when Microsoft Dataverse tables are generated using the **AL Table Proxy Generator** tool, see AL Proxy Table Generator
ExternalType Property	Fields	Specifies the data type of the corresponding column in the Microsoft Dataverse table.

PROPERTIES	APPLIES TO	DESCRIPTION
OptionMembers Property	Fields	Sets the option values for a field, text box, or variable.
OptionOrdinalValues Property	Fields	Specifies the list of option values. You can set this property, if the **ExternalType** is set to **Picklist**.

Enabling the table

Typically in Microsoft Dataverse, tables handle the internal processes. In order to access to the underlying Microsoft Dataverse table, you use the `TableType` property and select the value called **CDS**. This enables the table as an integration table for integrating Dynamics 365 Business Central with Microsoft Dataverse. The table is mainly based on a table in Microsoft Dataverse, such as the Accounts table.

Snippet support

Typing the shortcut `ttable` will create the basic layout for a table object when using the AL Language extension in Visual Studio Code.

Example

In the following example, the `SalesIntegration` table uses the `TableType` and `ExternalName` properties to link the underlying **Microsoft Dataverse** table for mapping the columns from the `Sales` table with the specified fields.

```
table 50100 SalesIntegration
{
    TableType = CDS;
    ExternalName = 'Sales';

    fields
    {
        field(1; ActualSales; Integer)
        {
            ExternalName = 'ActualSale';
            ExternalAccess = Full;
            ExternalType = 'String';
        }

        field(2; SalesCategories; Option)
        {
            ExternalName='SalesCategory';
            ExternalAccess = Read;
            ExternalType = 'Picklist';
            OptionMembers = Manufacturing, Marketing, Support;
            OptionOrdinalValues = -1, 1, 2;
        }
    }
}
```

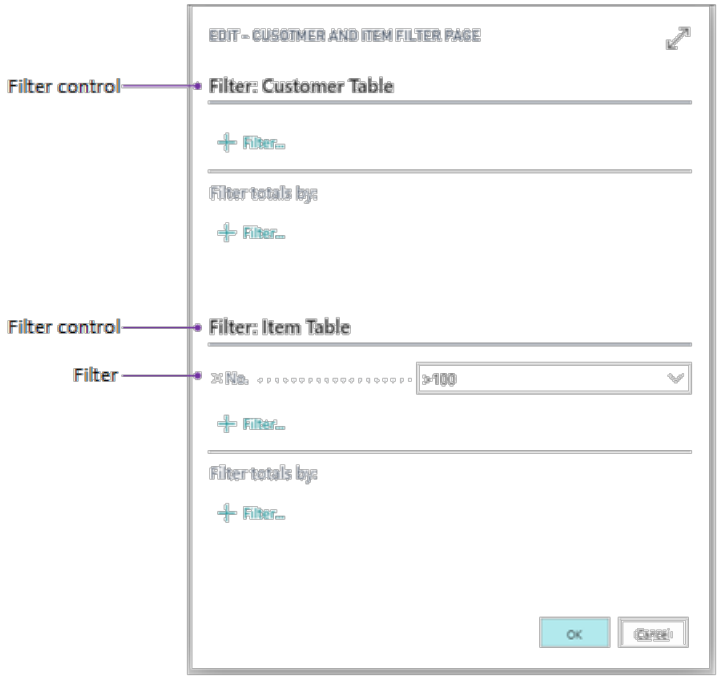
See Also

Table Properties
TableType Property
AL Proxy Table Generator

Creating Filter Pages for Filtering Tables

2/17/2021 • 2 minutes to read • Edit Online

In AL code, you can use the FilterPageBuilder data type to create a filter page that enables users to set filters on multiple tables. Filter pages contain one or more filter controls, where each filter control can be used to set filters on a specific table. In the Business Central client, filter pages are generated at runtime and run in a modal dialog box. The following figure illustrates a filter page that filters on the **Item** table.



To create a filter page, you use AL code and the methods that are available for the FilterPageBuilder Data Type. The following code example shows the code that creates the filter page in the figure.

```
var
    varItem: Record Item;
    varFilterPageBuilder: FilterPageBuilder;
begin
    varFilterPageBuilder.AddTable('Customer Table', Database::Customer);
    varFilterPageBuilder.AddRecord('Item Table', varItem);
    varFilterPageBuilder.Addfield('Item Table', varItem."No.", '>100');
    varFilterPageBuilder.PageCaption := 'Customer and Item Filter Page';
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type

Formatting Decimal Values in Fields

2/17/2021 • 5 minutes to read • Edit Online

This article describes how you can format the decimal values that appear in fields on table, pages and reports. For example, you can change how the data appears in a Cue on the Role Center page. To format data, you use a combination of the AutoFormatType Property, AutoFormatExpression Property, and DecimalPlaces Property of the field. These properties work together to enable you to specify the following:

- Display amounts and unit amounts in another currency.
- Specify the number of decimal places.
- Specify whether to display a thousand separator.
- Specify characters before and after the value, such as currency signs or %.

Implementation overview

When a field is used on a page or report, you can set the **AutoFormatType** and **AutoFormatExpr** properties directly on the page field or report field (column), or you can set them on the underlying table field. If you specify the properties on the table field, then the format applies wherever the field is used. Specifying the properties on the page or report field will only apply the format to the specific page or report. If you specify the properties on the table field and the page or report field, then the settings on the page or report field take precedence.

When you use the **AutoFormatType** and **AutoFormatExpression** properties to format a field, two events are raised by the system codeunit **45 Auto Format**: OnResolveAutoFormat and OnAfterResolveAutoFormat.

Setting up data formatting

The settings for the **AutoFormatType**, **AutoFormatExpression**, and **DecimalPlaces** properties will depend on the type of data that is displayed, for example, this could be currency amounts, unit amounts, simple decimals, or ratios. For the most part, the **AutoFormatType** property is the primary setting, which in turn determines the options for setting the **DecimalPlaces** and **AutoFormatExpr** properties.

If the **AutoFormatType** is not set or is set to an incorrect property value, then the default setting is used, regardless of whether the **AutoFormatExpression** or **DecimalValues** properties are set. The default setting uses `AutoFormatType = 1` and `AutoFormatExpression = ''`.

The following tables describes how to set each of the properties to achieve the format that you want.

Setting the DecimalPlaces property

With the following set up, the **AutoFormatExpression** property is ignored.

AUTOFORMATTYPE PROPERTY	DECIMALPLACES PROPERTY	USAGE DESCRIPTION

AUTOFORMATTYPE PROPERTY	DECIMALPLACES PROPERTY	USAGE DESCRIPTION

0	Set to the number of decimal places that you want to display for the value.	Use this configuration when you want to format the decimal value according the Standard Format 0 (which is the default format) with a specific number of decimal places.
		
		For example, if the value is a US decimal `-76543.21` and you set the **DecimalPlaces** property to `0`, then the value appears as **76,543**. The properties will look like this:
		
		`AutoFormatType = 0;`
		`DecimalPlaces = '0';`

Setting the AutoFormatExpression property

With the following setup, the **DecimalPlaces** property is ignored.

AUTOFORMATTYPE PROPERTY	AUTOFORMATEXPRESSION PROPERTY	USAGE DESCRIPTION

1	Set to return a currency code, such as USD or IDR. The blank currency code `''` denotes LCY and is the default value.	Use this configuration when you want to format the data as an amount. For example, a sales order will use two decimals when the currency is defined as US dollar and no decimals when the currency is defined as IDR (Indonesian rupiah). For example:
		
		`AutoFormatType = 1;`
		`AutoFormatExpression = 'IDR';`

2	Set to return a currency code such as USD or IDR. The blank currency code `''` denotes LCY and is the default value.	This is similar to the previous configuration where the **AutoFormatType** property is set to `1`, except you use this configuration when you want to format the data as a unit amount.

AUTOFORMATTYPE PROPERTY	AUTOFORMATEXPRESSION PROPERTY	USAGE DESCRIPTION

10	Set to the property according to the following syntax:
	`'[SubType][,<currencycode or expression>[,<PrefixedText>]]'`
	`SubType` can be `1`, `2`, another number, or omitted:
	`1` sets the value to an amount type (see 1 above). `2` sets the value to a unit amount type (see 2 above). The syntax for these two settings is:
	`'SubType,<currencycode[, <PrefixedText>]'`
	If you omit the subtype or use a number other than 1 or 2, the syntax is:
	`'<CustomNumber>, <expression>[, <PrefixedText>]'`
	where `<expression>` sets the precision and one of the standard formats. For more information, see Standard Formats.
		Use SubType `1` to add the currency symbol and use the amount type precision. You use SubType `2` for unit amount precision. For example, set the property to `'1,USD'` to add the **$** symbol, like **$543.21**.
		`AutoFormatType = 10;`
		`AutoFormatExpression = '1,USD';`
		If you omit the SubType, you can use this configuration to customize the format based on one of the standard formats. This option enables you to specify characters before and after the decimal value, such as currency signs **$** and percent **%**.
		For example, if you want to prefix the decimal value with a **$**, include a thousand separator, and have a maximum of two decimal places, such as **$76,453.21**, then you can set the properties to:
		`AutoFormatType = 10;`
		`AutoFormatExpression = '$<precision, 2:2><standard format, 0>'`
		If you want to display the decimal value as a percentage, then you can add `%` at the end of the setting. For example:
		`AutoFormatType = 10;`
		`AutoFormatExpression = '<precision, 1:1><standard format,0>%'`
		When you include a `%` at the end of the setting, then the decimal value is assumed to be the ratio, and the decimal value will be multiplied by 100. For example, a value of 0.98 will be formatted to **98%**.

11	Set the property to the standard format as explained below. For example: `'<Precision,3:3><Standard Format,0>'`	Use this option when you want full control over the formatting. The format string will be applied exactly as specified in the AutoFormatExpr property.

Precision

The precision determines the minimum and maximum number of decimal points for values. The precision takes the format `<precision,minimum:maximum>`. For example, `<precision,minimum:maximum>` sets the data with a minimum of 2 and a maximum of 3 decimal places.

Standard formats

The following table describes the standard formats that are available for the **AutoFormatExpr** property when the **AutoFormatType** property is set to 10.

STANDARD FORMAT	FORMAT DESCRIPTION	EUROPE DECIMAL EXAMPLE	US DECIMAL EXAMPLE
0	<Sign> <Integer Thousand> <Point or Comma> <Decimals>	-76.543,21	-76,543.21
1	<Sign> <Integer> <Point or Comma> <Decimals>	-76543,21	-76543.21
2	<Sign> <Integer> <Point or Comma> <Decimals>	-76543.21	-76543.21
3	<Integer Thousand> <Point or Comma> <Decimals> <Sign>	76.543,21-	76,543.21-
4	<Integer> <Decimals> <Point or Comma> <Sign>	76543,21-	76543.21-
9	XML format	-76543.21	-76543.21

See Also

AutoFormatType Property
AutoFormatExpression Property
DecimalPlaces Property

Working With Media on Records

2/17/2021 • 7 minutes to read • Edit Online

This topic describes how you can upload media, such as an image, to the database for displaying with records in the client. There are two ways that you can do this:

- Use a BLOB data type

You add media to a BLOB data type field on the record. For more information, see BLOB Data Type.

- Use a Media or MediaSet data type

This way enables you to store media in system tables of the database, and then reference the media from application records. For example, you can:

- Display media with records in list type pages, when the page is viewed in the **Tile** layout. For more information, see Displaying Data as Tiles.
- Display media on a card type page for a record.
- Display media in a report.

Using the Media or MediaSet data type provides better performance than using a BLOB data type and is more flexible in its design. With a BLOB data type, each time the media is rendered in the client, it is retrieved from the SQL database server, which requires extra bandwidth and affects performance. With the Media and MediaSet data types, the client uses media ID to cache the media data, which in turn improves the response time for rendering the media in the user interface.

Using Media and Media Sets on records

Table fields support two data types for adding media to records: **Media** and **MediaSet**. With these data types, you can import media directly from a file to a record, or media can be passed to the record in an InStream object. Imported media is stored as an object in the system table **2000000184 Tenant Media** of the tenant database. Each media object is assigned a unique identifier (ID).

Media data type

The **Media** data type associates a record with a single media object. For example, you can use this data type to display an image with each record in a list type page.

If a media object is added to Media data type field, the field references the media object by its ID.

MediaSet data type

The **MediaSet** data type associates a record with one or more media objects. This enables you to set up a collection or catalog of media for a record. For example, you can use this data type to set up a slide show of images for a record in a card type page.

If a media object is added to **MediaSet** data type field, the media object is assigned to a media set in the system table **2000000183 Tenant Media Set**. The media set is assigned a unique identifier, which is then referenced from the field. The media set is created with the first file media object that you add on the record. Any additional media objects for the record are then associated with the same media set.

Indexing of media objects in a media set

A media set is an ordered list of media objects, determined by the order in which the media objects were added to the media set. This order cannot be changed. To identify this order, each media object is assigned an index number, starting a 1. This means that the first media added gets the index 1, the second media gets the index 2, and so on. If a media object is removed from the set, the list is re-indexed accordingly.

> **NOTE**
> If a **MediaSet** data type field is used in a report object, then only the first associated media file is displayed in the generated report.

Supported Media (MIME) types

The media type, also referred to as the MIME (Multipurpose Internet Mail Extensions) type, is an Internet standard to describe the contents of a file. Internet browsers use the MIME type to determine how to handle the file. The Media and MediaSet datatypes support all recognized MIME types.

A MIME type is defined by two parts, the *type* and *subtype*, where the format is `type/subtype`. For example, the MIME type for a JPEG image is image/jpeg. There are several types, including image, application, audio, video, text, and more. Each MIME type is associated with one or more acceptable file extensions. The following table lists some of the more common MIME types and their file extensions.

MIME TYPE	FILE EXTENSION
image/bmp	bmp
image/jpeg	jpeg, jpg, jpe
image/gif	gif
application/msword	doc
application/octet-stream	json
application/vnd.openxmlformats-officedocument.wordprocessingml.document	docx
application/vnd.ms-excel	xls
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	xlsx
application/vnd.ms-powerpoint	ppt
application/vnd.openxmlformats-officedocument.presentationml.presentation	pptx
application/pdf	pdf
application/xml	xml
audio/mpeg	mp3
audio/x-wav	wav

MIME TYPE	FILE EXTENSION
video/mp4	mp4
video/x-msvideo	avi
text/html	htm, html
text/plain	txt

> **NOTE**
>
> GIF type is not supported on reports. If you want to display an image on a report, use another supported type.
>
> Files with extensions that are not recognized are also supported and can be imported. These are stored as BLOBs (binary larger objects).

General procedure for adding media to records

The general procedure for setting up media on records is as follows:

1. Obtain the media file or files that you want to use on the record.
2. In AL, modify the table object to include a field that has the data type **Media** or **MediaSet**.
3. Add AL code that imports the media on the field.

   The **Media** and **MediaSet** data types support several methods that you can use to manage the media on records. See the next section for a complete list of methods with a link to more details, such as usage, parameters and sample code.

   For example, you can create a codeunit that calls one of the import methods, or add a page action that calls one of the methods.

AL methods

To get an overview of the methods that are related to the Media and MediaSet data types, see Media Data Type and MediaSet Data Type.

Automatic deletion of unused Media objects

When a table record that contains a media object is deleted, the OnDelete trigger gets the media or media set's ID, and uses the ID to look for other references to the media object from the same field index in the same table. If no other references are found, the media object is assumed to be unreferenced and it is deleted. The runtime will not look in all tables in the database to see if a media object is referenced elsewhere, because doing this would decrease performance and result in costly SQL table scans. If media objects are to be shared between tables, they should be shared through a reference table or by sharing the media set field content as described in the next section.

Sharing Media objects between different tables

To maintain data integrity related to media object, it's important to notice that the Media and MediaSet data types are complex data types that are referenced by an ID. The ID is stored in the record field that contains the media object. If a simple copy operation is performed to copy the media object from one media set field to another, the ID is copied to the new field. However, the application does not know that the media object is

referenced in two different fields, which causes issues when a row that contains the media ID is deleted.

To avoid unintentionally deleting referenced media objects, media sharing should be done by using the INSERT method to insert the media (by its ID) into the new media set field. This will create the correct (new) MediaSet records in the system tables, which means that the media object in one field will not be deleted if media object in the other field is deleted.

Example

This example copies a media set field called `MediaSetField` in table `mediaSourceTable` to a field in another table `mediaTargetTable`. The `FOR` loop will iterate all media objects in the source, and then insert their ID in the target field.

```
for index := 1 to mediaSourceTable.MediaSetField.COUNT do
 mediaTargetTable.MediaSetField.INSERT(mediaSourceTable.MediaSetField.ITEM(index));
MediaTargetTable.Modify(true);
```

This will create a new media set that contains the shared media object references. When you delete the media set (by deleting the MediaTargetTable record), the runtime will detect that the media object is used in multiple media sets, and therefore will not delete the media objects. The media objects might eventually be deleted when the runtime cannot find other references.

> **IMPORTANT**
> The simple field copy statement `mediaTargetTable.MediaSetField := mediaSourceTable.MediaSetField;` can only be used if `mediaTargetTable` is declared as the same record subtype as `mediaSourceTable`, and the target and source field IDs are the same.

See Also

BLOB Data Type
Media Data Type
MediaSet Data Type

Using Partial Records

2/17/2021 • 5 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

The partial records capability in Business Central allows for loading a subset of normal table fields when accessing a SQL based data source. Using partial records improves performance of objects like reports and OData pages - objects whose source code loops through records. It's particularly beneficial when table extensions are used in the application.

Accessing a data source from AL code is typically done by using the record's methods GET, FIND, NEXT, and so on. Without using partial records, the runtime loads all normal fields when accessing the data source. Using the partial records API, you can now select a set of fields and only load them.

API Overview

To accommodate partial record loading, the following methods are available on both the RecordRef and Record data type in AL. These methods operate on the record instance that they're called on, changing the set of fields to load until otherwise altered. The methods are divided into two groups: methods that pertain to subsequent loads and methods that pertain to the currently loaded record.

Subsequent load methods

METHOD	DESCRIPTION	SEE MORE
SetLoadFields	Specifies a set of fields to be initially loaded when the record is retrieved from its data source. A call to this method will overwrite any fields that were previously set to load.	Record.SetLoadFields RecordRef.SetLoadFields
AddLoadFields	Adds fields to the current set of fields to be initially loaded when the record is retrieved from its data source. Subsequent calls to this method won't overwrite fields that were previously selected for loading.	Record.AddLoadFields RecordRef.AddLoadFields

Current load methods

METHOD	DESCRIPTION	SEE MORE
AreFieldsLoaded	Checks whether the specified fields are all initially loaded.	Record.AreFieldsLoaded RecordRef.AreFieldsLoaded
LoadFields	Accesses the table's corresponding data source to load the specified fields.	Record.LoadFields RecordRef.LoadFields

A record instance that has been previously loaded with fields can be reset to a non-partial load either by calling Record.SetLoadFields without any parameters, or by calling Reset. After being reset, subsequent loads will behave as if SetLoadFields hadn't previously been called on the record instance.

Example

The following code shows a way to load only a single field from the **Item** table for computing the arithmetic mean.

```
procedure ComputeArithmeticMean(): Decimal;
var
    Item: Record Item;
    SumTotal: Decimal;
    Counter: Integer;
begin
    Item.SetLoadFields(Item."Standard Cost");
    if Item.FindSet() then begin
        repeat
            SumTotal += Item."Standard Cost";
            Counter += 1;
        until Item.Next() = 0;
        exit(SumTotal / Counter);
    end;
end
```

Notice that the call to SetLoadFields occurs before the data fetching operations. This call determines which fields are needed for the FindSet call. You use the same pattern for AddLoadFields calls.

Usage guidelines

This feature gives you the ability to limit the fields that load for a record to only those fields that are necessary. In general, loading fewer fields will make operations faster. But the most significant performance gains can be seen with table extensions - by not loading unnecessary fields in table extensions. Table extensions that don't have any fields for loading won't be part of the data join, which saves time.

> **TIP**
> Testing on the previous example code showed that the execution time for loading only the "Standard Cost" field was nine times faster than loading all normal fields. Your performance numbers will vary depending on the machine and the setup with the SQL database.

For performance reasons, it's not recommended to use partial records on a record that will do inserts, deletes, renames, transferfields, or copies to temporary records. All these operations require that all fields on the record are loaded, so the platform will emit a JIT load if they're not already loaded. A JIT load requires to access the data source again, this cost is larger than the gains of loading fewer fields. For this reason, the feature is especially advantageous in reading-based scenarios.

Reports and OData pages

Currently, the platform implicitly uses partial records when fetching data for reports and calling pages through OData.

For reports, the fields that are selected for loading are fields setup as columns in the report data set. If a report accesses a field that isn't in the data set, it's beneficial to do one of the following to avoid just-in-time (JIT) loading:

- Add the field as a column in the data set.
- Add the field on the OnPreDataItem trigger.

  The following example code snippet illustrates how to use the AddLoadFields method on a report's

OnPreDataItem trigger to add a field for loading:

```
trigger OnPreDataItem()
begin
    CurrencyDataItem.AddLoadFields(CurrencyDataItem."ISO Numeric Code");
end;

trigger OnAfterGetRecord()
begin
    if (CurrencyDataItem."ISO Numeric Code" <> 'DKK') then begin
        CurrReport.Skip();
    end;
end;
```

The same issue arises for pages called through OData. If a field isn't requested for the OData output, it won't be loaded. In this case, it's beneficial to add the fields using the AddLoadFields method in the OnOpenPage trigger.

Just-in-time (JIT) loading

When a record is loaded as a partial record, the obvious question is: What happens when accessing a field that hasn't been selected for loading?". The answer is JIT loading. The platform, in such a case, does an implicit GET on the record and loads out the missing field.

When JIT loading occurs, another access to the data source is required. These operations tend to be fast because they're GET calls. GET calls can often be served by the server's data cache or can be resolved as a clustered index operation on the database.

A JIT load may fail for multiple reasons, like:

- The record has been modified between the original load and the subsequent JIT load.
- The record has been deleted between the original load and the subsequent JIT load.
- The record has been renamed between the original load and the subsequent JIT load.

Iterating over records

When iterating over records in the database, an enumerator is created based on selected fields. Then, a row is fetched when NEXT is called. This behavior is an optimization that allows for large parts of the operation to be done only once.

Certain operations will invalidate the enumerator and force the creation of a new one, which adds some overhead. As long as the enumerator isn't invalidated too frequently, this model is an effective optimization. When accessing fields that aren't loaded, the platform does a JIT load, followed by an update of the enumerator. This process eliminates the need to trigger a JIT load on subsequent iterations.

When passing a record by value, without using a VAR, a new copy of the record is created. The original record and its copy don't share filters, fields selected for load, and so on. So accessing an unloaded field will trigger a JIT load. But it won't update the enumerator, which means future iterations will also require JIT load.

There are a few options to remedy this situation:

- Pass the record reference using VAR parameter allows for updating of enumerator.
- Call AddLoadFields(fieldnames) on the original record before passing by value.
- Before calling FIND, GET, NEXT, and so on, use the SetLoadFields(fieldnames) to set all fields needed for load.

See Also

FAQ for Partial Records

Performance Articles For Developers
Get, Find, and Next Methods
Configuring Business Central Server

FAQ for Partial Records

2/17/2021 • 2 minutes to read • Edit Online

This article answers some of the most typical questions about the partial records capability in Business Central.

Do I need to change any code to keep my solution functional?

No. If a field that's not selected for loading is accessed, the data will be fetched automatically for the current record. The field will then be selected for loading on future requests by using this record instance. However, you may still get the message *Record has been modified by another user*, like you could before. But now, the message can also appear in read-only scenarios where the record isn't locked.

Where is "Partial Records" applied? To all records instances?

Currently, the partial records capability is only implicitly used on report data items and OData pages. The behavior for all other record instances is as before, that is, all fields will be loaded. However, as an AL developer, use in your own code to improve performance in looping code.

What happens when accessing fields not selected for load?

The platform will load any fields not selected for loading when accessed, making it seem like the field was already loaded.

What happens if the record has been modified or deleted in between the original load and JIT load?

The platform will validate that either the record is unchanged, or no change has been made to read fields.

What's a JIT load?

The platform determines that a field, which isn't currently loaded, is needed. Then it loads the field.

Can I optimize my code, so that I can avoid an extra database call to load my field?

Yes. You can add your field to the list of fields selected for loading by using the AddLoadFields method.

Which fields can be selected not to be loaded?

Any field that isn't a FlowField, FlowFilter, Primary Key, Timestamp, SystemId, Audit Fields, or Blobs.

How do partial records interact with table extensions?

When no fields from a table extension are selected for loading, then table extension data isn't joined, which saves time.

What's the performance overhead of determining the fields necessary for a record?

For reports, a calculation takes place at compile time and once per data item type per report execution. For OData pages, the calculation takes place at compile time and once per request segment.

Compared to the time it takes to fetch data, the overhead is insignificant.

Can I disable partial records in certain scenarios?

- With Business Central on-premises, you can turn off partial records by using **Enable Partial Records** setting of Business Central Server. See Configuring Business Central Server .
- With Business Central online, you can call the SetLoadFields method with no fields. Calling an empty SetLoadFields method will revert the behavior to what it was before Business Central 2020 release wave 2.

See Also

Performance Articles For Developers
Get, Find, and Next Methods
Configuring Business Central Server

Migrating Tables and Fields Between Extensions

2/17/2021 • 4 minutes to read • Edit Online

INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

Data migration allows you to move table and field data between extensions. The concepts and processes in this article apply to large-scale and small-scale data migrations. A large-scale migration is typically upgrade scenario, like upgrading from Business Central version 14 to version 16. Small-scale migrations are scenarios where you want to move a limited number of objects from one extension to another.

Overview

The move is divided into two phases: development and deployment. However, before you begin, you have to determine the direction of the migration within the dependency graph.

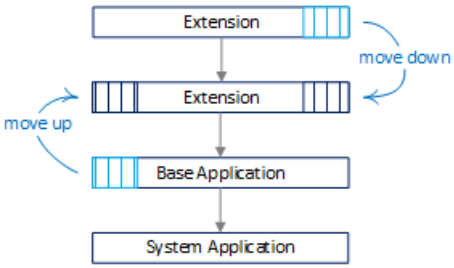
Limitations

- The concepts discussed in this article pertain only to AL. They're not supported in C/AL.
- You can only move fields from a table object to a table extension or move entire tables from one extension to another.
- You can't:
  - Move table extensions to other extensions.
  - Merge multiple table extensions into one extension.
  - Split table extensions into multiple extensions.

  Although you can't use this feature for these scenarios, you can achieve the scenarios by other means, which require more manual work. Examples include obsoleting, copying/re-IDing/renaming, moving data, and more.

Determining the migration direction in the dependency graph

The process to migrate tables and fields to another extension depends on the migration's direction in the dependency graph. The following figure illustrates a simplified extension dependency graph. From top to bottom, an extension is dependent on any extension below it in the graph.



When to move down

Typical move-down scenarios include:

- Moving to a shared extension.

  You want to move common tables to a separate extension that other extensions can have a dependency on.

- Transitioning from a customized base application extension with its own ID to an extension on top of the

Microsoft Base Application.

You have a customized base application extension with its own ID. You want to transition to the Microsoft Base Application. In this case, customizations remain in the current extension. Base objects are removed and ownership transferred to Microsoft Base Application.

This scenario is typical for embed ISV apps and on premises solutions moving to the cloud. It's also relevant for solutions that will remain on-premises for the foreseeable future. Use it to refactor code customizations into cleaner, standard base with extensions as part of upgrading.

For more information, see Moving Tables and Fields to Extension Down the Dependency Graph.

When to move up

Typical move-up scenarios include:

- Splitting an extension in two, with one dependent on the other.
- Extracting the system application from the base application.
- Transitioning from a customized base application extension with Microsoft ID to the Microsoft Base Application.

  You have a customized base application extension that reuses the Microsoft application ID. You want to transition to the standard Microsoft Base Application. In this case, customizations are moved out of the base application up into new extensions. The new extensions have new application IDs and dependencies to the standard Microsoft base application. The customization objects are removed from the custom base application and ownership transferred to the new extensions.

For more information, see Moving Tables and Fields to Extension Up the Dependency Graph.

Development

Development involves making application code changes required for the move. In short, the work involves:

- Creating the *releasing extension* version

  You create a new version of the original extension. This new version contains the tables or fields you want to keep in the extension. The tables and fields that you want to move are deleted from this extension. They're moved to the *receiving extension*.

- Creating the *receiving extension*

  You create new extension that includes the table and fields that you want moved. It essentially includes those tables and fields deleted from the releasing extension.

The key to the move is the *migration.json* file. You add the file to the project for the releasing extension. This file provides a pointer to the ID of new receiving extension where tables and fields are to be moved to. The *migration.json* is used in the deployment phase. Its purpose is to transfer ownership of tables and fields in the database from one extension to another. For more information, see Migration.json File.

Deployment

The deployment phase is when the data is migrated to new tables in the database. In this phase, ownership of tables and fields is switched from one extension to another. Deployment involves publishing, syncing, upgrading, and installing extensions.

The order that you synchronize extensions is important:

- The receiving extensions must be synchronized first.

- The releasing extensions, which include those extensions that include the migration.json file, must be synchronized last.

  These extensions are synchronized last because the tables are moved during the synchronization of the releasing extensions. At the end of the synchronization process, the system checks for breaking changes introduced by the extension. If the extension isn't synchronized last, breaking changes will be detected.

See Also

Publishing and Installing an Extension
JSON Files
Upgrading Extensions

Moving Tables and Fields to Extensions Down the Dependency Graph

2/17/2021 • 3 minutes to read • Edit Online

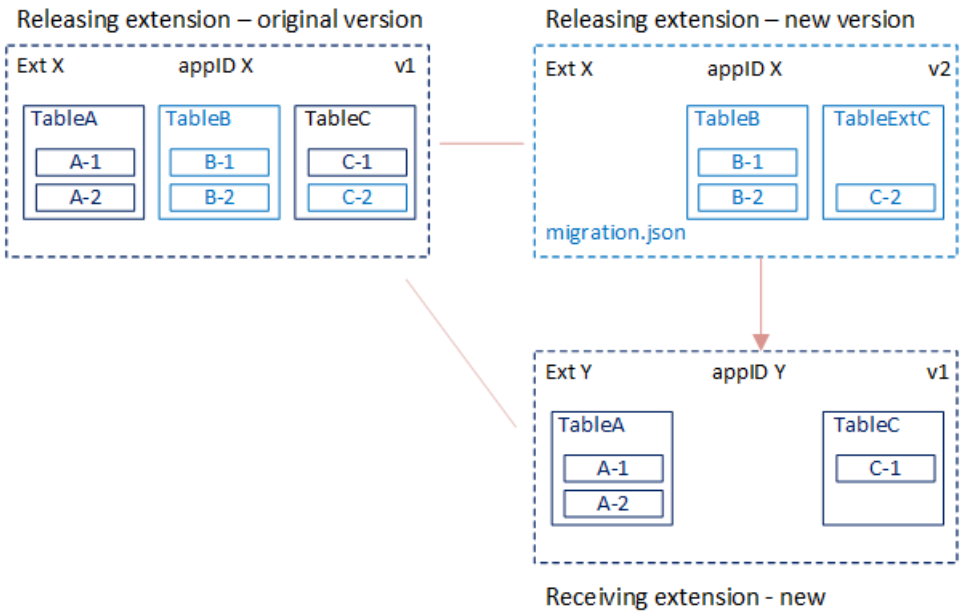
INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

This article explains how to move tables and fields from an extension to another extension that is down the dependency graph.

> **TIP**
> For more information about the general concepts for moving tables and fields between extensions, see Migrating Tables and Fields Between Extensions. This article explains the the difference between moving up and down the dependency graph.

Overview

The steps in this article are based on the example illustrated in the following figure. Although your scenario is different, the concept and process are much the same.



In the example, **TableB** and **Field C-2** are customizations. You'll keep these elements in the original extension, but create a new version without **TableA** and **TableC**. You'll move **TableA** and **TableC** down the dependency chain to a new, separate extension.

Prerequisite

If you're moving enum type fields, then your solution must be running on Business Central 2020 release wave 1, version 16.5 or later. For more information, see Known Issues.

Create receiving extension (Ext Y)

The receiving extension will contain the table and fields that you want to move. In this example, these objects

include **TableA** and **TableC**.

1. Create an AL project for the receiving extension.
2. Add a table object definition for **TableA**.

   The table definition (schema) must include the full schema of the releasing extension **Ext X**, with the same field definitions. You can add new fields.
3. Add a table object definition for **TableC**.

   The table definition (schema) must include the full schema of the releasing extension **Ext X**, with the same field definitions, except don't include field **C-2**. You can add new fields.
4. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `11111111-aaaa-2222-bbbb-333333333333`.
5. Compile the extension package.

Create new version of releasing extension (Ext X v2)

1. In the releasing extension AL project, add a migration.json file that points to the ID of the target extension.

   ```
   {
   "apprules": [
       {
           "id": "11111111-aaaa-2222-bbbb-333333333333"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.
2. Modify the app.json file as follows:

   - Ensure that `"target": "OnPrem"`.
   - Increase the `"version"` value.
   - In the `"dependencies"` parameter, set up a dependency on the new receiving extension **Ext Y**.

   For more information, see App.json file.
3. Complete the following steps for **TableC**.

   a. Add a table extension object **TableExtC**.
   b. In table extension object **TableExtC**, add a field definition for field **C-2** that matches its definition in the original **TableC** object.
   c. Delete the original **TableC** object.
4. Delete the entire table object for **TableA**.
5. Compile a new version of the extension package.

Deploy extensions

1. Uninstall the old version of the releasing extension **Ext X**.
2. Publish the new receiving extension **Ext Y** and releasing extension version **Ext X v2**.
3. Synchronize the receiving extension **Ext Y**.

This step creates empty database tables for **TableA** and **TableC** that are owned by the receiving extension **Ext Y**.

> **IMPORTANT**
>
> The receiving extension must always be synchronized first.

4. Synchronize the new version of the releasing extension **Ext X v2**.

   This step first reads rules in the migration.json file of the extension, then does the following operations in the database:

   - Creates a companion table for field **C-2** of the table extension object **TableExtC**.
   - Copies data from column **C-2** in the original **TableC** to new companion table **TableExtC**.
   - Temporarily renames the new empty tables **TableA** and **TableC** made by receiving extension **Ext Y**.
   - Renames the original tables **TableA** and **TableC** that include the data. Instead of including the ID of the releasing extension **Ext X**, the names are changed to include the ID of the receiving extension **Ext Y**. This step essentially transfers ownership from **Ext X** to **Ext Y**.
   - Deletes the unused column for **C-2** in the original table **TableC**.
   - Deletes the empty, renamed tables of **Ext Y**.
5. Install the receiving extension **Ext Y**.
6. Run Start-NAVAppDataUpgrade cmdlet on the new releasing extension version **Ext X v2**.

   This step basically installs the new extension version. You run a data upgrade because an earlier version has been installed and is still published.

See Also

Migrating Tables and Fields Between Extensions
Moving Tables and Fields to Extension Up the Dependency Graph
Upgrading Extensions
Publishing and Installing an Extension
JSON Files

Moving Tables and Fields to Extensions Up the Dependency Graph

2/17/2021 • 5 minutes to read • Edit Online

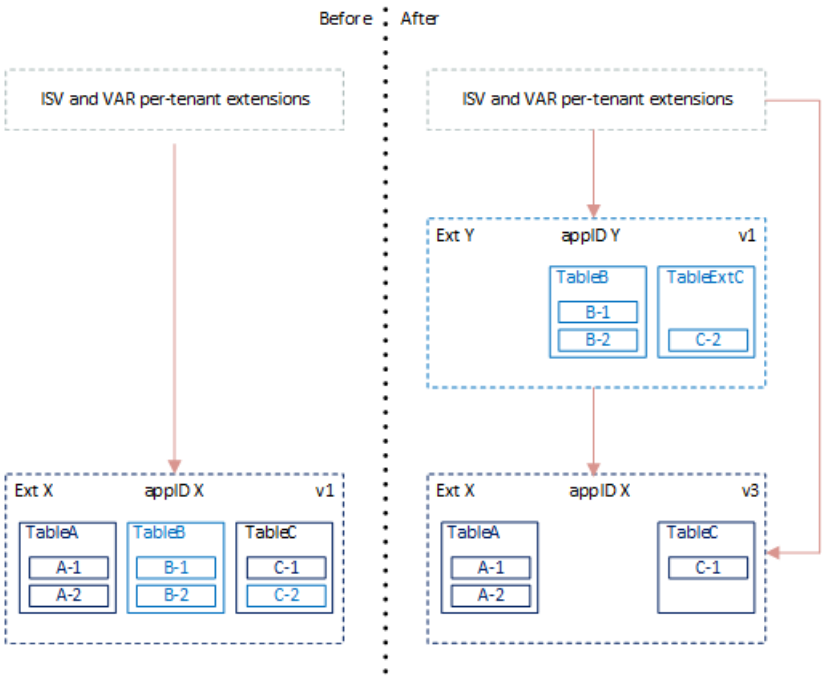
INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

This article explains how to move tables and fields from an extension to another extension that is up the dependency graph.

> **TIP**
> For more information about the general concepts for moving tables and fields between extensions, see Migrating Tables and Fields Between Extensions. This article explains the the difference between moving up and down the dependency graph.

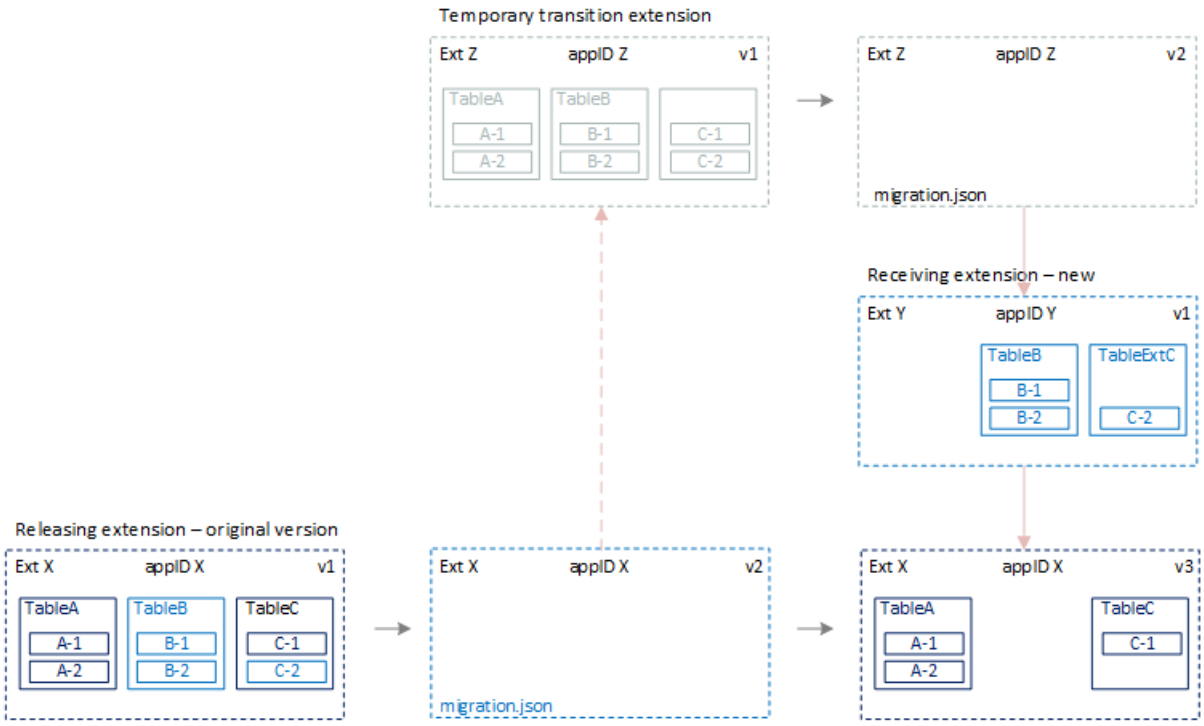
Overview

The steps are based on the example illustrated in the following figure. Although your scenario is different, the concept and process are much the same.



In the example, **TableB** and **Field C-2** are customizations. You'll move these elements from the original extension up to a new extension. This new extension will have a dependency on the original extension. You'll keep **TableA** and **TableC** in the original extension.

To accommodate data migration, you'll have to create an extension that is only used for deployment. This extension is **Ext Z** in the figure. There are two stages of deployment:



- In the first stage, **Ext Z** temporarily takes ownership of tables and fields from **Ext X**.
- In the second stage, **Ext Z** releases ownership to extensions **Ext X** and **Ext Y**. You uninstall and unpublish transition extension **Ext Z** when you finish deployment.

This process is a two-step process because we only support moving down the dependency graph. So instead, the concept is to first move the tables' ownership to an extension above the receiving extensions in the dependency graph. Then, the extensions are moved down. This concept essentially turns the process into a two-step, move-down process.

Ext Z is used just as a temporary extension for moving ownership. So, it only includes schema objects. As a result, customers can't run on the tenant until both steps have been done.

> **NOTE**
> You can only use this process for on-premise solutions.

Prerequisite

If you're moving enum type fields, then your solution must be running on Business Central 2020 release wave 1, version 16.5 or later. For more information, see Known Issues.

Create transition extension (Ext Z v1)

The transition extension will contain replicas of all table object definitions in the releasing extension, except logic code. In the illustration, these objects include **TableA**, **TableB**, and **TableC** and current their field definitions. The transition extension is **Ext Z**.

1. Create an AL project for the transition extension.
2. Add a table object that exactly matches the table object definitions for **TableA**, **TableB**, and **TableC** in the releasing extension.
3. Compile the extension package.
4. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

For purposes of the example, the ID is `11111111-aaaa-2222-bbbb-333333333333`. The value for your extension will be different.

Create empty version of releasing extension (Ext X v2)

In this step, you create a new version of the releasing extension that doesn't contain any objects. It only contains a `migration.json` file that points to **Ext Z**.

1. In the releasing extension AL project, add a migration.json file that points to the ID of the transition extension **Ext Z**.

   ```
   {
   "apprules": [
       {
           "id": "11111111-aaaa-2222-bbbb-333333333333"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.

2. Delete all objects from the extension. The objects include **TableA**, **TableB**, and **TableC**.
3. In the app.json file, increase the `version` value. Ensure that `"target": "OnPrem"`.
4. Compile a new version of the extension package.

Create receiving extension (Ext Y v1)

You now create a new extension that contains the customization you want to move from the releasing. In this example, the customizations include **TableB** and a **TableExtC**.

1. Create an AL project for **Ext Y**.
2. In the app.json file, set up a dependency on the releasing extension **Ext X**.
3. Add a table definition and code for **TableB** that exactly matches the definition in the original releasing extension.
4. Add a table extension object called **TableExtC**. Then, add a field definition for field **C-2** that matches its definition in the original **TableC** object of the releasing extension.
5. Compile the extension package.
6. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `44444444-cccc-5555-dddd-666666666666`. The value for your extension will be different.

Create final version of releasing extension (Ext X v3)

In this step, you create another version of the releasing extension **Ext X**. This version will contain the objects and code that you want to finally publish.

1. Create an AL project for **Ext X**.
2. Add a table definition and code for **TableA** that exactly matches the definition in the original releasing extension.
3. Add a table object for **TableC** and field definition for **C-1** that matches the definitions in the original

TableC object of the releasing extension.

4. In the app.json file, increase the `version` value.
5. Compile the extension package.
6. Make a note of the `ID` of the extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `77777777-eeee-8888-ffff-999999999999`. The value for your extension will be different.

Create new empty version of transition extension (Ext Z v2)

In this step, you create a new version of **Ext Z** that only contains a `migration.json` file. This `migration.json` file points the IDS of **Ext X** and **Ext Y**. The file is used to release ownership.

1. In the extension AL project, add a migration.json file that points to the IDs of the releasing extension **Ext X** and receiving extension **Ext Y**.

   ```
   {
   "apprules": [
       {
           "id": "77777777-eeee-8888-ffff-999999999999"
       }
       {
           "id": "44444444-cccc-5555-dddd-666666666666"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.

2. Delete the object definitions for **TableA**, **TableB**, and **TableC**.
3. In the app.json file, increase the `version` value.
4. Compile the extension package.

Deploy the extensions

1. Uninstall the current version of the releasing extension **Ext X**.
2. Complete the following steps for the first stage of deployment:

   a. Publish the transition extension **Ext Z** and empty version of **Ext X v2**.

   b. Synchronize the transition extension **Ext Z**.

      This step creates empty database tables **TableA**, **TableB**, and **TableC** that are owned by **Ext Z**.

      > **IMPORTANT**
      > Extensions receiving table objects must be synced first. Extension releasing/giving away table objects must be synced last.

   c. Synchronize the releasing extension **Ext X v2**.

      This step will read the migration.json of the extension. Then transfer ownership of the original tables **TableA**, **TableB**, and **TableC** to **Ext Z**.

3. Complete the following steps for the second stage of deployment:

    a. Publish the next version for **Ext Z v2** and **Ext X v3**, and the first version of **Ext Y**.

    b. Synchronize the extensions in the following order: **Ext X v3**, **Ext Y v1**, and **Ext Z v2**.

        Synchronize **Ext Z v2** last. When you synchronize **Ext Z v2**, ownership of the tables is transferred from **Ext Z** to **Ext X** and **Ext Y**.

4. Run Start-NAVAppDataUpgrade cmdlet on the new releasing extension version**Ext X v3**.

    This step basically installs the new extension version. You run a data upgrade because an earlier version has been installed and is still published.

5. Install the new receiving extension **Ext Y v1**.

6. Unpublish both versions of **Ext Z**.

See Also

Migrating Tables and Fields Between Extensions
Moving Tables and Fields to Extension Down the Dependency Graph
Upgrading Extensions
Publishing and Installing an Extension
JSON Files

The Migration.json File

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

Data migration allows you to move table and field data between extensions. The `migration.json` file provides a pointer to the ID of an app that one or more tables will be moved to. This allows you to move table and field data from, for example, a code-customization on the base application to an extension of the base application. The `migration.json` file can be added into the app project of an extension that a table is moved from to specify the ID of the app that the table will be moved to. It can, for example, be placed at the root of the AL project. The `migration.json` file must be created manually following the steps and syntax as described below.

In the extension `app.json` file, ensure that `"target": "OnPrem"`. For more information, see JSON Files.

Creating the migration.json file

1. In the root folder of the app project that will migrate data to a different app project, choose **New File**.
2. Name the file `migration.json`.
3. Edit the file by adding one or more IDs inside the `"apprules":[]` section, such as the following:

```
{
    "apprules": [
        {
            "id": "12345678-abcd-abcd-abcd-1234567890ab"
        }
    ]
}
```

4. Save the `migration.json` file in the project.

You now have the migration file in place for the data migration from one app project to another. This will be used for performing the data migration steps. For more information, see Migrating Tables and Fields Between Extensions.

See Also

JSON Files
Migrating Tables and Fields Between Extensions

Pages Overview

2/17/2021 • 7 minutes to read • Edit Online

In Dynamics 365 Business Central, pages are the main way to display and organize data. Pages are the primary object that a user will interact with and have a different behavior based on the type of page that you choose. Pages are designed independently of the device they are to be rendered on, and in this way the same page can be reused across phone, tablet, and web clients.

A page is defined in code as an object composed of controls, properties, actions, and triggers. You can also use Designer in Dynamics 365 Business Central to create a page. For more information, see Using Designer.

Whether you are creating a new page, or extending an existing page, you will add a new .al file to your project and describe the page object in code. The difference is basically that for a new page, you need to define the entire page, whereas when modifying an existing page, you only add the extra functionality or modify the existing.

The structure of a page is hierarchical and breaks down in to three sections. The first block contains metadata for the overall page. The metadata describes the page type and the source table it is showing data from. The next section; the layout, describes the visual parts on the page. The final section details the actions that are published on the page.

Furthermore, the page has properties. Properties work in the same way for pages as they do for other Dynamics 365 Business Central objects. For more information, see Page Properties.

> **TIP**
> For information about designing pages, see Page Types and Layouts.

Page metadata

For a new page object, you must at least specify the type of page; `PageType` and the data source; `SourceTable` of the page. And you can also set other metadata at the beginning of the declaration of the page object.

```
page 50102 PageName
{
    PageType = List;
    SourceTable = TableName;
    Editable = true;
    ContextSensitiveHelpPage = 'feature-overview';
    ...
}
```

Types of pages

Which page type you choose depends on the application task that you want to support, the content that you want to display, and how you want to display it. The Role Center page is the main or home page and it helps the user focus on the most important daily tasks and activities. Other types of pages, such as list pages or card pages are typically linked from the home page for easy access. The following page types are available:

PAGE TYPE	DESCRIPTION

PAGE TYPE	DESCRIPTION
RoleCenter	The Role Center page is the main page.
Card	A Card page is used to view and edit one record or entity from a table.
CardPart	A Card Part page is used in a FactBox on another page to view or edit additional fields associated with a selected entity in the page.
List	A List page displays content from a table in a list format.
ListPart	Similar to a List page, a List Part page displays content from a table in a list format. The difference is that you use the List part page as another page in a FactBox or as a part of the Role Center page.
ListPlus	A ListPlus page displays content from a table in a list format. The difference from a List page is that the main content is a ListPart, not a Repeater group as the List has it.
Document	A Document page usually consists of two separate pages combined into one, with one page nested in the other. A Document page is suitable for use when you want to display data from two tables that are linked together.
WorkSheet	You use a Worksheet page type for creating worksheet or journal task pages.
ConfirmationDialog	You use the ConfirmationDialog page to display messages or prompt users with a confirmation before they continue with the task that they are working on.
StandardDialog	The StandardDialog is a simple page type that you use when users only need to input data and do not need to perform other actions from the page.
NavigatePage	You use a Navigate page type to create a wizard that leads the user through a sequence of steps for completing a task.
HeadlinePart	You use a HeadlinePart page type to display a set of changing headlines on a Role Center.
API	Pages of this type are used to generate web service endpoints and cannot be shown in the user interface. This page type should not be extended by creating a page extension object. Instead, create a new API by adding a page object.

NOTE

For backwards compatibility we continue to support adding non-part pages as parts. We do, however, recommend that you redesign your page to only use Card part or List part, as we may remove support in a future update.

Page layout

The page layout of the page object determines what the page will look like and is specified in the `layout` section. The `layout` contains one or more `area` sections that define a certain placement on the page.

You can choose between the following `area` categories:

AREA TYPE	PLACEMENT ON THE PAGE
`Content`	The content area displays the content of, for example, a RoleCenter or a List page.
`FactBoxes`	The FactBox area is placed to the right-most side of a page. Displays content related to an item on the main content page.
`RoleCenter`	The RoleCenter is the main page of the application and is used for quick access to frequently used information and tasks.

Page actions

All pages contain menu items and navigation controls called actions. In Dynamics 365 Business Central, actions are displayed at the top of each page in the ribbon or in the navigation pane. The `actions` section of the page describes what the user is able to do on a page and must be designed with the user's need for process support in mind.

Actions can be displayed in the ribbon of all pages and grouped together under the following actions tabs:

- Home
- Actions
- Navigate
- Report

Creating actions can include adding activity buttons/cues to a page, configuring navigation items on a user role center, or adding Reports to a page. To learn how you can enable users to quickly locate the actions they want to use, see Actions.

Adding Help to the page objects

The Business Central user assistance model expects your solution to include tooltips and links to context-sensitive Help. For more information, see User Assistance Model.

Context-sensitive Help

To apply context-sensitive Help to your app, you specify a URL to your Help library in the app.json file, and you then set the relevant target Help files as property values for each of your page objects and page extension objects. Between them, these two settings then give users access to context-sensitive Help for the features in your app at runtime. For more information, see Configure Context-Sensitive Help.

Tooltips

In combination with descriptive captions and instructional text, tooltips are our current implementation of *embedded user assistance*, which is an important principle in today's world of software design. The tooltips are there to help users unblock themselves by providing an answer to the most likely questions the users might have, such as "What data can I input here?" or "What is the data used for?".

The base application has set the Tooltip property for all controls on (almost) all page objects. Most system

actions also include tooltips so that users get a consistent experience. Your extensions are expected to also include tooltips for the same reason. For more information, see ToolTip Property.

Instructional text

The base application has applied instructional text to setup guides and certain other types of page objects. Your extensions are expected to also include instructional text to setup guides for the same reason. For more information, see InstructionalText Property.

Example

The following example shows how you can apply user assistance and link to Help in a page object:

```
page 50101 "Reward Card"
{
    PageType = Card;
    SourceTable = Reward;
    ContextSensitiveHelpPage = 'sales-rewards';

    layout
    {
        area(content)
        {
            group(Reward)
            {
                InstructionalText = 'Fill in the fields so that you can reward customers with discounts.';
                field("Reward Id"; "Reward ID")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the unique ID of the reward.';
                }

                field(Description; Description)
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies what this type of reward is used for.';
                }

                field("Discount Percentage"; "Discount Percentage")
                {
                    ApplicationArea = All;
                    ToolTip = 'Specifies the impact of the reward on the customer''s price.';
                }
            }
        }
    }
}
```

In this example, the app.json file has specified a link to where the *sales-rewards* target file is published, such as `"contextSensitiveHelpUrl": "https://mysite.com/documentation"`.

Best practices for designing pages

We recommend that you simplify the user experience by reducing what users see by default. You can promote the information that the users most frequently need to see and hide the less important information. For example:

- Place common tasks in the ribbon
- Organize information pages under FastTabs and, by default, hide the FastTabs that are infrequently visited.
- Use one to three FactBoxes on a page to provide supplementary information and a place for adding notes
- Add a target Help file for context-sensitive Help for the feature that the page object supports

See Also

Page, Page Fields, and Page Extension Properties
Actions Overview
Using Designer
Page Types and Layouts
Adding a Factbox to a Page
Designing Role Centers
Configure Context-Sensitive Help

Page Object

2/17/2021 • 2 minutes to read • Edit Online

Pages are the main way to display and organize visual data in Dynamics 365 Business Central. They are the primary object that a user will interact with and have a different behavior based on the type that you choose. Pages are designed independently of the device they are to be rendered on, and in this way the same page can be reused across phone, tablet, and web clients.

The structure of a page is hierarchical and breaks down in to three sections. The first block contains metadata for the overall page; the type of the page and the source table it is showing data from. The next section; the layout, describes the visual parts on the page. The final section details the actions that are published on the page.

When developing a solution for Dynamics 365 Business Central, you will follow the code layout for a page as shown in the page example below, but for more details on the individual controls and properties that are available, see Page Property Overview.

If you want to, for example, add functionality to a page that already exists in Business Central, you can create a page extension object that changes an existing page object. For more information, see Page Extension Object. Depending on how much you want to change on an existing page, you can also create a page customization object, which offers modifications on actions and layout. For more information, see Page Customization Object.

> **IMPORTANT**
> Only pages with the Extensible Property set to **true** can be extended.

> **NOTE**
> Extension objects can have a name with a maximum length of 30 characters.

Snippet support

Typing the shortcut `tpage` will create the basic layout for a page object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Views

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. For more information, see Views.

Page example

```
page 50101 SimpleCustomerCard
{
    PageType = Card;
    SourceTable = Customer;
    ContextSensitiveHelpPage = 'my-feature';

    layout
    {
        area(content)
        {
            group(General)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                    CaptionML = ENU = 'Hello';

                    trigger OnValidate()
                    begin
                        if "No." < '' then
                            Message('Number too small')
                    end;
                }

                field(Name; Name)
                {
                    ApplicationArea = All;
                }
                field(Address; Address)
                {
                    ApplicationArea = All;
                }
            }
        }
    }
    actions
    {
        area(Navigation)
        {
            action(NewAction)
            {
                ApplicationArea = All;
                RunObject = codeunit "Document Totals";
            }
        }
    }
}
```

See Also

AL Development Environment
Views
Adding Help Links from Pages, Reports, and XMLports
Page Extension Object
Page, Page Fields, and Page Extension Properties
Page Properties
Developing Extensions
Configure Context-Sensitive Help

Page Extension Object

2/17/2021 • 4 minutes to read • Edit Online

The page extension object extends a Dynamics 365 Business Central page object and adds or overrides the functionality.

The structure of a page is hierarchical and breaks down into three sections. The first block contains metadata for the overall page; the type of the page and the source table it is showing data from. The next section; the layout, describes the visual parts on the page. The final section details the actions that are published on the page.

For more information about the Page and Page Extension objects, see Pages Overview.

IMPORTANT

Only pages with the Extensible Property set to **true** can be extended.

NOTE

Extension objects can have a name with a maximum length of 30 characters.

IMPORTANT

The API page type should not be extended by creating a page extension object. Instead, create a new API by adding a page object.

NOTE

Modifying actions in Cue groups on page extensions is not supported.

Snippet support

Typing the shortcut `tpageext` will create the basic layout for a page extension object when using the AL Language extension in Visual Studio Code.

TIP

Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Views

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. For more information, see Views.

Using keywords to place actions and controls

You can use the following keywords in the `layout` section to place and move fields and groups on the page extension. Similarly, in the `actions` section, you use these keywords to place actions in the ribbon.

KEYWORDS	SYNTAX	APPLIES TO
`addfirst`	`addfirst(Anchor)`	**Anchor**: areas and groups
`addlast`	`addlast(Anchor)`	**Anchor**: areas and groups
`addafter`	`addafter(Anchor)`	**Anchor**: controls, actions, and groups
`addbefore`	`addbefore(Anchor)`	**Anchor**: controls, actions, and groups
`movefirst`	`movefirst(Anchor; Target1, Target2)`	**Anchor**: area, group **Target**: list of actions or list of controls
`movelast`	`movelast(Anchor; Target1, Target2)`	**Anchor**: area, group **Target**: list of actions or list of controls
`moveafter`	`moveafter(Anchor; Target1, Target2)`	**Anchor**: controls, actions, and groups **Target**: list of actions or list of controls
`movebefore`	`movebefore(Anchor; Target1, Target2)`	**Anchor**: controls, actions, and groups **Target**: list of actions or list of controls
`modify`	`modify(Target)`	**Target**: controls, actions, and groups

Example

To modify the existing fields and groups on a page, you use the `modify` keyword. See the code snippet below for `addlast`, `modify` and `action` syntax. In the following example, the `actions` section creates a new group in the ribbon and places it last in the `Creation` group.

```
pageextension 70000020 CustomerCardExtension extends "Customer Card"
{
    layout
    {
        // Adding a new control field 'ShoeSize' in the group 'General'
        addlast(General)
        {
            field("Shoe Size"; ShoeSize)
            {
                Caption = 'Shoe size';

                trigger OnValidate();
                begin
                    if ShoeSize < 10 then
                        Error('Feet too small');
                end;
            }
        }

        // Modifying the caption of the field 'Address 2'
        modify("Address 2")
        {
            Caption = 'New Address 2';
        }

        // Moving the two fields 'CreditLimit' and 'CalcCreditLimitLCYExpendedPct'
        // to be the first ones in the 'Balance' group.
        movefirst(Balance; CreditLimit, CalcCreditLimitLCYExpendedPct)
    }
    actions
    {
        // Adding a new action group 'MyNewActionGroup' in the 'Creation' area
        addlast(Creation)
        {
            group(MyNewActionGroup)
            {
                action(MyNewAction)
                {
                    Caption = 'My New Action';

                    trigger OnAction();
                    begin
                        Message('My message');
                    end;
                }
            }
        }
    }
}

tableextension 70000020 CustomerTableExtension extends Customer
{
    fields
    {
        // Adding a new table field in the 'Customer' table
        field(50100; ShoeSize; Integer) { }
    }
}
```

Page extension examples

In the following example, we use a table extension to extend the Customer table with a new field named `ShoeSize` of the datatype Integer. Then we create a page extension object that extends the Customer Card page object by adding a field control `ShoeSize` to the `General` group on the page. The field control is added as the

last control in the group using the `addlast` method. The example also illustrates how to add a display-only control to the page. In the actions area, you can see what the syntax looks like for actions that execute triggers and actions that run objects.

```
tableextension 50115 RetailWinterSportsStore extends Customer
{
    fields
    {
        field(50116;ShoeSize;Integer)
        {
            Caption = 'ShoeSize';

            trigger OnValidate();
            begin
                if (rec.ShoeSize < 0) then
                begin
                   message('Shoe size not valid: %1', rec.ShoeSize);
                end;
            end;
        }
    }

    procedure HasShoeSize() : Boolean;
    begin
        exit(ShoeSize <> 0);
    end;

    trigger OnBeforeInsert();
    begin
        if not HasShoeSize then
            ShoeSize := Random(42);
    end;
}

pageextension 50110 CustomerCardExtension extends "Customer Card"
{
    layout
    {
        addlast(General)
        {
            // control with underlying datasource
            field("Shoe Size"; ShoeSize)
            {
                ApplicationArea = All;

                trigger OnValidate();
                begin
                    if ShoeSize < 10 then
                        Error('Feet too small');
                end;
            }

            // display-only control (without underlying datasource)
            field(ShoesInStock; 10)
            {
                ApplicationArea = All;
                Caption = 'Shoes in stock';

            }
        }

        modify("Address 2")
        {
            Caption = 'New Address 2';
        }
    }
```

```
    actions
    {
        addlast(Creation)
        {
            group(MyActionGroup)
            {
                Action(MyAction1)
                {
                    ApplicationArea = All;
                    Caption = 'Hello!';

                    trigger OnAction();
                    begin
                        Message('My message');
                    end;
                }

                Action(MyAction2)
                {
                    ApplicationArea = All;

                    // Run page to test how actions work
                    RunObject = page "Absence Registration";
                }
            }
        }
    }
}
```

You can reference Report and XMLPort objects and use these objects in the **RunObject** property, as well as, declare variables of the types **Report** and **XMLPort** and call AL methods on them. This page extension object extends the Customer List page object by adding two actions; the first action calls the **Customer - List** report, the second action calls the **Export Contact** XMLPort.

```
pageextension 50114 AddCustomerReport extends "Customer List"
{
    actions
    {
        AddLast("&Customer")
        {
            action("Customer List Report")
            {
                trigger OnAction();
                var
                    rep : Report "Customer - List";
                begin
                    rep.Run;
                end;
            }

            action("Export Contact List")
            {
                trigger OnAction();
                var
                    xml : XmlPort "Export Contact";
                begin
                    xml.Run;
                end;
            }
        }
    }
}
```

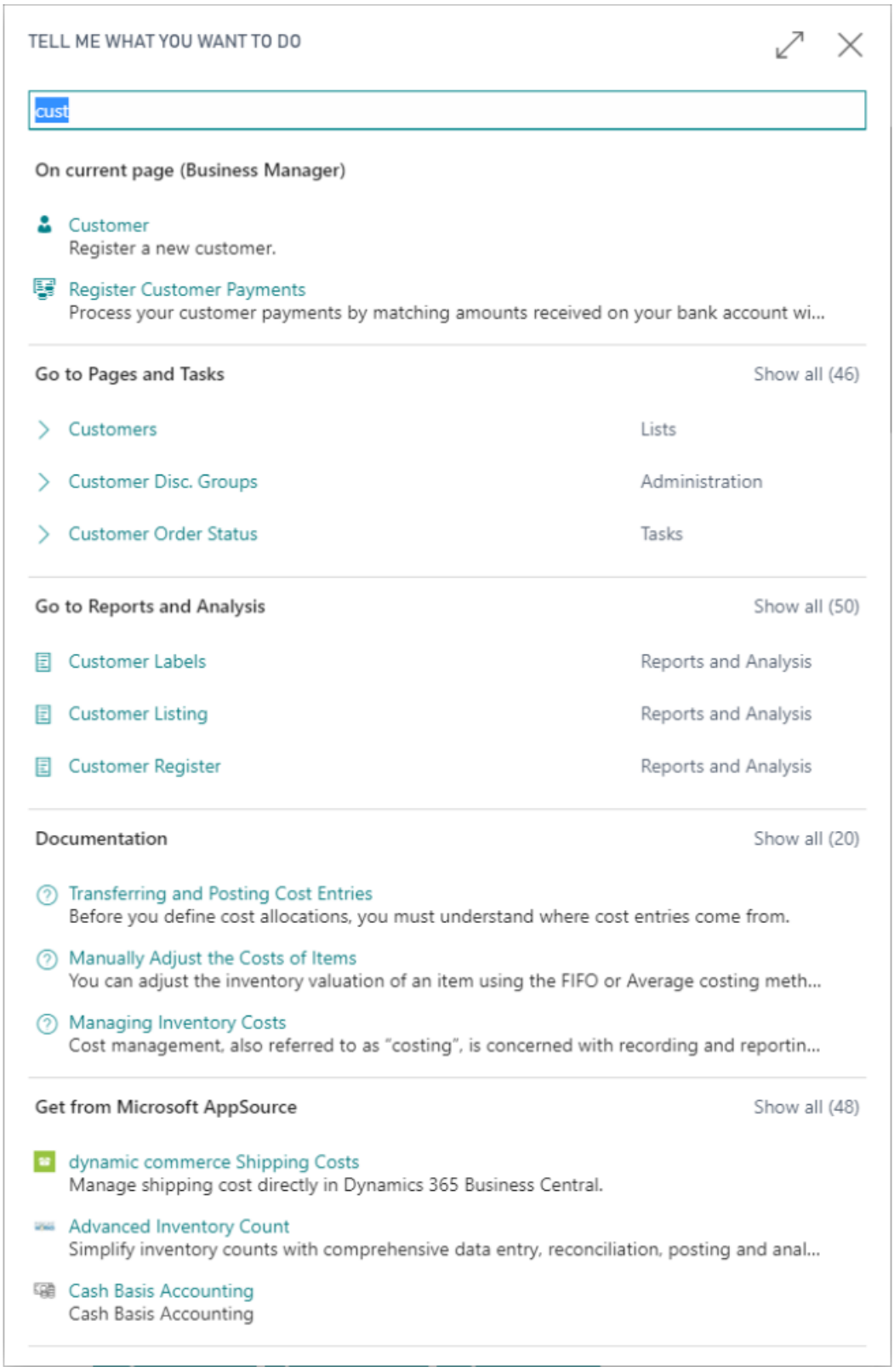
See Also

Page Object
Views
Page, Page Fields, and Page Extension Properties
Extending Pages Previously Based on the Date Virtual Table Developing Extensions
AL Development Environment

Adding Pages and Reports to Tell me

2/17/2021 • 3 minutes to read • Edit Online

The Business Central client includes the **Tell me** feature that lets users find objects and online help articles by entering search terms. When you have added a page or a report in your extension, you most likely want it to be discoverable to users in **Tell me**. In AL, you make a page or report searchable from **Tell me** by setting the UsageCategory property in code. The **UsageCategory** setting will make the page or report searchable, and the value chosen for the setting will further sub categorize the item.



Tell me finds pages and reports by searching the captions that are specified on page and report objects by the CaptionML property.

Working with the UsageCategory property

When you create a Page or a Report, you add the UsageCategory Property. If the **UsageCategory** is set to **None**, or if you do not specify **UsageCategory**, the page or report will not show up when you search in Dynamics 365 Business Central.

UsageCategory property values

The values for the **UsageCategory** property are listed below. The sub category will help the user navigate through the search results and it is a best practice to be consistent when categorizing the pages and the reports that you add. A consistent approach will help guiding the user and improve productivity.

VALUE	DESCRIPTION
None	The page or report is not included in search.
Lists	The page or report is listed as **Lists** under the **Pages and Tasks** category.
Tasks	The page or report is listed as **Tasks** under the **Pages and Tasks** category.
ReportsAndAnalysis	The page or report is listed as **Reports and Analysis** under the **Reports and Analysis** category.
Documents	The page or report is listed as **Documents** under the **Reports and Analysis** category.
History	The page or report is listed as **Archive** under the **Reports and Analysis** category.
Administration	The page or report is listed as **Administration** under the **Pages and Tasks** category.

Adding additional search terms

You can specify other words or phrases that can help users find a page or report by using the AdditionalSearchTerms and AdditionalSearchTermsML properties. If the page or report is searchable by **Tell me** (that is, th **UsageCategory** property is set a value other than `None`), the search terms specified by these properties are used in addition to the caption of the page or report. These properties are useful when the caption does not always reflect what users will look for. A good example of this in Business Central is pages and reports associated with **Item**. Users unfamiliar with Business Central might look for 'product' or 'merchandise' instead of 'item'.

> **NOTE**
>
> For Business Central on-premises, the Business Central Web Server configuration file (navsettings.json) includes a setting called `UseAdditionalSearchTerms` that enables or disables the use of additional search terms by the **Tell me**. For more information, see Configuring Business Central Web Server Instances.

Example

The following example creates a `SimpleItemList` page and sets a `UsageCategory` property to the page, so that the `SimpleItemList` page is discoverable through search using the **Tell me** feature. Also, the example sets the `AdditionalSearchTerms` property to add two search terms for the page.

```
page 50210 SimpleItemList
{
    PageType = List;
    SourceTable = Item;
    UsageCategory = Lists;
    AccessByPermission = page SimpleItemList = X;
    ApplicationArea = All;
    AdditionalSearchTerms = 'product, merchandise';

    layout
    {
        area(content)
        {
            group(General)
            {
                field("No.";"No.") {}
                field(Name;Name) {}
                field(Description;Description) {}
            }
        }
    }
}
```

Optional settings

In addition to making a page or report searchable, you can control the access of an object by providing **Read**, **Insert**, **Modify**, **Delete**, and **Execute** (RIMDX) permissions by adding the AccessByPermission property. Likewise, control the application area access on the specified object by adding the ApplicationArea Property.

The **AccessByPermission** property and **ApplicationArea** property are the optional settings, which can be applied with the **UsageCategory** property. These settings are used to set restrictions on an object when you enable the Search functionality.

Working in the Dynamics NAV Development Environment

> **NOTE**
> Dynamics NAV Development Environment is **DISCONTINUED AFTER:** Business Central Spring 2019.

If you are using the Dynamics NAV Development Environment, you can also set **UsageCategory**, **AdditionalSearchTerms**, **AccessByPermission**, and **ApplicationArea** properties on pages and reports to control their search.

After you change these properties by using the Dynamics NAV Development Environment, before the changes take effect in the client, you must run **Build Object Search Index** from the **Tools** menu.

See Also

Adding Menus to the Navigation Pane
UsageCategory Property
Page Object
Report Object
AL Development Environment

Designing Role Centers

2/17/2021 • 8 minutes to read • Edit Online

The strength of Dynamics 365 is its role-tailored experience that helps users focus on the work that is important to them. The Role Center is an integral part of the role-tailored experience. And as a developer, role-tailoring should be the foundation for your Role Center design.

About the Role Center

The Role Center is the user's entry point and home page for Dynamics 365. You can develop several different Role Centers, where each Role Center is customized to the profile of the intended users. For example, you could have Role Centers that target the different levels within an organization, such business owners, department leads, and information workers.

Role Centers are based on a user-centric design model. You should design a Role Center to give users quick access to the information that is most important to them in their daily work - displaying information that is pertinent to their role in the company and enabling them to easily navigate to relevant pages for viewing data and performing tasks.

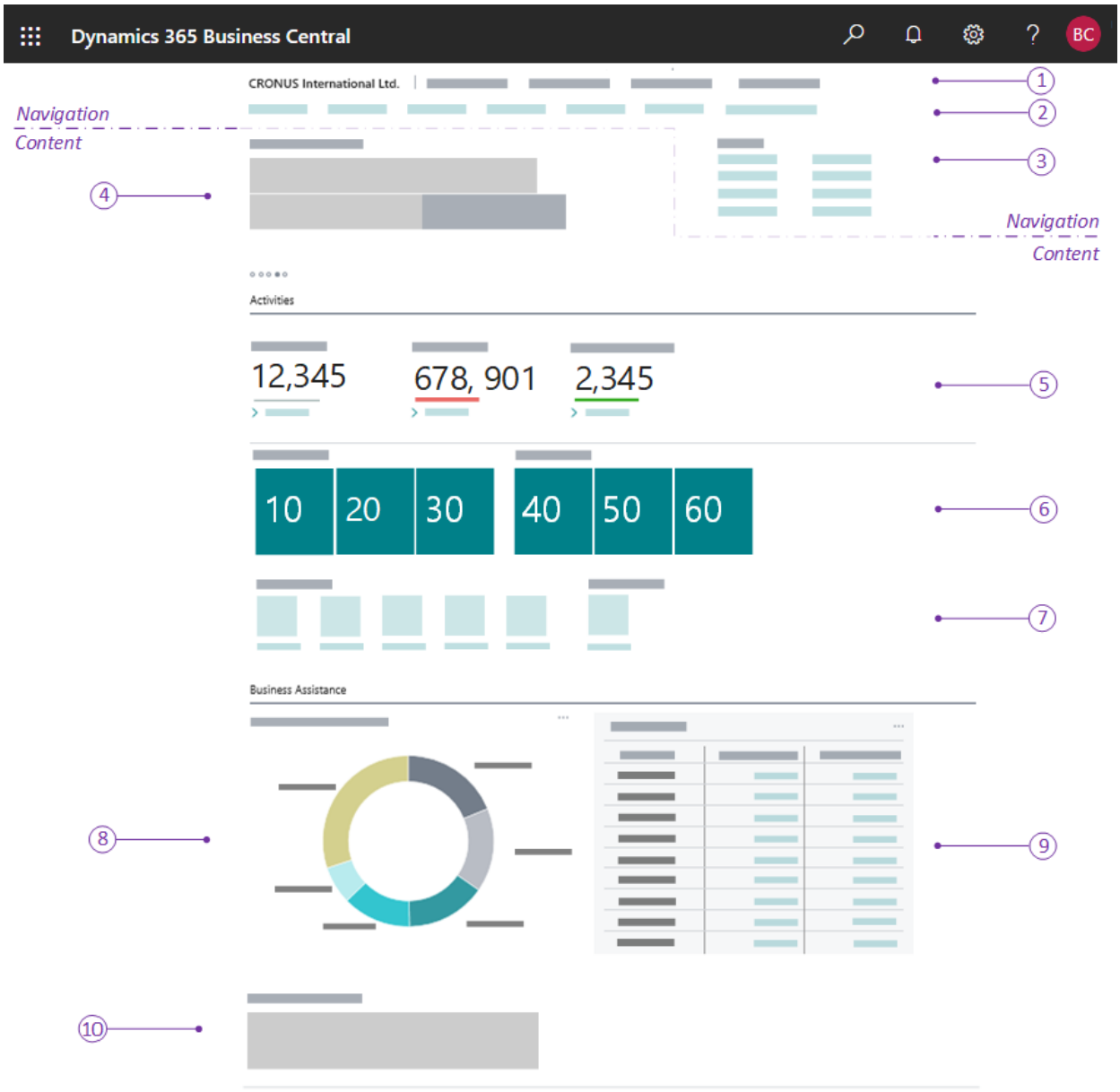
Customizing a Role Center from the client

In the client, users who work across multiple roles can easily switch Role Centers to shift their focus to different tasks. Users can also personalize their Role Centers by rearranging or hiding content as they like. For more information, see Personalizing Your Workspace.

As a developer or administrator, you can use Designer to customize a Role Center the same way that individual users personalize their own workspaces. The difference is that changes you make are applied to all users assigned to the Role Center. For more information, see Using Designer.

Role Center structure

A Role Center is defined by page that has the PageType property set to `RoleCenter`. The Role Center page is divided into two main areas: navigation/actions area and content area. The following figure illustrates the general layout and elements of a Role Center page.



For a simple code example of a role center page, see Simple Role Center Code Example.

Navigation and Actions area

The navigation and actions area appears at the top of the Role Center page, and provides links to other objects, such as pages, reports, and codeunits. You define the navigation area by adding actions to the Role Center page code, under the `actions` control in the page code. The navigation and actions area is subdivided into smaller areas by using different `area()` controls as described in the following table:

NO.	AREA	DESCRIPTION	USAGE GUIDELINES

NO.	AREA	DESCRIPTION	USAGE GUIDELINES
1	Navigation menus	The top-level navigation consists of one or more root menu items that expand to display links to other pages. The links can be grouped into sub-menus, enabling you to create a logical hierarchy. The pages targeted by the links will open in the content area of the Role Center. You define this area with an `area(Sections)` control in the page code.	The top-level navigation should provide access to relevant entity lists for the role's areas of business. For example, typical root items for a business manager could be finance, sales, and purchasing. You should place the root items in order of importance, starting from the left.
2	Navigation bar	The second-level navigation displays a flat list of links to other pages. The pages targeted by the links will open in the content area of the Role Center. You define this area with an `area(embedding)` control in the page code.	You should use these items to link to users' most useful entity lists in their business process. For example, with a business manager, these could be links to customers, sales orders, and bank accounts. You should place items in the order that reflects the business process sequence. Try to limit the number of second-level items, and consider placing items in the top-level navigation instead, if the number gets too large.
3	Action bar	The actions bar provides links to pages, reports, and codeunits. The links can be displayed on the root-level or grouped in a sub-menu. The objects targeted by these links will open in a separate window in front of the Role Center page. You can define the actions by using the three different `area()` controls that are described below:	The action area is designed for running the most important or most often used tasks and operations required by users. Actions will typically target card type pages that enable users to create new entities, such as customers, invoices, and sales orders, or run reports. Place the most important action at the root-level, and group closely related actions in a sub-menu.
		`area(creation)` - Actions in this control will appear first in the action area, and will display with a plus (+) icon.	Use this control to target pages that enable the user to create new entities.

NO.	AREA	DESCRIPTION	USAGE GUIDELINES
		`area(processing)` - Actions in this control will appear after the `area(creation)` items. You can group actions in sub-menus by using a `group` control.	Use this control to target pages that are associated with the work flow for processing documents, such as payments or sales orders. Use the `group` control to organize similar actions under a common parent.
		`area(reporting)` - Actions in this control will appear last in the action area. They display with a default report icon.	Use this control to target report objects.

For more information about navigation, see Adding to Navigation.

Behavioral points of interest

- The order of the `area()` controls in the page code is not important. However, the order of the individual actions and groups is important because they will appear in the order in which they appear in page code.
- In page code, if the first part in the content area is a Headline part, then in the client, the actions area will be automatically positioned either to the right of the Headline part or after the Headline part, depending on the browser window size. If the first part is not a Headline, the actions area will appear directly after the navigation area, and extend the width of the workspace.

Content area

The content area consists of one or more parts that display content from other pages. Unlike the navigation and actions area that is completely defined in the Role Center page code, the content area consists of self-contained, independent page part objects that can be used across Role Centers and in other pages. You define the content area by adding a `layout` control in the page code, and then a `part` control for each individual part to display.

The following table describes some of the most common parts for Role Centers, as illustrated in the previous figure.

NO.	ELEMENT	DESCRIPTION	MORE INFORMATION	
4	Headline	Displays a series of automatically changing headlines that provide users with up-to-date information and insight into the business and daily work. This is created by a `HeadlinePart` page type.	Creating Role Center Headlines	

NO.	ELEMENT	DESCRIPTION	MORE INFORMATION	
5	Wide data cues	A set of cues for displaying large numbers, like monetary values. This is created by using a `cuegroup` control on a `CardPart` page type, where the Layout property is set to `wide`.	Wide Cues	
6	Data cues	Provide a visual representation of aggregated business data, such as the number of open sales invoices or the total sales for the month. These are created by using a `cuegroup` control on a `CardPart` page type.	Creating Cues	
7	Action cues	Tiles that link to tasks or operations, like opening another page, starting a video, targeting another URL, or running code. These are created by using a `cuegroup` control on a `CardPart` page type	Action Tiles	
8	Chart	A graphical and interactive representation of your business data that can be sourced by a custom business chart control add-in or an embedded Power BI report.		
9	CardPart or ListPart page	Displays data fields in a form or tabular layout.	Page Object	
10	Control add-in	Displays custom content by using HTML-based control add-in.	Control Add-in Object	

Behavioral points of interest

- In general, the parts will appear in the client according to the order in which they are defined in the Role

Center page code and will automatically rearrange horizontally and vertically to fill the available workspace.

- However, in the Web client, page parts that contain cues are automatically grouped under a common **Activities** section, no matter where they are placed in the code. All other page parts are grouped under the **Business Assistance** section. Within **Activities** and **Business Assistance** sections, the parts will arrange according to the order in which they are defined in the page code.

Development tips for overall page design

- Do not apply grouping to parts in the content area because this prevents parts from flowing to fill the available space. This gives the best experience to users with different screen resolutions or those on mobile devices.
- To achieve the best readability and discoverability, place Headlines first, followed by cues, and then the remaining parts.
- You cannot add custom logic directly to a Role Center page code. Code is limited to defining navigation, actions, and parts. All other code is ignored.
- Role Centers can be highly specialized, in the fact that all navigation, actions, and content is optional. For example, you could have a single part that fills the entire workspace.

Design for all display targets

- Role Center pages are also the primary entry point on mobile devices. Mobile devices will display the same content as the Web client, but is presented in a different way to suit how users hold and interact with their mobile device.
- You can preview how your Role Center will look on mobile devices directly in Designer.
- Some limitations on mobile devices include the following:
  - On tablets, there is a limit on the number of cues that can be displayed.
  - On phones, there is a limit on the number of parts in the content area that can be displayed.
  - Role Center pages cannot be displayed when they are embedded in Outlook or SharePoint.

Using the Role Center in the client

To use or test the new Role Center in the client, you must first associate the Role Center page with a profile. Profiles define user roles and each profile is associated with a single Role Center page. Create a new profile object that references your page. Then, go to **My Settings** and select the new profile.

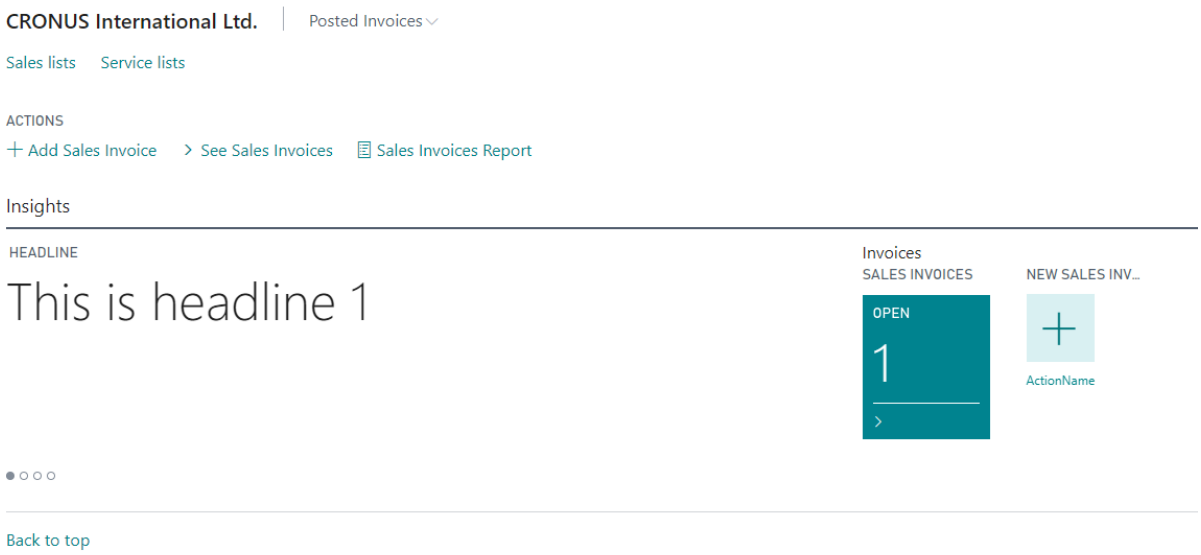
See Also

AL Development Environment
Page Types and Layouts
Page Extension Object
Actions Overview
Adding Pages and Reports to Tell Me
Personalizing Your Workspace
Using Designer

Simple Role Center Code Example

2/17/2021 • 2 minutes to read • Edit Online

The AL code in this article creates a simple Role Center customized for users assigned to a new profile.



For a more detailed explanation of Role Centers, see Designing Role Centers.

This example uses the **RoleCenterHeadline** page code example to display the headline and the **SalesInvoiceCuePage** page and the following code example for the Cue and Action tile.

```
page 50125 MyRoleCenter
{
    PageType = RoleCenter;
    Caption = 'My Role Center';

    layout
    {
        area(RoleCenter)
        {
            group(Group1)
            {
                part(Part1; RoleCenterHeadline)
                {
                    ApplicationArea = All;
                }

                part(Part2; SalesInvoiceCuePage)
                {
                    Caption = 'Invoices';
                }
            }
        }
    }


    actions
    {
        area(Sections)
        {
            group(PostedInvoices)
            {
                Caption = 'Posted Invoices';
```

```
            Caption = 'Posted Invoices';
            Image = RegisteredDocs;
            action(PostedServiceInvoices)
            {
                Caption = 'Posted Service Invoices';
                RunObject = Page "Posted Service Invoices";
                ApplicationArea = All;
            }

            action(PostedSalesInvoices)
            {
                Caption = 'Posted Sales Invoices';
                RunObject = Page "Posted Sales Invoices";
                ApplicationArea = All;
            }

            group(SalesDocuments)
            {
                Caption = 'Sales Documents';
                action("Sales Document Entity")
                {
                    ApplicationArea = All;
                    RunObject = page "Sales Document Entity";
                }
                action("Sales Document Line Entity")
                {
                    ApplicationArea = All;
                    RunObject = page "Sales Document Line Entity";
                }
            }
        }
    }

    area(Embedding)
    {

        action(Sales)
        {
            Caption = 'Sales lists';
            RunObject = Page "Sales list";
            ApplicationArea = All;
        }

        action(Services)
        {
            Caption = 'Service lists';
            RunObject = Page "Service list";
            ApplicationArea = All;

        }


    }

    area(Processing)
    {
        action(SeeSalesInvoices)
        {
            Caption = 'See Sales Invoices';
            RunObject = Page "Posted Sales Invoices";
        }

    }

    area(Creation)
    {
        action(AddSalesInvoice)
        {
            Caption = 'Add Sales Invoice';
            Image = NewInvoice;
```

```
                Image = NewInvoice;
                RunObject = Page "Sales Invoice";
                RunPageMode = Create;
            }
        }

        area(Reporting)
        {
            action(SalesInvoicesReport)
            {
                Caption = 'Sales Invoices Report';
                Image = "Report";
                RunObject = Report "Sales - Invoice";
            }
        }
    }
}

// Creates a profile that uses the Role Center
profile MyProfile
{
    ProfileDescription = 'Sample Profile';
    RoleCenter = MyRoleCenter;
    Caption = 'My profile';
}
```

See Also

AL Development Environment
Page Extension Object
Actions Overview
Adding Pages and Reports to Tell Me
Personalizing Your Workspace
Using Designer

Adding Menus to the Navigation and Actions Area

2/17/2021 • 3 minutes to read • Edit Online

The navigation area appears at the top of the Dynamics 365 Business Central window, and contains multiple sections that enable users to quickly navigate and perform actions in Dynamics 365 Business Central. In the client, the navigation area is separated into three separate areas: navigation menu, navigation area, and actions area. For an illustration that identifies the different areas in a Role Center, see Designing Role Centers. In AL, these areas are defined by the `area()` control, as described in the sections that follow.

Adding to the navigation menu

The top-level navigation area is referred to as the navigation menu. The navigation menu contains one or more root menu items that expand to display a set of links to pages, and other objects like reports, XMLPorts, and codeunits. These links are defined by `action()` controls. You can also group `action()` controls in sub-menus. This enables you to create a logical hierarchy that matches the needs of the user role. The pages targeted by the links in the navigation menu will open in the content area of the Role Center.

> **NOTE**
> The Dynamics NAV Client connected to Business Central doesn't support sub-menus in the navigation menu.

You define the navigation menu by using an `area(Sections)` control in the page code.

Example

The example below adds the root menu item called `My Customers` to the navigation menu of the **Sales Order Processor** Role Center. The `My Customers` menu item contains two actions, the `Customer Bank Account List` and `Customer Ledger Entries` actions, which open corresponding page objects. The `My Customers` menu item also includes a group that contains two other actions, which open sales-related documents.

```
pageextension 50120 ExtendNavigationArea extends "Order Processor Role Center"
{

    actions
    {
        addlast(Sections)
        {
            group("My Customers")
            {
                action("Customer Bank Account List")
                {
                    RunObject = page "Customer Bank Account List";
                    ApplicationArea = All;
                }
                 action("Customer Ledger Entries")
                {
                    RunObject = page "Customer Ledger Entries";
                    ApplicationArea = All;
                }

                // Creates a sub-menu
                group("Sales Documents")
                {
                    action("Sales Document Entity")
                    {
                        ApplicationArea = All;
                        RunObject = page "Sales Document Entity";
                    }
                    action("Sales Document Line Entity")
                    {
                        ApplicationArea = All;
                        RunObject = page "Sales Document Line Entity";
                    }
                }
            }
        }
    }
}
```

You can also enable pages and reports to appear in the Dynamics 365 Business Central search for a quick navigational support. For more information, see Adding Pages and Reports to Tell Me.

Adding to the navigation bar

The second-level navigation is referred to as the navigation bar. The navigation bar offers a flat list of links to other pages. These should be the most relevant pages needed for a user's business process. We recommend to have only the most important items on this level and to place the others in the top-level navigation instead.

You define the navigation bar by using an `area(Embedding)` control in the page code.

Example

The following code adds a new link to the navigation bar by defining this area with an `area(Embedding)` control in the page code. The object targeted in this case is the `Sales Cycles` page and it will appear as the last one.

```
...
addlast(Embedding)
{
    action("Sales Cycles")
    {
        RunObject = page "Sales Cycles";
        ApplicationArea = All;
    }
}
```

Adding to actions

The actions area displays the most important or most often used tasks and operations required by users. It contains links to pages, reports, and codeunits. The links are placed on the root-level, and they can be grouped in a submenu.

You can define the actions by using three different `area()` controls.

The first action area that appears at the top of the Role Center page is `area(Creation)`. The following example adds the item last, and it allows opening the `Sales Journal` page.

Example

```
...
addlast(Creation)
{
    action("Sales Journal")
    {
        ApplicationArea = All;
        RunObject = page "Sales Journal";
    }
}
```

The actions in the `area(Processing)` control appears after the `area(Creation)` items. The example below shows how you can use the group control to organize similar actions under a common parent. The created group is placed at the end of this action area, and it targets pages needed for processing sales documents.

Example

```
...
addlast(Processing)
{
    group(Documents)
    {
        action("Sales Document Entity")
        {
            ApplicationArea = All;
            RunObject = page "Sales Document Entity";
        }
        action("Sales Document Line Entity")
        {
            ApplicationArea = All;
            RunObject = page "Sales Document Line Entity";
        }
    }
}
```

The actions in the `area(Reporting)` control will appear last in the action area and they display with a default report icon. This control's purpose is to target report objects and the following example opens the

`Customer Sales Statistics` report.

Example

```
...
addlast(Reporting)
{
    action("Customer Statistics")
    {
        ApplicationArea = All;
        RunObject = report "Customer Sales Statistics";
    }
}
```

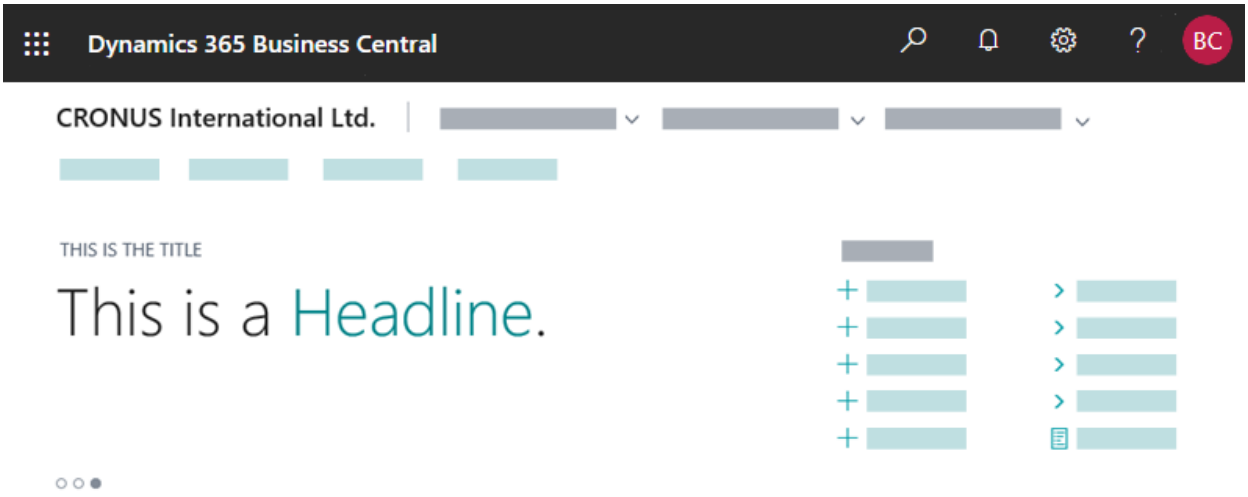
See Also

AL Development Environment
Page Extension Object
Actions Overview
Adding Pages and Reports to Tell Me

Creating a Role Center Headline

2/17/2021 • 5 minutes to read • Edit Online

You can set up a Role Center to display a series of headlines, where headlines appear one at a time for a predefined period of time before displaying the next.



The headlines can provide users with up-to-date information and insight into the business and daily work. Typical categories of headlines might include:

- My performance
- My workday
- Organizational health
- Productivity tips
- Cross-tenant insights (performance relative to peers)
- Getting started information

> **IMPORTANT**
> Headlines will only appear in the Web client; they will not be shown on other client types.

Design concept

In development

In short, the Headline is basically a page that contains one or more fields. The page must be the **HeadlinePart** type page. Each field defines an individual headline to be displayed. The source for a field can be an expression or a field in an underlying table.

- The **HeadlinePart** page is designed for Role Centers, that is, pages that have the type **RoleCenter**. If you use a **HeadlinePart** page on another page type, the part will not render in the client.
- Using the OnDrillDown trigger, headlines can be made interactive, meaning that users can select the headline to dig deeper into numbers or values that are shown in the headline or link to another page or URL.
- You can dynamically toggle visibility of a specific headline, for example based on its relevancy, by setting the Visible property on the field.

- There are only a few field properties that apply to fields that are used on a **HeadlinePart** type page, including Expression, Visible, ApplicationArea, Drilldown, and DrillDownPageID. All other properties are ignored.

In the client

The Role Center will start by displaying the first visible headline that is defined on the HeadlinePart page. The headline will appear for 5 seconds, then the next headline will appear for 5 seconds, and so on. When all the headlines have been displayed, it will cycle back to the first headline, and continue from there.

- If a headline is interactive, users can select the headline to open the target defined in the headline.
- Users can pause on a headline by pointing to it.
- Users can manually switch among headlines by selecting a corresponding dot that is displayed under the headlines.
- Users can personalize their Role Center to show or hide the Headline part as they like.

Creating a HeadlinePart page

1. Implement the logic that resolves field expressions for the headlines that you will use on the page.

   You can apply more flexible and complex patterns, such as having data tables drive the text, drill-down and relevance engine for headlines.

2. Create a page that has the PageType property set to `HeadlinePart`.

3. For each headline, add a field, and set the Expression property. The order of the fields, determines the order in which they appear.

   The following example shows the AL code for a simple **HeadlinePart** page that consists of four fields that display static text.

```
page 50100 RoleCenterHeadline
{
PageType = HeadLinePart;

layout
{
    area(content)
    {
        field(Headline1; hdl1Txt)
        {

        }
        field(Headline2; hdl2Txt)
        {

        }
        field(Headline3; hdl3Txt)
        {

        }
        field(Headline4; hdl4Txt)
        {

        }
    }
}

var
    hdl1Txt: Label 'This is headline 1';
    hdl2Txt: Label 'This is headline 2';
    hdl3Txt: Label 'This is headline 3';
    hdl4Txt: Label 'This is headline 4';
}
```

4. You can now add the **HeadlinePart** page to the **RoleCenter** page.

Constructing Headlines with the Expression property

The `Expression` property supports the following syntax that enables you to specify a title for the headline, the headline text itself, and emphasize a string of text in the headline:

```
'<qualifier>Title</qualifier><payload>This is the <emphasize>Headline</emphasize>.</payload>'
```

TAG	DESCRIPTION
`<qualifier></qualifier>`	Specifies the title that appears above the headline. If you omit this tag, the text **HEADLINE** will be used by default.
`<payload></payload>`	Specifies the actual headline text.
`<emphasize></emphasize>`	Applies the style to the text.

The `Expression` property must evaluate to the correct syntax. For example, looking back at the previous example, the label `hdl1Txt` could be:

```
hdl1Txt: Label '<qualifier>The first headline</qualifier><payload>This is the <emphasize>Headline
1</emphasize>.</payload>';
```

Making headlines interactive

You can use the OnDrillDown trigger of a headline field to link the headline to more details or relevant information about what is shown in the headlines. For example, if the headline announced the largest sales order for the month, you could set up the headline to open a page that shows a sorted list of sales order for the month.

The following code uses the OnDrillDown trigger to link `Headline1` to the Dynamics 365 online help.

```
field(Headline1; hdl1Txt)
{
    trigger OnDrillDown()
    var
        DrillDownURLTxt: Label 'https://go.microsoft.com/fwlink/?linkid=867580', Locked = True;
    begin
        Hyperlink(DrillDownURLTxt)
    end;
}
```

Changing the visibility of headlines

You can use the Visible property to show or hide headlines that are defined on the **HeadlinePart** page. With the `Visible` property, you can show or hide the control either statically by setting the property to **true** or **false**, or dynamically by using a `Boolean` variable.

Static visibility

With static visibility, you can simply set the `Visible` property on specific fields. For example, following code hides `Headline3`:

```
{
    field(Headline1; hdl1Txt)
    {

    }
    field(Headline2; hdl2Txt)
    {

    }
    field(Headline3; hdl3Txt)
    {
         Visible=false;
    }
    field(Headline4; hdl4Txt)
    {
    }
}
```

By adding fields under `Group` controls, you can hide or show more than one headline by setting the `Visible` property on the `Group` control. For example, the following code hides headings `Headline3` and `Headline4`:

```
group(Group1)
{
    field(Headline1; hdl1Txt)
    {
    }
    field(Headline2; hdl2Txt)
    {

    }
    }
group(Group2)
{
    Visible=false;
    field(Headline3; hdl3Txt)
    {
    }
    field(Headline4; hdl4Txt)
    {
    }
}
```

> **IMPORTANT**
> Unlike other page types, the `group` control has no effect on the UI on pages of type **HeadlinePart**. Its primary purpose is to enable developers to group headlines for controlling visibility.

Dynamic visibility

With dynamic visibility, you can show or hide a headline based on a condition that evaluates to `true` or `false`.

- To dynamically show or hide a headline when the **HeadlinePart** page opens, the headline field must be in `group` control, and you set the `Visible` property on the `group` control to the `Boolean` variable that determines the visibility. For example, you could add code on the page's `OnAfterGetRecord` trigger that evaluates the relevance of displaying `Headline3` and results in a `Boolean` variable being set to `true` or `false`.
- To dynamically show or hide a headline while a page is open, you set the `Visible` property on the `field` control to the `Boolean` variable that determines the visibility.

```
group(Group1)
{
    field(Headline1; hdl1Txt)
    {
    }
    field(Headline2; hdl2Txt)
    {

    }
    }
group(Group2)
{
    // Determines visibility when the page opens
    Visible=ShowHeadline3;
    field(Headline3; hdl3Txt)
    {
        // Determines visibility while the page is open
        Visible=ShowHeadline3;
    }
    field(Headline4; hdl4Txt)
    {
    }
}
```

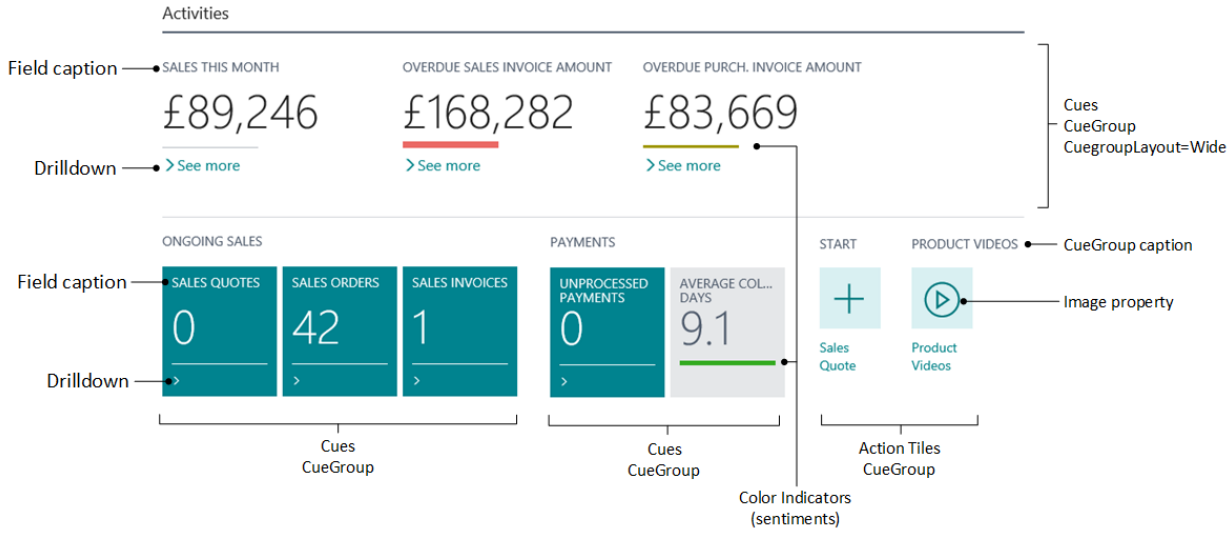
See Also

Pages Overview
Page Object

Creating Cues and Action Tiles on Role Centers

2/17/2021 • 7 minutes to read • Edit Online

This article provides an overview of Cues and Action tiles, and the tasks involved in creating and customizing them for displaying on Role Centers, as illustrated in the following figure:



NOTE

Modifying actions in Cue groups on page extensions is not supported.

Designing Cues

A Cue provides a visual representation of aggregated business data, such as the number of open sales invoices or the total sales for the month. Cues are interactive, meaning that you can select the Cue to drill down to data or open another page, run code, and more. Cues display data that is contained in a table field. This can be raw data or calculated data.

Normal and wide layout

There are two layout options that influence how Cues appear in the client: *normal* and *wide*.

- The *normal* layout displays Cues as tiles. With this layout, Cue groups are automatically arranged to fill in the width of the workspace, which means there can be more than one group horizontally across the workspace.
- The *wide* layout is designed to display large values, such as monetary values. The wide layout gives you a way emphasize a group of Cues. Wide and normal Cue groups can be interleaved. However, wide groups that precede all normal groups will appear in their own section of the workspace, spanning the entire width - providing space for the large values. Wide groups that are placed after normal groups will behave just like the normal layout groups. With this in mind, it is good practice to place Cue groups that use the wide layout, above those that use the normal layout. The wide layout is specified by setting the CuegroupLayout property to `wide`.

> **NOTE**
>
> The wide layout is only supported in the Web client.
>
> The Caption and CaptionML properties of the `cuegroup` control are ignored when the layout is wide.

Supported data types

You can only base Cues on integer and decimal data types. Other data types are not supported and will not display in a Cue.

FlowFields versus normal fields

A Cue can be based on a FlowField or Normal field. If you base the Cue on a FlowField, then you add the logic that calculates the data for the Cue to the CalcFormula property of the FlowField. If you use a Normal field, then you will typically add the logic that calculates the Cue data to an AL trigger or method. Unlike a FlowField, where data is extracted from tables, a Normal field enables you to extract data from other objects such as queries.

Creating a Cue

The implementation of a Cue involves the following elements:

- A table object with a field that holds the data that is contained in the Cue at runtime.
- A page object that contains the table field and displays the Cue in the client.
- Logic that calculates the data to display in the Cue at runtime.

The logic can consist of a combination of AL code and objects, such as tables, queries, and codeunits. How and where you implement the logic will depend on whether the Cue is based on a FlowField or Normal field and what you want to achieve.

> **NOTE**
>
> The examples in this section will set up a Cue that extracts the number of open sales invoices from the **Sales Header** table.

Create a table for Cue data

The first thing that you must do is to create a table that contains fields that will hold the calculated data to display in the Cues at runtime.

1. Create a table object or use an existing one.
2. Add fields for the Cue data.

   For each Cue that you want to display on the page, you must add a **Field** control in the table object. When you add the **Field** control, specify the following properties:

   - Set the Data Type property to **Decimal**, **Integer**, or **Text**, depending on the type of data the Cue will display.
   - Set the FieldClass property to **FlowField** or **Normal**.

     If field is a FlowField, then set the `CalcFormula` property to calculate the Cue data. For more information, see Calculation Formulas and the CalcFormula Property.
3. Add a primary key field for FlowFields.

   A table must have at least one data field. Because a **FlowField** is based on a calculation, it not considered an actual data field. Therefore, if the Cue table only includes FlowFields, you must add "dummy" primary

key field that does not yield any data.

To add primary key, for example, add a field with the name **Primary Key**, and then set its data type to **Code**.

Example

```
table 50100 SalesInvoiceCueTable
{
    DataClassification = ToBeClassified;

    fields
    {
        field(1;PrimaryKey; Code[250])
        {

            DataClassification = ToBeClassified;
        }
        field(2; SalesInvoicesOpen ; Integer)
        {
            FieldClass = FlowField;
            CalcFormula = count("Sales Header" where("Document Type"=Filter(Invoice), Status=FILTER(Open)));

        }
    }

    keys
    {
        key(PK; PrimaryKey)
        {
            Clustered = true;
        }
    }
}
```

Add Cues to a Page object

After you have a table for holding the Cue data, you create a page that you associate the table, and then add Cue fields on the page. Typically, you will create Card Part type page that will be part of the Role Center page. Cues are arranged into one or more groups on the page. Each group will have its own caption.

1. Create a page object that has the SourceTable property set to the Cue data table.
2. Add a `cuegroup` control.
3. Under the `cuegroup` control, for each Cue that you want to display, add a `field` control.
4. If you want to set the `cuegroup` to use the wide layout, set the `CuegroupLayout` property to `wide`.

   Repeat steps 2-4 to add additional Cue groups.

5. Initialize the Cue fields.

   You must initialize the Cue fields on the page. To do this, for example, you can add the following AL code to the OnOpenPage Trigger.

```
      RESET;
    if not get then begin
        INIT;
        INSERT;
    end;
```

Example

```
page 50105 SalesInvoiceCuePage
{
    PageType = CardPart;
    SourceTable = SalesInvoiceCueTable;

    layout
    {
        area(content)
        {
            cuegroup(SalesCueContainer)
            {
                Caption='Sales Invoices';
                // CuegroupLayout=Wide;
                field(SalesCue; SalesInvoicesOpen)
                {
                    Caption='Open';
                    DrillDownPageId="Sales Invoice List";
                }
            }
        }
    }

    trigger OnOpenPage();
    begin
        RESET;
        if not get then begin
            INIT;
            INSERT;
        end;
    end;
}
```

Designing Action tiles

Action tiles promote an action or operation to the user on the Role Center. Action tiles act as links that perform a task or operation, like opening another page, starting a video, targeting an another resource or URL, or running code. They will arrange on the workspace just like that use the normal layout.

Similar to Cues, Actions tile can be grouped together, under a common caption, by using the `cuegroup` control. The difference is that instead adding field controls under the `cuegroup` control, you create Action tiles by adding actions to the `cuegroup` control.

Create an Action tile

1. Develop or locate the functionality that you want to Action tile to perform.

For example, create the page object that you want the Action tile to open, add AL code that you want the Action tile to run, find the URL to the video.

2. Open the page on which you want to display the Action tiles. For example, this could be the page that you created in the previous task.

3. In the location where you want the Action group, add a `cuegroup` control.

4. Configure the control to the desired operation.

   For example, if it should open a page, set the control's RunObject property to the appropriate page. Or, set it to call a function or method.

Example

The following code adds an Action tile that opens **Sales Invoice** page.

```
cuegroup(SalesActionontainer)
{
    Caption='New Sales Invoice';

    actions
    {

        action(ActionName)
        {
            RunObject=page "Sales Invoice";
            Image=TileNew;

            trigger OnAction()
            begin

            end;
        }
    }
}
```

Styling an Action tile

You can use the Image property on an `action` control to change the look of the Action tile. For Action tiles, the `Image` property supports several standard values that start with the text `Tile`, such as `TileNew` and `TileYellow`. These values change the Action's background color and icon as follows:

- A value that has the format `Tile[color]` will set the Action tile to use the circle icon and a background that is specified by `[color]`. For example, `TileBlue` will display a circle icon in a blue background.
- A value that has the format `Tile[picture]` will set the Action tile to use an icon that is specified by `[picture]` and a neutral background color. For example, `TileCamera` will display a camera icon on the neutral background.

> **NOTE**
> If you use a value that is not valid or recognized, the Action tile will default to display the circle icon on the neutral background.

See Also

FlowFields
Page Object
Pages Overview
Table Object

Page Types and Layouts

2/17/2021 • 14 minutes to read • Edit Online

Understanding how Business Central displays a page dependent on its *page type* is important to be able to create a good user experience. There are also several page properties and variations of the page structures that can help create an intuitive and efficient user interface.

In the following we are focusing on how pages appear when a user accesses Business Central from a desktop browser. But it is an important point that the same page types apply across the different form factors of apps, and that the Business Central page type layouts automatically adapt to work well on different devices, e.g. on a phone or a tablet.

Understanding page types

Choosing the right page type is the first step when creating or modifying a page. The next step is to organize the page contents to suit its purpose in your solution. We recommend that you design pages based on the user tasks that you want to support.

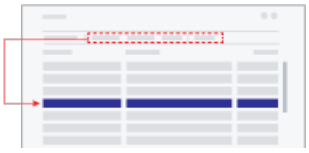
The following table provides an overview of the page types supported in Business Central, their typical uses, and basic characteristics. To specify the page type, use the `PageType` property. For more information, see PageType Property.

PAGE TYPE	EXAMPLES OF USE	MAIN DATA DISPLAY	CHARACTERISTICS
`RoleCenter`	Overview of business performance and the start page for a specific user profile.	Defined by the embedded parts.	A collection of parts (Cues, KPIs, etc.) and the contents of the navigation pane.
`Card`	Master, reference, and set up data management. Card page example	Single entity	Titled entity with FastTabs. May embed parts.
`Document`	Transaction and other document management.	Single entity	Titled entity with FastTabs. Should have the document lines ListPart immediately follow the header section(s).
`ListPlus`	Statistics, details, and related data management.	Single entity	Titled entity with at least one `ListPart`. Can have fields above or below the part(s).
`List`	Entity overviews and navigation, and inline editing of simple entities. List page example	Collection of entities/entries	A single list with a caption. May have field groups and subpages above and below the list's `Repeater`.
`Worksheet`	Line-based data entry tasks (such as journals) and inquiries.	Collection of entities	A single list or table with a caption. May have field groups and subpages above and below the worksheet's `Repeater`.

PAGE TYPE	EXAMPLES OF USE	MAIN DATA DISPLAY	CHARACTERISTICS
`StandardDialog`	Routine dialog that starts or progresses a task.	Single or collection	A cancelable dialog with an instruction to the user. Can have one or more groups of fields, a list, and parts.
`ConfirmationDialog`	Confirmative or exceptional dialog, such as warnings.	Single or collection	A Yes/No dialog with an instruction to the user. Can have one or more groups of fields, a list, and parts.
`NavigatePage`	Multi-step dialog (also known as a "Wizard").	Single or collection	Can have one or more groups of fields, a list, and parts.
`CardPart`	A page that is embedded in another page, such as in a FactBox.	Single entity	Single group of fields representing fields in a FastTab.
`ListPart`	A page that is embedded in another page, such as in a FactBox.	Collection of entities/entries	Single `Repeater` representing columns in a list or table. Can have fields above or below the repeater.
`HeadlinePart`	A page that is embedded in a `RoleCenter` page to display relevant insights from across the business.	Single entity	Single group of fields representing headlines.

The two principal categories of page types

A fundamental characteristic of a page type is how it relates to the data presented on the page. Two principal ways exist: *entity*-oriented (typical for the `Card` page type) and *collection*-oriented (typical for the `List` page type.)

`CARD`, `DOCUMENT`, AND `LISTPLUS`	`LIST` AND `WORKSHEET`
	
The **entity-oriented** page types have actions (in top and in action bar) that affect the entity or context given by the title of the page.	The **collection-oriented** page types provide actions in action bar (and on the rows' action menu) that take effect on the selected row(s) in the collection.

Entity-oriented pages

In Business Central, entity-oriented pages are used to support users when their tasks revolve around a single business entity. The most typical entity-oriented page is the `Card`, which provides details about a single customer or other master data, and the `Document`, which represents a single transaction, or other important business event, e.g. a sales transaction.

`ListPlus` is also an entity-oriented page type. Unlike `Card` and `Document` pages, the `ListPlus` page type is for pages that have a prominent `ListPart` and either few or no header fields.

The `CardPart` page type is an entity-oriented page type for inclusion in another page, for example, in a FactBox.

> **NOTE**
>
> Since entity-oriented pages represent a *single* entity, such as a customer or an item, we recommend that you do not use a `Repeater` group in the construction of entity-oriented pages. If you do, some of the repeater's features may not work properly, and it may not get the expected size. However, an entity-oriented page *can* embed a list part page that, in turn, contains a repeater control.

Collection-oriented pages

In Business Central, collection-oriented pages are used to support users when their tasks involve multiple entities or records at the same time. The most typical collection-oriented page type is the `List` (for example showing customers, items, and so on.) from which the user can seek out the entities to work with.

The `Worksheet` is the other prominent collection-oriented page type, suited for data entry (for example, in journal pages) and other tasks related to managing a set of entities/entries based on custom fields above and/or below the collection.

The `ListPart` page type is a collection-oriented page type for inclusion in another page, for example, in a FactBox.

Dialog pages

The page types in Business Central that are available exclusively for displaying dialogs, such as the `StandardDialog` and `ConfirmationDialog` page types, can represent an entity *or* a collection. The title caption and actions are suited for both types.

Composing pages

Within a page, the developer can combine page fields into groups. This can help the user overview the page by placing related fields together. And within a group, *subgroups* can further increase the structure of the data displayed in a page.

Besides adding fields and groups to a page, it is possible to embed another page of type `CardPart` or `ListPart`. (These two page types can in turn not embed other pages.)

When pages are created that embed parts, Business Central divides the available screen real estate between the page's groups of fields and any embedded pages. Screen space is divided between field groups and embedded pages such that the user can get access to the full contents of the page and collapse/expand specific sections of interest.

How space allocation takes place for a given page depends on the chosen page type, the structure of page contents (field groups and page parts), and on the size of the browser window.

A page is Content + Actions + FactBoxes

For all pages (excluding `RoleCenter`, dialogs, and part pages) there is a common structure to the areas of a page where content, FactBoxes, and actions can be displayed.

The `content` area provides rich layout capabilities, which are described in the coming sections. The `FactBoxes` area is limited to showing a list of parts, usually in a vertical arrangement. The header consists of the title, action bar, and controls for filtering, views, and so on.

For more information about page areas, see Pages Overview.

Field groups and page parts

In the following sections you find descriptions of typical page layouts, recommendations for how to organize the contents, and illustrations of the principles by which the sections of page share screen real estate. The types of content on a page are illustrated this way:

SYMBOL	SECTION	NOTES
	Group of fields, or CardPart	Appears on the page as a FastTab, with fields wrapped in one or more columns.
	ListPart	List or table layout, with integrated action bar.
	Repeater	List or table layout.

Sizing of page sections

The groups of fields and page parts making up a page are rendered when the user opens the page.

Dependent on the size of the available screen real estate, for example, in the browser window on a desktop computer, Business Central sizes the sections automatically to make the most of the space.

There are three different ways a section's size are determined.

SYMBOL	LAYOUT BEHAVIOR	NOTES
	Size to content	Enclosing page will use a scrollbar if needed.

SYMBOL	LAYOUT BEHAVIOR	NOTES
	Size to content within certain limits	The part will use a scrollbar if content exceeds available space.
	Size to fill space	The part will use a scrollbar if content exceeds available space.

Which of the section sizing behavior is used is dependent on the chosen page type. For each of the page types described in the sections below, we present the typical layouts, and the way that the sections are sized.

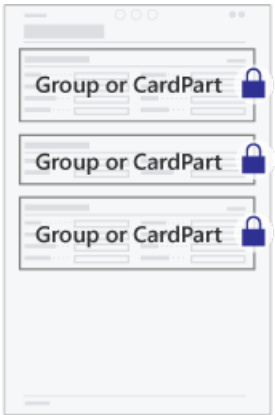
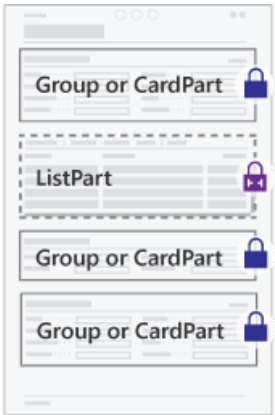
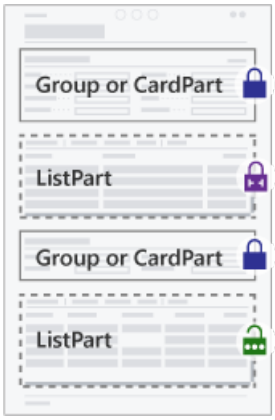
Card and Document page layouts

The primary purpose of Card pages is to support users managing master and reference data, such as Customer, Vendor, and Item entities. (The name *Card* refers to how this kind of business data was kept on paper cards in filing cabinets before being computerized.) The Card page type is also often used for setup pages.

The Document pages' primary purpose is to represent a transaction or other important event in the domain of business. Document pages are the computerized counterpart to paper-based documents (quotes, invoices, orders, etc.), and as such, document pages often have associated workflow or audit trail requirements.

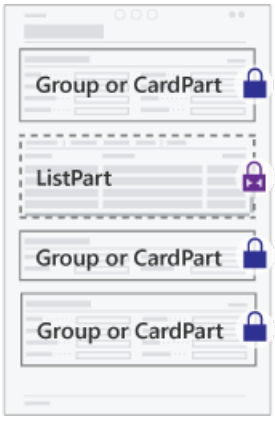
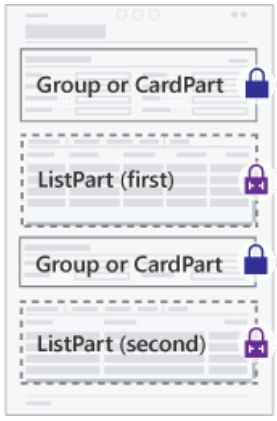
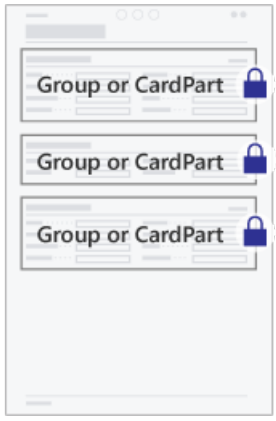
Below are examples of Card and Document page compositions, showing how space is divided. Parts can be combined in more ways than shown here to suit different scenarios.

Card layouts

EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
Sections are placed vertically from top to bottom of the page.	A ListPart can be embedded. In this case, the ListPart's height is limited.	When a ListPart is embedded as the last part on the page, it will expand to fill space.
		

Document layouts

EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
Sections are placed vertically from top to bottom of the page. The lines ListPart comes after the header section(s).	Multiple ListParts can be embedded. In this case, the first ListPart is allowed the most space.	When no ListPart is embedded, the Document layout follows the Card layout exactly.

EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
		

> **NOTE**
>
> The Document page type allows the first ListPart on the page to use additional vertical space before showing a scrollbar. This allows more space for showing the document lines without requiring the user to scroll.

The well-designed card or document page

From the user's perspective, the following are qualities of a well-designed card or document page:

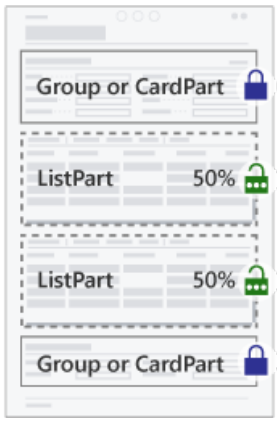
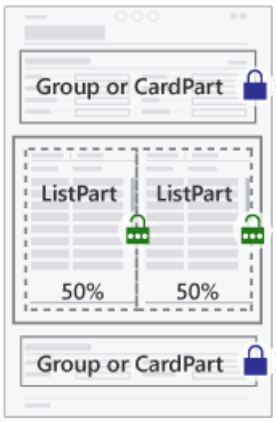
- Uses the page type `Card` if the page represents master or reference data, or is a setup page.
- Uses the page type `Document` if the page represents a transaction or other important event in the domain of business.
- Has a page title that clearly identifies the data represented in the page.
- Is optimized for overview by organizing data in FastTabs and marking relevant fields as `Promoted` or `Additional`.
- Favors header fields and other important fields by placing them in a FastTab titled General that come first on the page.
- Has one or two FactBoxes to give relevant statistics and quick access to related documents.
- For Document pages, a FastTab titled *Lines* comes second on the page with the document lines.

ListPlus page layouts

The ListPlus pages' primary purpose is to support users in managing or browsing a collection of data, or entries, related to a specific business entity or event. For example, the *Customer Sales* page is a ListPlus page that shows sales numbers for a customer and providing dedicated viewing options for sales analysis.

The ListPlus page type is a versatile means to support analysis and management tasks in a specific entity context (named by the page title). ListPlus pages can show persistent data about the entity/event in addition to giving options for how data is viewed or filtered.

A ListPlus page should generally not contain a repeater control but will typically embed a `ListPart` page that in turn embeds a repeater. In addition, a ListPlus page can embed groups of fields and cardparts. Below are examples of ListPlus page compositions, showing how space is divided. Parts can be combined in more ways than shown here to suit different scenarios.

EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
Page sections are placed from top to bottom. The first ListPart fills vertical space.	When placing two or more ListParts, they'll share available vertical space.	When placing two ListParts in a group, they share horizontal space.
		

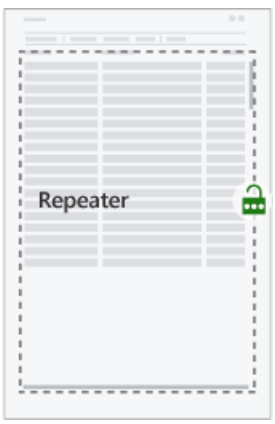
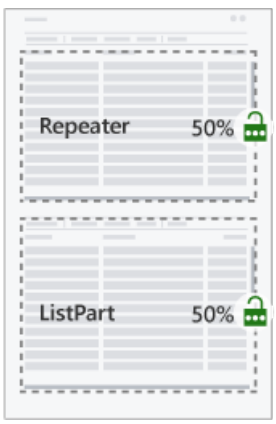
From the user's perspective, the following are qualities of a well-designed ListPlus page:

- Has a page title that clearly identifies the context for the information presented in the page.
- Is optimized for showing one set of details, and giving the user means to work with them.
- Presents information in the page in such a way that the hierarchy can be understood when read from top to bottom.
- If present, fields that control which data is presented in another FastTab come before that FastTab.
- If present, fields that show data dependent on the chosen row (in the ListPart) come after the ListPart.

List page layouts

List pages support users viewing and finding specific entities or entries in a collection. Lists that are editable have cells available for data entry and update.

List pages must contain a single `Repeater` group. In addition, a list can embed groups of fields, card parts, and list parts. Below are examples of list page compositions, showing how space is divided. Parts can be combined in more ways than shown here to suit different scenarios.

EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
The repeater control assumes full vertical space.	When a field group or cardpart is embedded, space for repeater is reduced.	When a listpart is embedded, space is shared equally between part and repeater.
		

From the user's perspective, the following are qualities of a well-designed List page:

- Defines a set of columns that is optimized for viewing and filtering the given collection. Optimize the column order for data entry if the list is editable.
- Has a page title that clearly names or identifies the collection of entities/entries presented.
- If a summary or additional detail related to the selected row are shown, these appear below the list.
- If custom viewing options are available, these appear above the list.
- Has one or two FactBoxes to give essential collection statistics, and relevant related details for the selected row.

Worksheet page layouts

A Worksheet page lets users view and manage a collection of entries in tabular or matrix form. It is well suited for cases when a custom filter or a set of default field values is the basis for users' understanding of the collection, such as with journals (where the user selects a batch) and inquiry pages (where the user forms a query).

Use a Worksheet page (instead of a List page) when there are header fields that determine or subdivide a collection, i.e. the user must make a choice in the header field before the repeater has data to show at all. Or when there are header fields that represent default values for data entry in the repeater.

> **NOTE**
> The Worksheet page type doesn't support the same part and group compositions as the List page type.

Worksheet pages must contain a single `Repeater` group. In addition, a worksheet can embed groups of fields, cardparts, and listparts. Below are examples of list page compositions, showing how space is divided. Parts can be combined in more ways than shown here to suit different scenarios.

EXAMPLE 1	EXAMPLE 2	EXAMPLE 3
The repeater control takes full vertical space, but leaving space for a group or CardPart above.	A group, CardPart, or ListPart can be embedded below the repeater that then assumes the remaining vertical space.	Groups and/or parts are embedded above and below, leaving the remaining vertical space for the repeater.
		

From the user's perspective, the following are qualities of a well-designed worksheet page:

- Defines a set of columns that is optimized for overviewing and managing the given collection. Columns are ordered relative to their importance.
- Has fields above the grid that specify filtering options or specify the default values effective during data entry

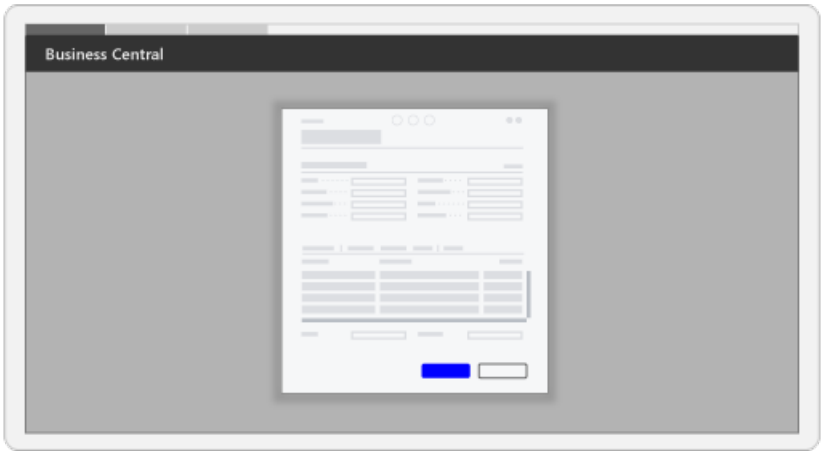
and editing in the grid.

- If summary fields or additional details of the selected row are shown, these appear below the repeater.

Dialog page layouts

Some page types in Business Central are available exclusively for displaying dialogs, such as the `StandardDialog` and `ConfirmationDialog` page types. In addition, there are programmatic ways in AL to display a dialog to the user with the Dialog data type, as well as dialogs defined as report request pages.

It is also possible to use the common page types (`Card` , `Document` , `List` , etc.) to present a dialog to the user. In this case, pages are created, composed, and can embed the same elements, as when displayed ordinarily. Presenting the page to the user as a dialog requires certain AL code that activates the dialog mode. When this happens, the page is shown with dismiss buttons in the page footer.



Generally, Business Central displays dialogs on the screen in a frame that is more narrow and not taking up full vertical height, compared to how pages appear ordinarily. Aside from that, pages lay out their contents according to the same principles, whether displayed as a dialog or not.

Given the size of the screen where the dialog appears, more or less of the page contents will be visible without scrolling. When a page contains a lot of content, it is possible for the user to increase the dialog size with the maximize button.

> **NOTE**
> The dialogs created from the ConfirmationDialog and StandardDialog page types are not currently providing a maximize button.

See Also

Page, Page Fields, and Page Extension Properties
PageType Property
Actions Overview
Using Designer
Adding a FactBox to a Page
Designing Role Centers

Designing List Pages

2/17/2021 • 8 minutes to read • Edit Online

The *List* page type displays records from an underlying table, either as rows and columns or as individual tiles.

You design list pages when you want to provide users with a collection of data, enabling them to get an overview of and find entities to work with, such as customers, vendors, or sales orders. Typically, a list page will link to an associated card page that lets users view or modify specific entities in the list.

There are different ways to incorporate a list page into that application:

- Make the list page available from the navigation of a Role Center page.

  This gives users quick access to the page. With this implementation, the list page opens in the content area of the Role Center page, where the Role Center's navigation area is still present and accessible at the top of the page. For more information about Role Centers, see Designing Role Centers.

- Make the list page available from an action on another page.

  With this implementation, the list page opens in a separate window in front of the current page.

- Make the list page searchable from the **Tell me what you want to do** feature.

  With this implementation, the list page also opens in a separate window. For more information, see Adding Pages and Reports to Search.

Customizing a list pages from the client

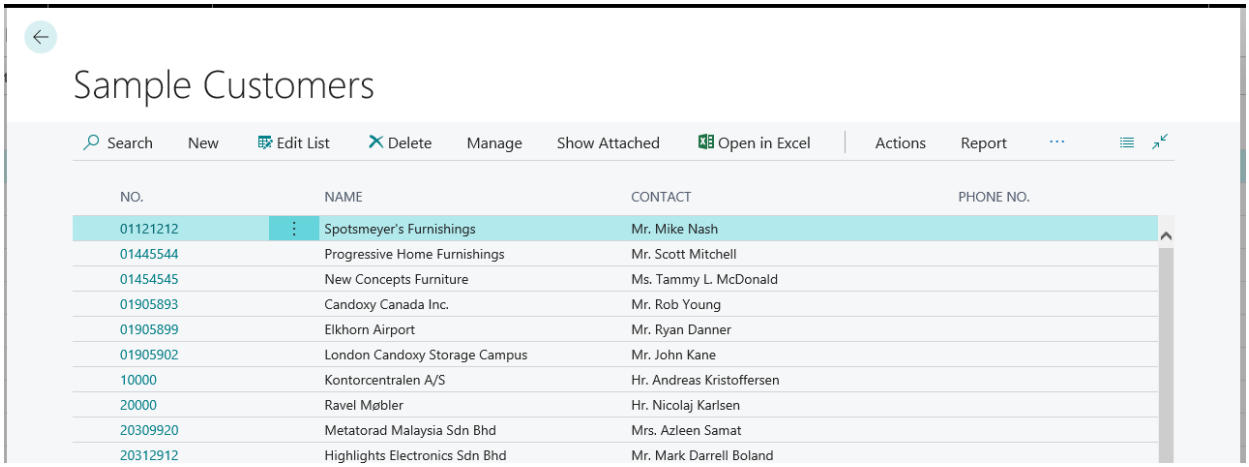
In the client, users can personalize list pages by rearranging or hiding records or FactBoxes as they like. For more information, see Personalizing Your Workspace.

As a developer or administrator, you can use Designer to customize the list page the same way that individual users personalize their own work spaces. The difference is that changes you make are applied to all users. For more information, see Using Designer.

Simple List Page Code Example

2/17/2021 • 2 minutes to read • Edit Online

The AL code in this article creates a simple list page that displays records from an existing table.



For a more detailed explanation of the list page, see Designing List Pages.

```
page 50111 SampleCustomerList
{
    PageType = List;
    ApplicationArea = All;

    // Specifies the page to display records from the Customer table.
    SourceTable = Customer;

    // Makes the page searchable from the Tell me what you want to do feature.
    UsageCategory = Lists;

    // Specifies the card page Sample Customers to be uses for modifying or creating new customer records.
    CardPageId = 50112;

    // Sets the title of the page to Sample Customers.
    Caption = 'Sample Customers';

    layout
    {
        area(Content)
        {
            // Sets the No., Name, Contact, and Phone No. fields in the Customer table to be displayed as
columns in the list.
            repeater(Group)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;

                }
                field(Name; Name)
                {
                    ApplicationArea = All;

                }
                field(Contact; Contact)
                {
                    ApplicationArea = All;
                }
```

```
                field(Phone; "Phone No.")
                {
                    ApplicationArea = All;

                }
            }
        }

    }

    actions
    {
        // Adds an action on the Actions menu of the action bar that opens the page Customer Ledger Entries.
        area(Processing)
        {
            action("Ledger Entries")
            {
                ApplicationArea = All;
                RunObject = page "Customer Ledger Entries";
                trigger OnAction();
                begin

                end;
            }
        }

        // Promotes an action for creating a sales quote to promoted action menu called New.
        area(Creation)
        {
            action("New Sales Quote")
            {
                ApplicationArea = All;
                RunObject = page "Sales Quote";
                Promoted = true;
                PromotedCategory = New;
                Image = NewSalesQuote;
                trigger OnAction();
                begin

                end;
            }
        }

        // Adds an action on the Report menu that opens the Top 10 List report.
        area(Reporting)
        {
            action("Top 10 List")
            {
                ApplicationArea = All;
                RunObject = report "Customer - Top 10 List";
                trigger OnAction();
                begin

                end;
            }
        }
    }
}
```

See Also

AL Development Environment
Page Extension Object

Working with Repeater Controls

2/17/2021 • 3 minutes to read • Edit Online

A repeater is a control used to define a list of records from the source table of a page. These are displayed as rows and columns, where each row is a record and each column is a field.

You add a `repeater()` control within the `area(Content)` control of a page, and then you nest a `field()` control for each of the fields from the table specified in the SourceTable Property that you want to include. The order of the field controls determines the order in which they appear on the page.

A list is a characteristic element of the layout of pages of the type **List** and **ListPart**. You define the type of a page using the PageType Property. List pages are designed for using a single `repeater()` control, which must be defined at the beginning of the content area. If you include more than one repeater or another control like a group or grid, the page might not behave as expected. If you want to design a page that includes controls in the content area other than a repeater, then try using a **Worksheet** page type instead. For more information, see Pages Types and Layout.

Pages of the type **API** are also designed to integrate a repeater control, but they cannot be displayed in the user interface.

The following figure illustrates how a list created by a repeater is displayed in the Web Client.

NO.	NAME	CONTACT	PHONE NO.
01121212	Spotsmeyer's Furnishings	Mr. Mike Nash	
01445544	Progressive Home Furnishings	Mr. Scott Mitchell	
01454545	New Concepts Furniture	Ms. Tammy L. McDonald	
01905893	Candoxy Canada Inc.	Mr. Rob Young	
01905899	Elkhorn Airport	Mr. Ryan Danner	
01905902	London Candoxy Storage Campus	Mr. John Kane	
10000	Kontorcentralen A/S	Hr. Andreas Kristoffersen	
20000	Ravel Møbler	Hr. Nicolaj Karlsen	
20309920	Metatorad Malaysia Sdn Bhd	Mrs. Azleen Samat	
20312912	Highlights Electronics Sdn Bhd	Mr. Mark Darrell Boland	

NOTE

The Web client does not support displaying Repeater controls that contain other Parts or FlowFilter fields.

How to customize a list

You can make various changes to customize the appearance of the list in a page and determine which data is displayed. This can be done using AL code or directly from the Web Client.

To the improve the visibility and readability of the list you can set some properties at control and field level. For example, you can specify the width of a column using the Width Property or set row indentation using the IndentationColumn Property and the IndentationControls Property.

You should also consider limiting the number of columns and enabling a freeze column, by setting the FreezeColumn Property, as well as applying filters to display only the most relevant data and minimize scrolling. This is particularly important when it comes to mobile devices, where there are restrictions as to how much content can be shown. You can also do this by defining a new view on a page. For more information, see Views.

Alternatively, you can add, move or hide fields and make adjustments like modifying the column width, directly from the user interface. To apply these changes you use the Designer tool. For more information, see Using

Designer. If these changes are meant for your own workspace, then you use the Personalize tool. For more information, see Personalizing Your Workspace. To sort the data and apply filters you can use the Filter Pane from the page. For more information, see Sorting, Searching, and Filtering.

You can also use the Tile View to display records as tiles (or bricks) instead of as rows and optimize the space and readability of data. For more information on how to customize the tile view, see Displaying Data as Tiles.

Example

The following example shows how the repeater control is used to define a list page for the Customer table.

```
page 50111 SampleCustomerList
{
    PageType = List;
    ApplicationArea = All;
    SourceTable = Customer;
    UsageCategory = Lists;
    CardPageId = 50112;
    Caption = 'Sample Customers';

    layout
    {
        area(Content)
        {
            // Sets the No., Name, Contact, and Phone No. fields in the Customer table to be displayed as
columns in the list.
            repeater(Group)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;

                }
                field(Name; Name)
                {
                    ApplicationArea = All;

                }
                field(Contact; Contact)
                {
                    ApplicationArea = All;
                }

                field(Phone; "Phone No.")
                {
                    ApplicationArea = All;

                }
            }
        }
    }

    actions
    {
        area(Processing)
        {
            action("Ledger Entries")
            {
                ApplicationArea = All;
                RunObject = page "Customer Ledger Entries";
                trigger OnAction();
                begin

                end;
            }
        }
    }

}
```

You can use page customizations to modify the layout of a page for concrete users. The following code hides the `"Phone No."` column from the users with `Accountant` profile.

```
profile Accountant
{
    Description = 'Functionality for finance staff performing any AR or AP work and managerial reporting.';
    RoleCenter = "Accountant Role Center";
    Customizations = MyCustomization;
}

pagecustomization MyCustomization customizes "Customer List"
{
    layout
    {
        modify("Phone No.")
        {
            Visible = false;
        }
    }
}
```

See Also

Pages Overview
Designing List Pages
Page Customization Object

Displaying Data as Tiles for Lists

2/17/2021 • 3 minutes to read • Edit Online

In the client, on `List` type pages, users can view the page in the tile view. The tile view shows records as tiles (or bricks) instead of as rows. Tiles optimize space and readability of data. Tiles are especially useful for images, like on a page that show items, customers, and contacts. Business Central also provides Microsoft Teams integration, which enables users to display an interactive, visual representation of records in a Teams chat. These features both use the same design concept, which is based defining a `Brick` field group on the an entities source table.

By default, the tile view will display the first five fields that are defined in page's `repeater` control. This article describes how you can customize the tile view for list type pages.

> **NOTE**
> The `Brick` field group is also used to define the fields that appear when a record is shown in a Microsoft Teams conversation. For more information, see Extending Teams Cards.

Tile view in the client

To switch between the list and tile view, users selecting the **View layout options** icon in the action bar of the page. If tiles contain a media field type, then there are two tile view options: **Tiles** and **Tall Tiles**. The same information is displayed except with **Tall Tiles**, images are larger and display at the top of the tiles.

Tiles are interactive. A context menu is available in the upper right corner. The context menu contains the actions that are defined for the record, like in the list view. To drill down to a card page for a record, the user selects the tile.

Customizing the tile view in AL

You specify the data that you want shown in the tile view in the source table of the page by adding a `Field Group` that has the name `Brick`:

```
fieldgroups
{
    fieldgroup(Brick; <field 1>, <field 2>, <field 3>, <field 4>, <field 5>)
    {

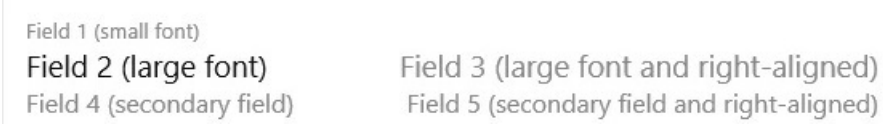
    }
}
```

There's no limit on the number fields that you can display in a tile. However, we recommend that you limit tiles to five data fields and one image field.

> **IMPORTANT**
> By default, the `Field Group` named `DropDown` is interpreted as `Brick` when a `Brick` definition has not been set. The `DropDown` is typically set on entities such as customer, vendor, and items. For more information, see Field Groups (Drop-Down Controls).

Field layout in tiles

The order of the fields determines how they appear in the layout of the tile, no matter the order in the page object. Depending on the number of columns that you define in the `Field Group`, the layout will dynamically change. This concept is illustrated in the following figure:



The fields 2 and 3 are shown in a large font. These fields should typically contain data that identifies the brick. For example, in the **Customers** list, the **Customer Name** and **Balance** are displayed in fields 2 and 3.

Including images in tiles

To display an image in the brick, you include a `Media` data type field in the `Field Group` definition. You don't have to include a field control for the media field in the page object, because the image will be shown in the tile view automatically.

The image will be displayed on the left side of the tile (or at the top in the **Tall Tiles** view), whatever its position in the `Field Group` definition. If an image doesn't exist for a certain record, a default picture is displayed instead.

For information including media on records, see Working With Media on Records.

Styling text in tiles

Like the list view, the tile view supports the Style Property and StyleExpr Property. You apply these properties on the page field controls. These properties, for example, let you mark numbers as favorable or unfavorable.

Example

The following code is a simple example of a table that includes `Field Group` control for displaying data in the tile view of a list page.

```
Table 50100 MyTable
{

    fields
    {
        field(1; Number; Integer)
        {
        }

        field(2; Description; Text[50])
        {
        }
        field(3; Inventory; Integer)
        {
        }
        field(4; Image; Media)
        {
        }
    }

    keys
    {
        key(PK; Number)
        {
        }
    }

    fieldgroups
    {
        fieldgroup(Brick; Number, Description, Inventory, Image)
        {
```

```
        }
    }
}

page 50100 MyListPage
{
    PageType = List;
    ApplicationArea = All;
    UsageCategory = Lists;
    SourceTable = BrickTableTest;
    Editable = true;
    CardPageId = MyCardPage;

    layout
    {
        area(Content)
        {
            repeater(GroupName)
            {
                field(Number; Number)
                {
                    ApplicationArea = All;
                }
                field(Description; Description)
                {
                    ApplicationArea = All;
                }
                field(Inventory; Inventory)
                {
                    ApplicationArea = All;
                    Style = Attention;
                }
            }
        }
    }
}
```

See Also

Designing List Pages
Working With Media on Records

Views

2/17/2021 • 3 minutes to read • Edit Online

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. Views are defined on page extension objects to provide an alternative view of data and/or layout on an existing page, and in views on page customization objects, they can be used to provide an alternative view for a certain profile.

A view offers:

- Filtering on multiple table fields on the source table defined for the page.
- Sorting of the data on multiple table fields, but only in one direction; either *ascending* or *descending*.
- Layout changes, modifying page columns, moving them, etc.

Views are defined directly in code, on the list page that they modify. The defined view or views are available to the user through **Filter Pane** on a page and appear in the sequence that they are defined in code.

> **NOTE**
> `showMyCode` does not apply to views. Views defined in an extension with `showMyCode` set to `false` can still be copied using Designer.

Snippet support

Typing the shortcut `tview` will create the basic layout for a view when using the AL Language extension in Visual Studio Code.

Filtering and sorting

You can filter on the data in a view by using the `Filters` property. The following is an example of the syntax:

```
Filters = where ("Balance (LCY)" = filter (> 500), Name = filter ('G*'));
```

For more information, see Filters.

You can sort on the data in a view by using the `OrderBy` property. The following is an example of the syntax:

```
OrderBy = ascending ("Balance (LCY)", Name);
```

For more information, see OrderBy.

> **NOTE**
> All filters are applied to the table field(s), not the page field(s), which allows filtering on a table field not shown on the page.

View example

The following example shows a page customization of the **Customer List** page, which is available for a specific

role center only; the **My Role Center**. Change the role center view under **My Settings**. The definition of the view adds a caption which is displayed on the left side in the UI. The view sorts the customer balance in ascending mode and the view modifies the layout by moving the customer balance first and adding a freeze column after it.

> **IMPORTANT**
> The definition of the `view` section must come after any definition of layout and actions, otherwise you will get a compilation error.

```
profile MyProfile
{
    Description = 'My Role Center';
    RoleCenter = "Order Processor Role Center";
    Customizations = MyCustomization;
}

pagecustomization MyCustomization customizes "Customer List"
{
    layout
    {
        // Change the layout of the page
    }

    actions
    {
        // Add any actions to the page
    }

    views
    {
        addfirst
        {
            view(BalanceLCY)
            {
                Caption = 'Ordered Balance LCY';
                OrderBy = ascending ("Balance (LCY)");

                layout
                {
                    // Change the layout of the view

                    movefirst(Control1; "Balance (LCY)")

                    modify(Control1)
                    {
                        FreezeColumn = "Balance (LCY)";

                    }
                }
            }
        }
    }
}
```

Limitations

In general, views can in several ways be compared to page customizations. These are the limitations of views:

- For views you can modify the same control properties as for page customization objects independently of where the view has been defined (page, page extension, or page customization level). This is validated by the

compiler.

- It is not possible to use variables or methods in a view. When writing client-side expressions for properties like **Visibility**, it will only be possible to use constant values or table field references. This is validated by the compiler.
- It is not possible to create new controls for a page from a view.

See Also

AL Development Environment
Developing Extensions
Page Object
Page Extension Object
Page Customization Object
SharedLayout Property

Adding Custom Filter Tokens

2/17/2021 • 3 minutes to read • Edit Online

In the client, when filtering lists using the filter pane, users can enter filter tokens, which are special words that resolve to one or more values. This powerful feature makes filtering easier by reducing the need to navigate to other pages to look up values to enter as filter criteria.

There are several useful filter tokens available in Business Central. For example, entering **%mycustomers** in a **Customer No.** field will resolve to the set of customers in the user's **My Customers** list such as `1001|1002`, making it easy to find relevant sales orders for customers 1001 and 1002.

You can add custom filter tokens and make these available in any language and across the application. To add your custom filter token, you need to define the token word that users will enter as filter criteria, and define a handler that resolves the token to a concrete value at runtime. For more information, see Filter Tokens in our ALAppExtensions repository on GitHub.

Defining the token word and the handler

To create the token word, start by defining a multi-language text string for your word. Subscribe to the `OnResolveTextFilterToken` event associated with the `MakeTextFilter` method from the `Filter Tokens` codeunit. In the event subscriber, if the value of the `TextFilter` parameter contains the token string, process its value and construct the final filter string. If the filter string must contain multiple values, you must handle the operators that join them by adding the `|` filter symbol (OR operation). Complete the operation by setting the value of the `TextFilter` parameter to the value of the final filter string.

> **TIP**
> Filter criteria will often contain symbols along with filter tokens. We recommend that you only modify the filter token you have introduced and preserve the rest of the filter string.

Example

This example shows how you can use the guidelines above to create the **%MYTOKEN** filter token. This will return a filter with the accounts marked as favorite by the user.

> **NOTE**
> To keep this sample short and simple, the entire filter string is overwritten.

```
codeunit 50101 MyAccountFilterTokenSimple
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Filter Tokens", 'OnResolveTextFilterToken', '', true,
true)]
    local procedure FilterMyAccounts(TextToken: Text; var TextFilter: Text; var Handled: Boolean)
    var
        MyAccount: Record "My Account";
        MaxCount: Integer;
    begin
        if StrLen(TextToken) < 3 then
            exit;

        if StrPos(UpperCase('MYTOKEN'), UpperCase(TextToken)) = 0 then
            exit;

       Handled := true;

        MaxCount := 20;
        MyAccount.SetRange("User ID", UserId());

        if MyAccount.FindSet() then begin
            MaxCount -= 1;
            TextFilter := MyAccount."Account No.";

            if MyAccount.Next() <> 0 then
                repeat
                    MaxCount -= 1;
                    TextFilter += '|' + MyAccount."Account No.";
                until (MyAccount.Next() = 0) or (MaxCount <= 0);
        end;
    end;

}
```

To try it out in the client, open the `Charts of Accounts` page, filter on **No.** field, and type in a substring that starts the same way with the chosen token word, like **%MYTO**.

Design considerations

Resolving tokens is intended to be fast, simple, and reliable. When implementing event subscribers to resolve filter tokens, keep in mind that these events can be triggered from any user task in Business Central, and in some cases may be triggered repeatedly such as when searching across columns. To improve usability and reduce the impact on performance, do consider the following practices:

- Avoid implementing tokens that are only relevant to few business tasks, or assume they are used in the context of a specific page.
- Avoid implementing tokens that are time-consuming to resolve. Examples of this include looking up records in large or poorly indexed tables, or fetching data from a remote service.
- Avoid implementing tokens that are complex or unreliable and may result in error.
- Avoid displaying pages, dialogs or any other form of interactive UI.

See Also

Sorting, Searching and Filtering Lists

Designing Indented Hierarchy Lists

2/17/2021 • 4 minutes to read • Edit Online

This article explains how to indent specific rows in a list.

Overview

Using the indentation properties in AL, you can display rows in a parent-child structure.

A row that's indented from a row above is considered a *child* of that row. The row above is considered the *parent*. Indenting rows can help organize related records in the list and make it more readable for the user.

You can display indented hierarchy lists on any page type, including List pages, Worksheets, and ListParts. Pages can also be editable.

There are two kinds of indented hierarchy lists: fixed and collapsible. In a fixed hierarchy, rows that are indented are always shown. In a collapsible, users can collapse and expand parent rows to show and hide child records.

Setting up a fixed indented hierarchy

In a fixed hierarchy, child rows are always shown, as illustrated in the following figure.

In the figure, indentation is applied to the second column. Setting up the fixed indented hierarchy involves configuring two properties on the page object: IndentColumn and IndentationControls.

- The IndentationColumn Property determines which records get indented and by how much. You set the property to either a field in the page's source table or to a variable. The important thing, is that property resolves to an integer. This integer determines the indentation level.
- The IndentationControls property specifies which column in the list gets indented. You can only specify one column.

Example

In this example, you indent records based on the value of the **Indent** column and apply the indentation to **Name** column. You set the IndentationColumn and IndentationControls on the repeater of the page, as shown in the following code:

```
page 50100 MyPage
{
    PageType = List;
    ApplicationArea = All;
    UsageCategory = Lists;
    SourceTable = MyTable;
    Editable = true;

    layout
    {
        area(Content)
        {
            repeater(Control1)
            {
                IndentationColumn = Indent;// IndentationColumn = IndentVariable;
                IndentationControls = Name;

                field(Number; Number)
                {
                }
                field(Name; Name)
                {
                }
                field(Indent; Indent)
                {
                }
            }
        }
    }

    //trigger OnAfterGetRecord()
        //begin
          //IndentVariable := Indent;
        //end;

        //var
           //IndentVariable: Integer;
}
```

You can achieve the same results using a variable instead of the table field for the IndentationColumn property. Look at the commented lines of code in the example above.

For a more detailed implementation example, see the Chart of Accounts page in the base application (link requires sign-in to Business Central online).

Setting up a collapsible indented hierarchy

In a collapsible hierarchy, users can collapse and expand parent rows to show and hide child records.

Setting up a collapsible hierarchy is similar to the fixed indented list, except for the properties that you set. A collapsible hierarchy involves three properties: IndentColumn, ShowsAsTree, and TreeInitialState.

- Like in fixed indented hierarchy, the IndentationColumn Property is an integer data type field or variable that determines which records get indented and by how much.

- The ShowAsTree Property makes the hierarchy collapsible.
- The TreeInitialState Property, which is optional, specifies whether the list is collapsed or expanded when the page opens.

Unlike fixed indented lists, a collapsible hierarchy always indents the left-most visible column in the repeater. The IndentationControls property is ignored. If users customize the page by moving another column first, the moved column will be indented instead.

Example

In this example, you'll indent records based on the value of the **Indent** column. Records will indent on the **Number** column and parent records will be collapsible. You add the IndentationColumn, ShowAsTree, and TreeInitialState properties to the pages' repeater:

```
page 50100 MyPage
{
    PageType = List;
    ApplicationArea = All;
    UsageCategory = Lists;
    SourceTable = MyTable;
    Editable = true;

    layout
    {
        area(Content)
        {
            repeater(Control1)
            {
                IndentationColumn = Indent; // IndentationColumn = IndentVariable;
                ShowAsTree = true;
                TreeInitialState = CollapseAll;

                field(Number; Number)
                {
                }
                field(Name; Name)
                {
                }
                field(Indent; Indent)
                {
                }
            }
        }
    }

    //trigger OnAfterGetRecord()
        //begin
          //IndentVariable := Indent;
        //end;

        //var
           //IndentVariable: Integer;
}
```

You can achieve the same results using a variable instead of the table field for the IndentationColumn property. Look at the commented lines of code in the example above.

For a more detailed implementation example, see the Assisted Setup page in the base application (link requires sign-in to Business Central online).

Collapsed or Expanded lists

Users can change whether the page opens with rows collapsed or expanded, essentially overriding the TreeInitialState property. They change the behavior by selecting the **Toggle Expand All / Collapse All** button

in the header of the first column, or using the button in the top-left corner of the repeater. It stays this way, until they delete personalization on the page.

Design and behavior considerations

When using an indented hierarchy, consider the following behavior:

- When indentation is specified, it's no longer possible to use sorting on the columns in the repeater control or display the list as tiles.
- Right-aligned data in columns, such as the integer data type, won't appear as indented.
- Indentation is used to visually communicate structure, without modifying the table of records itself. There's no tightly defined *parent-child* relationship between records, so you must implement additional logic if records need to relate together. For example, if a user deletes a parent record, Business Central won't delete all of its child records.
- Indenting records in a list doesn't automatically apply any additional styling to emphasize parent records and distinguish them from child records. You can implement styling using style expressions. For example, you could format all fields on parent records to display bold values. Learn more about StyleExpr Property.

See Also

IndentationColumn Property
IndentationControls Property
ShowAsTree Property
TreeInitialState Property
Page and Page Extension Properties

Designing Card and Document Pages

2/17/2021 • 7 minutes to read • Edit Online

The *card* page type displays selected fields from an underlying table. The *document* page type is very similar in structure to the card page, but in addition to fields, it also includes a part that includes another page, called a sub-page.

Card pages

You design card pages when you want to enable users to view, create, and modify records (master and reference data) in a table, such as a customer, vendor, or item.

Document pages

Design document pages when you want to represent a transaction or other important event in the domain of business. Document pages are the computerized counterpart to paper-based documents, such as quotes, invoices, orders, and so on. As such, document pages often have associated workflow or audit trail requirements.

Associate with a list page

Both page types are typically associated with list pages (like the customers or sales orders list) that uses the same table as their source. From the list page, users can select a record and open it the card or document page for viewing and editing.

Customizing a card and document pages from the client

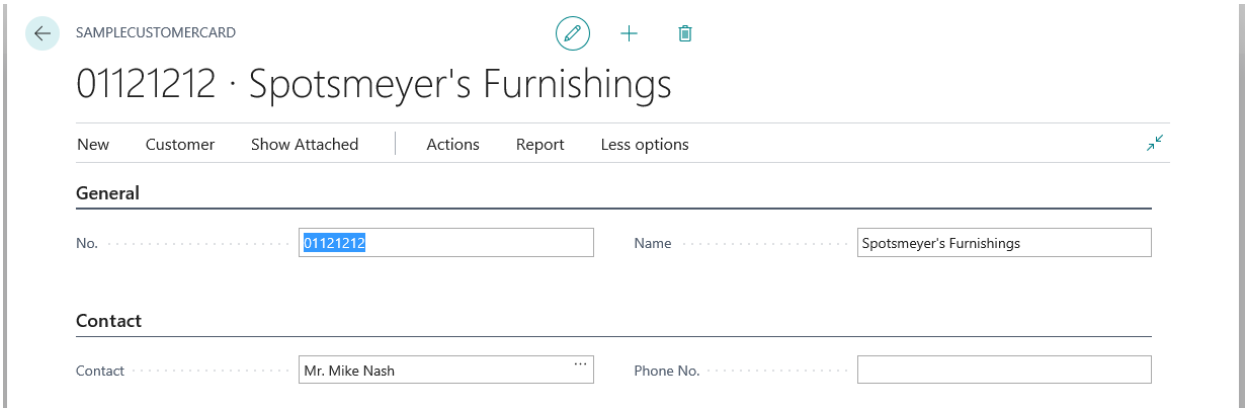
In the client, users can personalize card pages by rearranging or hiding content as they like. For more information, see Personalizing Your Workspace.

As a developer or administrator, you can use Designer to customize a card and document page the same way that individual users personalize their own work spaces. The difference is that changes you make are applied to all users. For more information, see Using Designer.

Simple Card Page Code Example

2/17/2021 • 2 minutes to read • Edit Online

The AL code in this article creates a simple card page that displays records from an existing table.



For a more detailed explanation of the list page, see Designing Card and Document Pages.

```
page 50112 SampleCustomerCard
{
    PageType = Card;
    ApplicationArea = All;
    UsageCategory = Administration;
    SourceTable = Customer;

    //Defines the names for promoted categories for actions.
    PromotedActionCategories = 'New,Process,Report,Manage,New Document,Request Approval,Customer';

    layout
    {
        area(Content)
        {
            //Defines a FastTab that has the heading 'General'.
            group(General)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;

                }
                field(Customer; Name)
                {
                    ApplicationArea = All;

                }
            }

            //Defines a FastTab that has the heading 'Contact'.
            group(Contact)
            {
                field(Name; Contact)
                {
                    ApplicationArea = All;

                }
                field(Phone; "Phone No.")
                {
                    ApplicationArea = All;
```

```
                }
            }
        }
    }

    actions
    {
        area(Processing)
        {

            //Defines action that display under the 'Actions' menu.
            action("New Sales Quote")
            {
                ApplicationArea = All;
                RunObject = page "Sales Quote";
                Promoted = true;
                PromotedCategory = New;
                Image = NewSalesQuote;
                trigger OnAction();
                begin

                end;
            }
            action("Banks Account")
            {
                ApplicationArea = All;
                RunObject = page "Customer Bank Account List";
                Promoted = true;

                //Promotes the action to the category named 'Customer'.
                PromotedCategory = Category7;
                Image = BankAccount;
                trigger OnAction();
                begin

                end;
            }

        }
        area(Reporting)
        {

            //Defines action that display under the 'Report' menu.
            action("Statement")
            {
                ApplicationArea = All;
                RunObject = codeunit "Customer Layout - Statement";
                trigger OnAction();
                begin

                end;
            }
        }
    }
    var
        myInt: Integer;
}
```

See Also

AL Development Environment
Page Extension Object
Actions Overview
Adding Pages and Reports to Search

Page Parts Overview

2/17/2021 • 5 minutes to read • Edit Online

Parts are a special category of page designed to be embedded within another page. The hosting page can be composed of one or more page parts. Parts are useful when designing richer user experiences, by displaying information from a table that is different from the source table of the hosting page. Using parts is a great way to reuse your code across multiple pages.

Hosting parts on a page

The following table illustrates how parts could be creatively arranged on a page to deliver unique experiences.

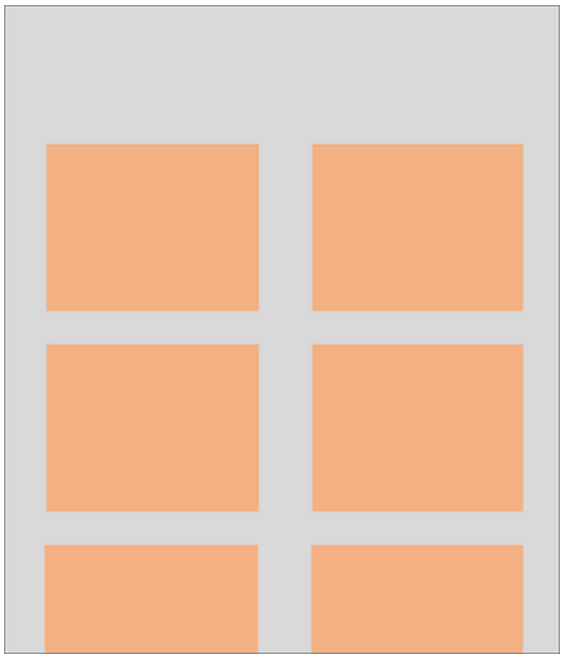
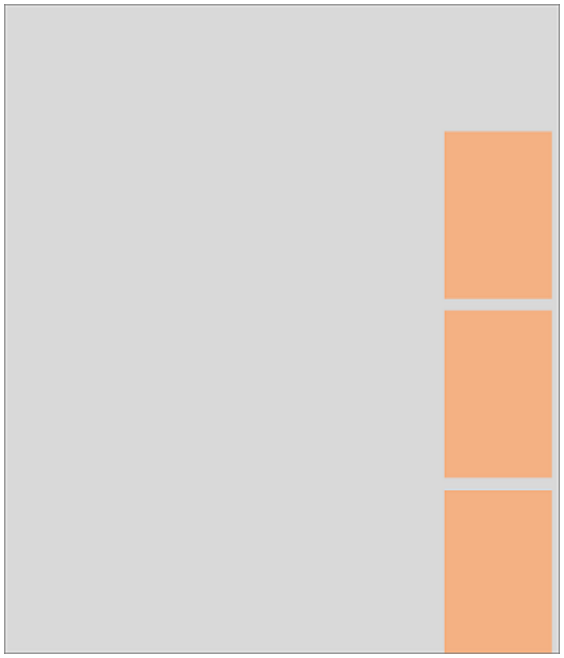
ILLUSTRATION OF PAGE PATTERN	EXPLANATION
	Pages of type **Role Center** are typically composed of multiple pages parts. These parts typically display business headlines, KPIs, and other cues to help users get an overview of their work.
	Most page types can display a number of FactBoxes in the FactBox pane, that display information related to the current record.

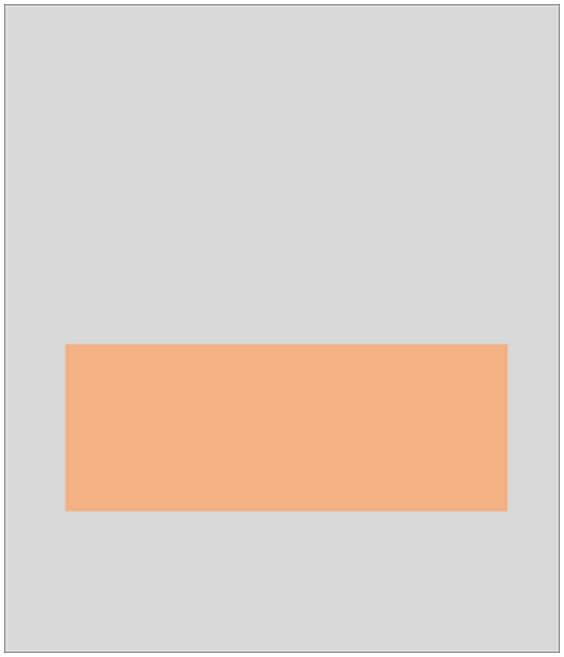
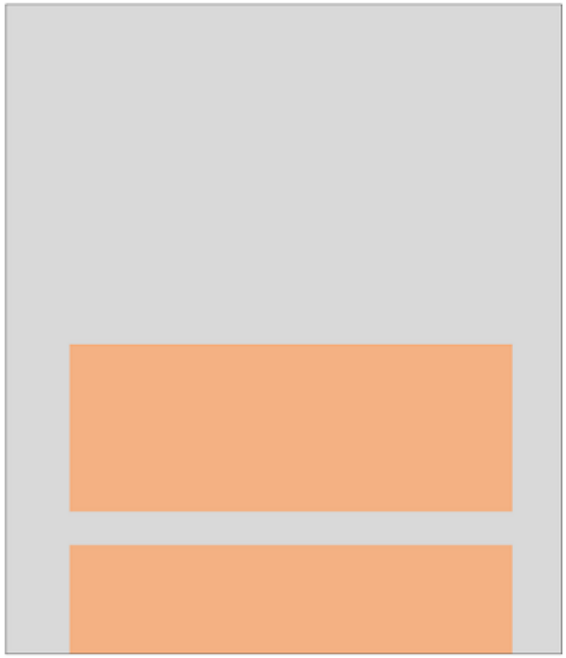
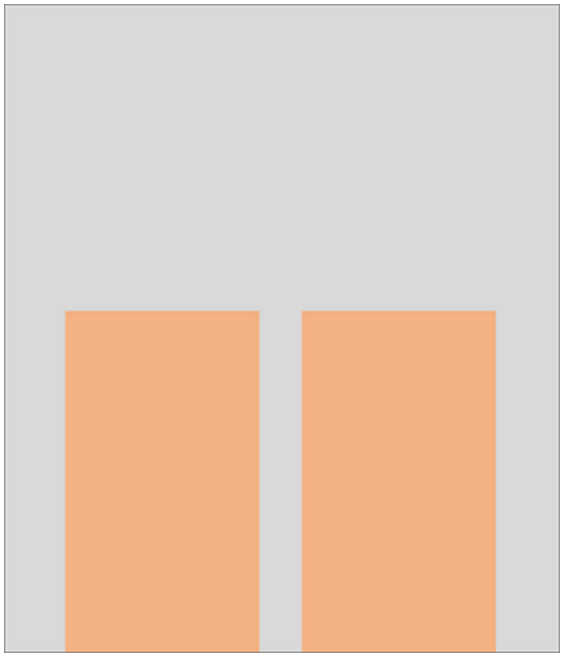
ILLUSTRATION OF PAGE PATTERN	EXPLANATION
	Pages of type **Document** are used to model transactional data that typically consists of a header and detailed lines. Since these records come from different tables, a part is used to display lines that originate from a related table.
	Related lists can be displayed above each other. This pattern is a common pattern allowing one list to be filled by the selected record of another list. **Note:** We're working on adding guidance for more optimal layouts.

ILLUSTRATION OF PAGE PATTERN	EXPLANATION
	When records from two tables need to be compared together, parts can be displayed side by side. **Note:** We're working on adding guidance for more optimal layouts.

Different types of parts

Business Central offers different types of parts that display data in a specific way.

PAGETYPE	PURPOSE	HOSTING PAGE TYPES
ListParts	Display a list of records.	Role Centers; FactBoxes on pages of type Card, Document, Worksheet, List, ListPlus; Tabular step in a Wizard; Subpage on a Document page.
CardParts	Flexible canvas that can be used to display almost any page controls, such as fields, cue tiles, charts, images, or control add-ins.	Role Centers; FactBoxes on pages of type Card, Document, Worksheet, List, ListPlus; Step in a Wizard.
HeadlineParts	Display relevant insights from across the business.	Role Centers only.

Adding a part to a page

To add a part to a page in Visual Studio Code, you add a part control on the hosting page object that references the page part object. The part control also defines a small set of properties, such as the caption that will accompany the part. This allows separation of responsibility: the page part object defines self-contained functionality, whilst the hosting page defines how the container of the part should behave without knowledge of its' functionality.

For more information about the properties of a part control, see Page Properties.

Design considerations

Part size

The size of a part is automatically determined by the user interface and will vary depending on where the part has been embedded on the page, other content surrounding the part, and the overall available space of the

display target. Developers can't specify the preferred, minimum, or maximum height or width of a part.

Part actions

A part can define actions that operate on the fields within the part or navigate to another page. The area in which a part is embedded on a page determines how the action menu is displayed.

- In the `FactBoxes` or `RoleCenter` area of the hosting page, parts display a condensed action menu.
- In the `Content` area of the hosting page, parts will only display an expanded action menu if they satisfy the following rules:
  - parts are hosted on a card or document page.
  - parts aren't placed within a FastTab.

Collapsed or expanded parts

The area in which a part is embedded on a page determines whether the part can be collapsed.

- In the `FactBoxes` or `RoleCenter` area of the hosting page, parts can't be collapsed.
- In the `Content` area of the hosting page, parts can only be collapsed if they satisfy the following rules:
  - parts are hosted on a task dialog, card, or document page.
  - parts aren't placed within a FastTab.

Dynamics 365 Business Central automatically determines whether parts are initially displayed as expanded or collapsed. For example, when a document page is opened the first time, the first two parts or FastTabs are automatically expanded and all other parts and FastTabs are shown as collapsed to provide an optimal starting experience. Users can change the state of a part to be expanded or collapsed directly within the user interface, but developers can't specify the starting state.

Choosing the visibility of parts

Parts can be hidden on the hosting page to provide an optimal starting experience. For example, a part could be hidden because it contains secondary content, or content that is needed by only some categories of users. To hide a part, set the **Visible** property of the part to `false` on the hosting page.

When you design a page with hidden parts, users can choose to display those parts again using personalization and role customization features in the user interface. Parts can be made visible programmatically using an expression, for example, depending on whether the feature has been set up by administrators. Learn more about Dynamic Visibility of Controls.

> **NOTE**
> Parts embedded on Role Center pages can't be made visible using expressions, because the hosting Role Center page can't execute code.

Using page background tasks

Like other page types, you can design a part page to use one or more page background tasks. However, unlike other page types, a part page won't display any data until all page background tasks have completed. For more information about this behavior, see Designing part pages for page background tasks.

Good to know

- Parts can either represent self-contained functionality, or can be contextual and exchange information with the hosting page.
- A part can't be hosted within another part. Business Central allows a maximum of one level of page nesting.
- Parts can't be placed within repeater controls.
- Parts aren't intended to be displayed on their own without a hosting page.

See Also

AL Development Environment
Page Types and Layouts
FactBoxes
Headlines
List Parts
Card Parts
Page Extension Object
Personalizing Your Workspace
Using Designer

Working with List Parts

2/17/2021 • 2 minutes to read • Edit Online

A *ListPart* page is a type of page part used to display a list of records embedded within another page. It consists of a repeater, which presents the records of the source table as rows and columns, and optionally, of an action bar.

A list part can be contained in Role Centers, in the FactBox and content area of other pages, in a tabular step in a Wizard and as a subpage in a Document page. Depending on the type of the hosting page, a list part is subject to different design constraints, which determine its position and dimensions. For more information, see Design Considerations.

Creating a list part

To create a list part, you create a page object, set the PageType Property to `ListPart` and specify the source table. A list part page uses a single `repeater` control, located inside `area(Content)`, where you specify the `field` controls that you want to display. The structure is similar to that of a `List` page, except that there is no FactBox section, since part pages cannot host other parts. For more information, see List Page Structure.

Adding a list part to a page

To place the list part in a page, you add a `part` control to the hosting page and associate it with the list part page object. Here you can also define additional properties. These will only apply to the container of the list part, whose functionality is independent from the hosting page.

Example

The following code sample illustrates how to create a `ListPart` page, `"Pending Shipments"`, and how to integrate it in the card page `"Customer Card"`.

```
page 50101 "Pending Shipments"
{
    PageType = ListPart;
    SourceTable = "Sales Header";
    // Filter on the sales orders that are pending completion.
    SourceTableView = WHERE("Completely Shipped" = CONST(False));

    layout
    {
        area(Content)
        {
            repeater(General)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                }
                field("Order Date"; "Order Date")
                {
                    ApplicationArea = All;
                }
                field("Location Code"; "Location Code")
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}

page 50102 "Customer Card"
{
    PageType = Card;
    SourceTable = Customer;

    layout
    {
        area(Content)
        {
            group(General)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                }
                field(Name; Name)
                {
                    ApplicationArea = All;
                }
            }
            group(Shipments)
            {
                part("Pending Shipments"; "Pending Shipments")
                {
                    // Filter on the sales orders that relate to the customer in the card page.
                    SubPageLink = "Sell-to Customer No." = FIELD("No.");
                }
            }
        }
    }
}
```

See Also

Page Parts Overview

Working with Card Parts

2/17/2021 • 2 minutes to read • Edit Online

A *CardPart* page is a type of page part embedded within another page used to display additional data relevant to the page that hosts it. It can display the data in the form of almost any page control, such as fields, cue tiles, charts, images, or control add-ins. You can also define actions to operate on the card part page.

A card part can be placed in Role Centers, in the FactBox and content area of other pages or in a tabular step in a Wizard. Depending on the type of the hosting page, a card part is subject to different design constraints, which determine its position and dimensions. For more information, see Design Considerations.

Creating a card part

To create a card part, you create a page object, set the PageType Property to `CardPart` and specify the source table. You can nest different controls inside the `area(Content)` control depending on how you want to display the data. The controls you can specify are the following.

CONTROL	DEFINITION
`field`	Defines a field. It can also be used to display an image.
`cuegroup`	Defines a group of cue tiles. For more information, see Designing Cues.
`chartpart`	Defines a chart.
`usercontrol`	Defines a control add-in. For more information, see Control Add-In Object.

Adding a card part to a page

To place a card part in a page, you add a `part` control to the hosting page and associate it with the card part page object. Here you can also define additional properties. These will only apply to the container of the card part, whose functionality is independent from the hosting page.

Example

The following code sample illustrates how to create a `CardPart` page, `"Customer Sales History"`, and how to integrate it in the FactBox area of the card page `"Customer Card"`.

```
page 50101 "Customer Sales History"
{
    PageType = CardPart;
    SourceTable = Customer;

    layout
    {
        area(Content)
        {
            // Display data as cue tiles
            cuegroup(Overview)
            {
```

```
                field("No. of Quotes"; "No. of Quotes")
                {
                    ApplicationArea = All;
                    // Make the cue interactive
                    DrillDownPageID = "Sales Quotes";
                }
                field("No. of Orders"; "No. of Orders")
                {
                    ApplicationArea = All;
                    DrillDownPageID = "Sales Order List";
                }
                field("No. of Invoices"; "No. of Invoices")
                {
                    ApplicationArea = All;
                    DrillDownPageID = "Sales Invoice List";
                }
            }
        }
    }
}

page 50102 "Customer Card"
{
    PageType = Card;
    SourceTable = Customer;

    layout
    {
        area(Content)
        {
            group(General)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                }
                field(Name; Name)
                {
                    ApplicationArea = All;
                }

            }
        }

        // Display the card part in the Factbox area
        area(FactBoxes)
        {
            part("Customer Sales History"; "Customer Sales History")
            {
                // Filter on the sales history that relate to the customer in the card page.
                SubPageLink = "No." = FIELD("No.");
            }
        }
    }
}
```

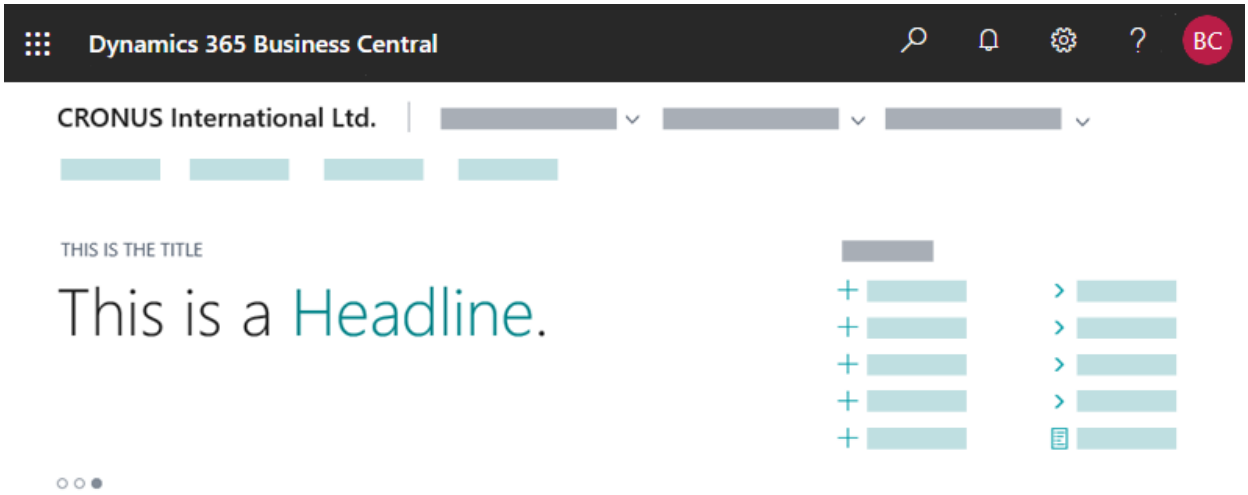
See Also

Page Parts Overview
Page Types and Layouts
Page Object
Page Extension Object
Adding a FactBox to a page
ListPart Pages
AL Development Environment

Creating a Role Center Headline

2/17/2021 • 5 minutes to read • Edit Online

You can set up a Role Center to display a series of headlines, where headlines appear one at a time for a predefined period of time before displaying the next.



The headlines can provide users with up-to-date information and insight into the business and daily work. Typical categories of headlines might include:

- My performance
- My workday
- Organizational health
- Productivity tips
- Cross-tenant insights (performance relative to peers)
- Getting started information

> **IMPORTANT**
> Headlines will only appear in the Web client; they will not be shown on other client types.

Design concept

In development

In short, the Headline is basically a page that contains one or more fields. The page must be the **HeadlinePart** type page. Each field defines an individual headline to be displayed. The source for a field can be an expression or a field in an underlying table.

- The **HeadlinePart** page is designed for Role Centers, that is, pages that have the type **RoleCenter**. If you use a **HeadlinePart** page on another page type, the part will not render in the client.
- Using the OnDrillDown trigger, headlines can be made interactive, meaning that users can select the headline to dig deeper into numbers or values that are shown in the headline or link to another page or URL.
- You can dynamically toggle visibility of a specific headline, for example based on its relevancy, by setting the Visible property on the field.

- There are only a few field properties that apply to fields that are used on a **HeadlinePart** type page, including Expression, Visible, ApplicationArea, Drilldown, and DrillDownPageID. All other properties are ignored.

In the client

The Role Center will start by displaying the first visible headline that is defined on the HeadlinePart page. The headline will appear for 5 seconds, then the next headline will appear for 5 seconds, and so on. When all the headlines have been displayed, it will cycle back to the first headline, and continue from there.

- If a headline is interactive, users can select the headline to open the target defined in the headline.
- Users can pause on a headline by pointing to it.
- Users can manually switch among headlines by selecting a corresponding dot that is displayed under the headlines.
- Users can personalize their Role Center to show or hide the Headline part as they like.

Creating a HeadlinePart page

1. Implement the logic that resolves field expressions for the headlines that you will use on the page.

   You can apply more flexible and complex patterns, such as having data tables drive the text, drill-down and relevance engine for headlines.

2. Create a page that has the PageType property set to `HeadlinePart`.

3. For each headline, add a field, and set the Expression property. The order of the fields, determines the order in which they appear.

   The following example shows the AL code for a simple **HeadlinePart** page that consists of four fields that display static text.

```
page 50100 RoleCenterHeadline
{
PageType = HeadLinePart;

layout
{
    area(content)
    {
        field(Headline1; hdl1Txt)
        {

        }
        field(Headline2; hdl2Txt)
        {

        }
        field(Headline3; hdl3Txt)
        {

        }
        field(Headline4; hdl4Txt)
        {

        }
    }
}

var
    hdl1Txt: Label 'This is headline 1';
    hdl2Txt: Label 'This is headline 2';
    hdl3Txt: Label 'This is headline 3';
    hdl4Txt: Label 'This is headline 4';
}
```

4. You can now add the **HeadlinePart** page to the **RoleCenter** page.

Constructing Headlines with the Expression property

The `Expression` property supports the following syntax that enables you to specify a title for the headline, the headline text itself, and emphasize a string of text in the headline:

```
'<qualifier>Title</qualifier><payload>This is the <emphasize>Headline</emphasize>.</payload>'
```

TAG	DESCRIPTION
`<qualifier></qualifier>`	Specifies the title that appears above the headline. If you omit this tag, the text **HEADLINE** will be used by default.
`<payload></payload>`	Specifies the actual headline text.
`<emphasize></emphasize>`	Applies the style to the text.

The `Expression` property must evaluate to the correct syntax. For example, looking back at the previous example, the label `hdl1Txt` could be:

```
hdl1Txt: Label '<qualifier>The first headline</qualifier><payload>This is the <emphasize>Headline
1</emphasize>.</payload>';
```

Making headlines interactive

You can use the OnDrillDown trigger of a headline field to link the headline to more details or relevant information about what is shown in the headlines. For example, if the headline announced the largest sales order for the month, you could set up the headline to open a page that shows a sorted list of sales order for the month.

The following code uses the OnDrillDown trigger to link `Headline1` to the Dynamics 365 online help.

```
field(Headline1; hdl1Txt)
{
    trigger OnDrillDown()
    var
        DrillDownURLTxt: Label 'https://go.microsoft.com/fwlink/?linkid=867580', Locked = True;
    begin
        Hyperlink(DrillDownURLTxt)
    end;
}
```

Changing the visibility of headlines

You can use the Visible property to show or hide headlines that are defined on the **HeadlinePart** page. With the `Visible` property, you can show or hide the control either statically by setting the property to **true** or **false**, or dynamically by using a `Boolean` variable.

Static visibility

With static visibility, you can simply set the `Visible` property on specific fields. For example, following code hides `Headline3`:

```
{
    field(Headline1; hdl1Txt)
    {

    }
    field(Headline2; hdl2Txt)
    {

    }
    field(Headline3; hdl3Txt)
    {
         Visible=false;
    }
    field(Headline4; hdl4Txt)
    {
    }
}
```

By adding fields under `Group` controls, you can hide or show more than one headline by setting the `Visible` property on the `Group` control. For example, the following code hides headings `Headline3` and `Headline4`:

```
group(Group1)
{
    field(Headline1; hdl1Txt)
    {
    }
    field(Headline2; hdl2Txt)
    {

    }
    }
group(Group2)
{
    Visible=false;
    field(Headline3; hdl3Txt)
    {
    }
    field(Headline4; hdl4Txt)
    {
    }
}
```

IMPORTANT

Unlike other page types, the `group` control has no effect on the UI on pages of type **HeadlinePart**. Its primary purpose is to enable developers to group headlines for controlling visibility.

Dynamic visibility

With dynamic visibility, you can show or hide a headline based on a condition that evaluates to `true` or `false`.

- To dynamically show or hide a headline when the **HeadlinePart** page opens, the headline field must be in `group` control, and you set the `Visible` property on the `group` control to the `Boolean` variable that determines the visibility. For example, you could add code on the page's `OnAfterGetRecord` trigger that evaluates the relevance of displaying `Headline3` and results in a `Boolean` variable being set to `true` or `false`.
- To dynamically show or hide a headline while a page is open, you set the `Visible` property on the `field` control to the `Boolean` variable that determines the visibility.

```
group(Group1)
{
    field(Headline1; hdl1Txt)
    {
    }
    field(Headline2; hdl2Txt)
    {

    }
    }
group(Group2)
{
    // Determines visibility when the page opens
    Visible=ShowHeadline3;
    field(Headline3; hdl3Txt)
    {
        // Determines visibility while the page is open
        Visible=ShowHeadline3;
    }
    field(Headline4; hdl4Txt)
    {
    }
}
```

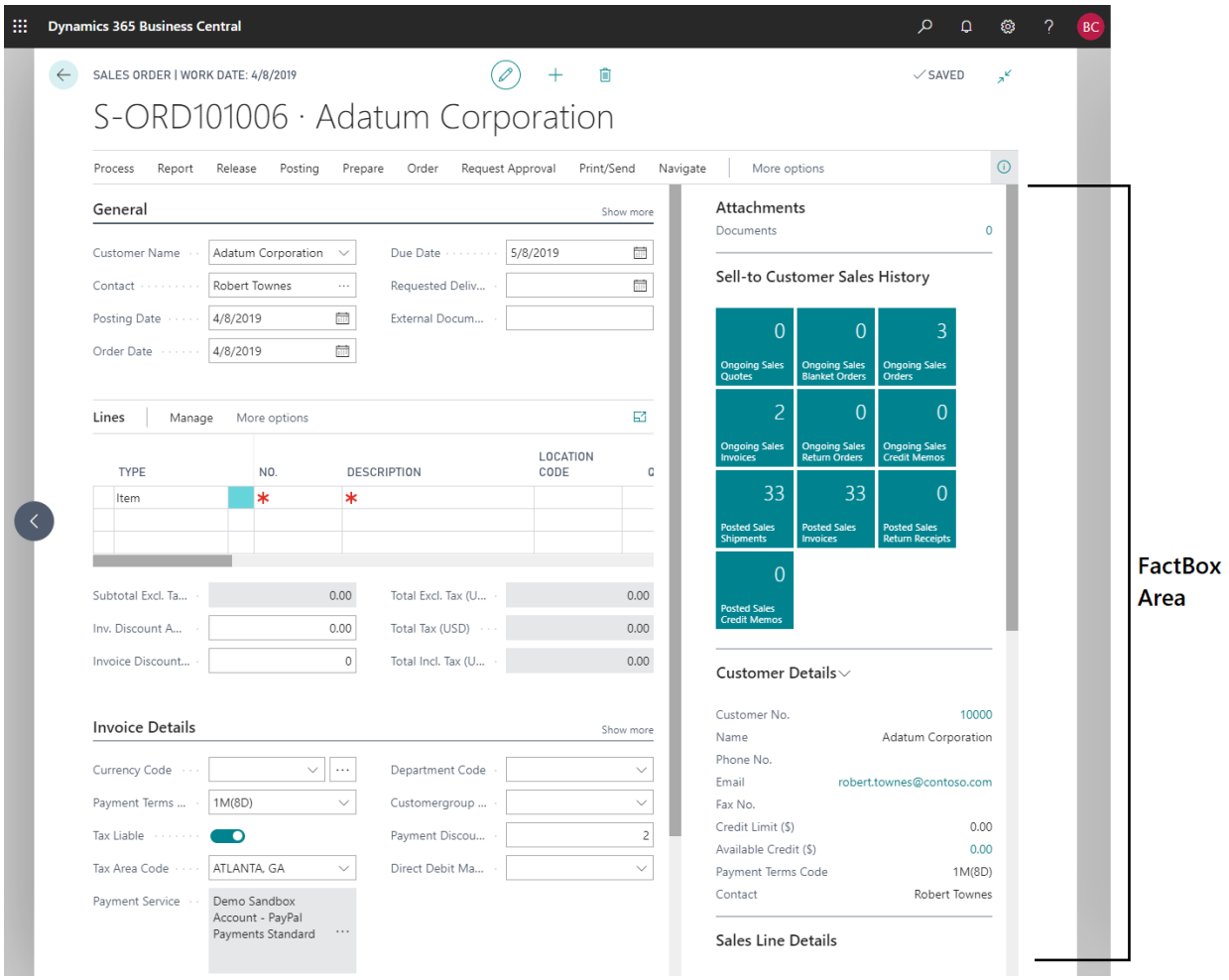
See Also

Pages Overview
Page Object

Adding a FactBox to a Page

2/17/2021 • 6 minutes to read • Edit Online

A FactBox is the area that is located on the right-most side of a page and it is divided into one or more parts that are arranged vertically. This area is used to display content including other pages, charts, and system parts such as Notes, and Links. Typically, you can use a FactBox to display information that is related to an item on the main content page. For example, on a page that shows a sales order list, you can use a FactBox to show sell-to customer sales history for a selected sales order in the list as shown below.



The following list highlights a few categories of FactBoxes:

- Show related records/fields which are modeled as ListParts or CardParts.
- Show related KPIs which are modeled as CardParts with charts or Cues. For more information, see Designing Role Centers.
- Visualize related data or display from external sources which are modeled as CardParts containing a Client AddIn. For example, Bing maps, PowerBI, Microsoft Social Engagement, and more.

Adding a FactBox area to a page

You define the FactBox by adding a FactBox area container control to the page. The FactBox area control acts as a placeholder to which you can add different parts for the FactBox. You can add a FactBox area control on the following page types.

- Card

- Document
- ListPlus
- List
- Worksheet

> **NOTE**
> Only one FactBox area control is allowed on a page.

> **WARNING**
> You can add a part to the FactBox area that displays an existing page of the CardPart or ListPart type only. If you attempt to use another page type, you will get an error.

Example

```
page 50100 "Simple Customercard Page"
{
    PageType = Card;

    layout
    {
        area(FactBoxes)
        {
            part(MyPart; "Acc. Sched. KPI Web Srv. Lines")
            {
                ApplicationArea = All;
                SubPageView = SORTING ("Acc. Schedule Name");
            }
            systempart(Links; Links)
            {
                ApplicationArea = All;
            }
            systempart(Notes; Notes)
            {
                ApplicationArea = All;
            }
        }
    }
}
```

> **TIP**
> When used on Lists, FactBoxes can be used to show information about the entire list, or more contextually about the user's current selection; the currently selected rows. You can control the filter which gets passed to the FactBox that determines its contextual contents.

System Parts

There are two system parts that you can define by using the `systempart()` keyword: `Links` and `Notes`:

VALUE	DESCRIPTION

VALUE	DESCRIPTION
Links	Allows the user to add links to a URL or path on the record shown in the page. For example, on an Item card, a user can add a link to the supplier's item catalog. The links will appear with the record when it is viewed. When a user chooses a link, the target file opens.
Notes	Allows the user to write a note on the record shown in the page. For example, when creating a sales order, a user can add a note about the order. The note will appear with the item when it is viewed.

NOTE

The `systempart` keyword also includes an `Outlook` and `MyNotes` value, for example, `systempart(Outlook; Outlook)`. These values are only supported by the Dynamics NAV Client connected to Business Central (which has been deprecated after Business Central Spring 2019). These values are ignored in the Business Central Web client.

Filtering data that is displayed on a page in a FactBox

In many cases, you want to change the content that is displayed on the page in the FactBox based on the content of the main page. For example, if the main page is a Customer List, you can have a FactBox that includes the Customer Details page that shows information about a customer. When a user selects a customer in the Customer List, the Customer Details page displays information about the selected customer. To implement this functionality, you set up a table filter that associates a field in the table that is used by the Customer Details page with a field in the table that is used by the Customer List page, as shown in the example below. You can also filter on a constant value or set of conditions.

Example

```
page 50101 "Simple Customerlist Page"
{
    PageType = List;
    SourceTable = Customer;

    layout
    {
        area(content)
        {
            repeater(Control)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                }
            }

        }

        area(FactBoxes)
        {
            part(CustomerList; "Customer Details FactBox")
            {
                ApplicationArea = All;
                SubPageLink = "No." = FIELD ("No.");
            }
        }
    }
}
```

Performance considerations

Having a page composed of multiple FactBox pages that each process data from different sources can degrade performance. To improve responsiveness and the time it takes to load the page, Business Central 2020 release wave 2 and later optimizes the sequence in which content is loaded. The sequence is as follows:

1. Content on the hosting page is loaded first, and users can immediately begin interacting with it.
2. The FactBox pane is loaded next, where each FactBox is loaded independently in sequence starting from the top.
   a. FactBoxes having the `Visible` property evaluate to `false` won't be loaded.
   b. FactBoxes that aren't within view are only loaded when the user scrolls them into view. If the FactBox pane is collapsed, no FactBoxes are loaded until the user expands the FactBox pane.

Below are some practical tips to help you make the most of this optimization:

- Consider hiding any FactBoxes that represent secondary content that only some users will require. Learn more about Choosing the Visibility of Parts.
- For FactBoxes that require heavy processing, consider processing in the page background task. Learn more about Using Page Background Tasks.
- Avoid having triggers on the hosting page that call into a FactBox because this condition forces the FactBox to ignore performance optimizations and load along with the content of the hosting page, adding to the total loading time.

FAQ about performance

Are any FactBox triggers run when the FactBox is hidden?

No. The trigger is only run when the FactBox is visible and within the user's view.

How often are triggers run if the FactBox pane is expanded, collapsed, and then expanded again?

In this scenario, the `OnOpenPage` trigger is only run the first time. Once a FactBox is loaded, it isn't loaded again

for as long as the page remains open.

Are FactBoxes processed asynchronously?

No. This optimization is simply a controlled sequence in which triggers are run, still within the same session as the hosting page. For more information about asynchronous processing in the background, see Designing page parts for page background tasks.

Does this optimization work with SubPageLink or SubPageView properties?

The use of these properties has no effect on the sequence of loading content on a page. Using properties such as `SubPageView` is preferred to writing trigger code to update a FactBox.

Does this optimization apply to parts that aren't FactBoxes?

This optimization doesn't apply to Role Center pages. When parts are used in the content area of a page, such as on a Card page, they aren't loaded if their `Visible` property evaluates to `false`.

Can I force a FactBox to load along with page content?

There's no AL API to force FactBoxes to load along with the content of the hosting page.

Can I set the FactBox pane to start collapsed on all pages?

No. The default state of the FactBox pane is set by the Business Central platform and modifed by the user.

Does the experience vary on different browsers?

Each browser has its own definition of whether a FactBox is considered within view or not. For example, opening Business Central in a new browser tab and quickly switching back to the original tab may pause loading of any FactBoxes in the new tab.

Does this optimization apply to other form factors?

This applies to desktop, tablet, and phone clients.

See Also

Pages Overview
Page and Page Extension Properties Overview
Designing Role Centers
Using Designer
Arranging Fields on a Fasttab
Actions Overview

Field Arrangement on FastTabs

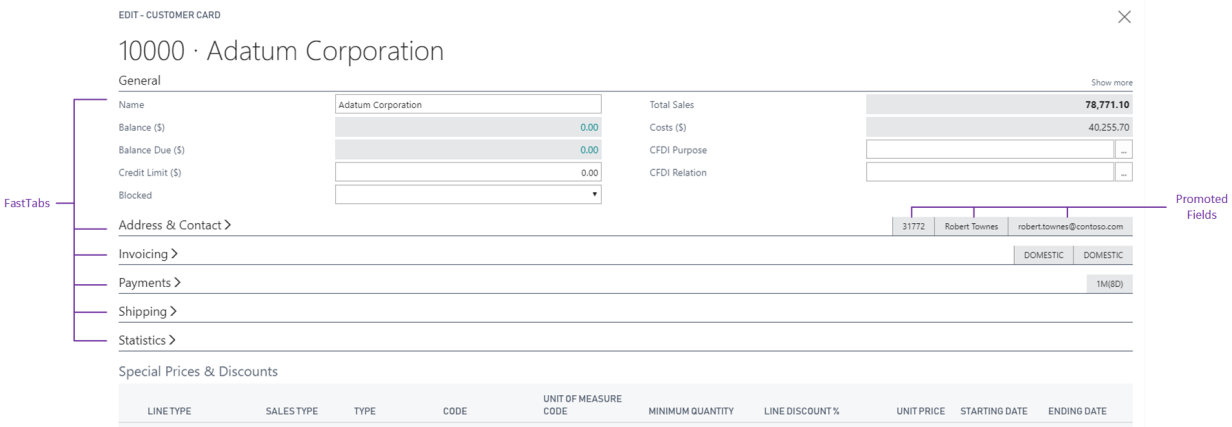
2/17/2021 • 3 minutes to read • Edit Online

FastTabs in Dynamics 365 Business Central allow users to find key information on a page by displaying the data in separate groups. This article describes how individual fields are arranged on a FastTab and ways that you can change the layout.

Organizing data using FastTabs helps users to find key information quickly, while at the same time giving an overview of areas that otherwise would remain hidden. For example, the customer card page displays customer information in the following categories: General, Communication, Invoicing, Payments, Shipping, and Foreign Trade. Each category is a separate FastTab that can be expanded or collapsed, making it easier for users to focus on one subject at a time. On task pages, a FastTab typically represents a single step in a task.

How fields are arranged on a FastTab of a page

By default, a FastTab is divided into two columns for containing fields. Fields are automatically distributed between the left and right columns based on their order in the `layout` section of the page. Starting with the first field in the Page Layout and working downward, fields are positioned in the left column and then in the right column. The area that is occupied by the fields in each column is as equal as possible. Field captions are positioned to the left of fields.



Pages automatically adjust to the available space on the screen. If horizontal space is reduced, a FastTab will adapt and distribute fields into a single column. Similarly, a FastTab will automatically distribute fields into more than two columns to take advantage of wider screens.

FastTab example

Creating a FastTab is easy. A FastTab is a group control directly within the `content` area of a card, document, or task page. The following example shows how you can create a FastTab containing a pair of fields.

```
page 50101 SimpleCustomerCard
{
    PageType = Card;
    SourceTable = Customer;

    layout
    {
        area(content)
        {
            group(GeneralFastTab)
            {
                CaptionML = ENU = 'General';

                field(Name; Name)
                {
                    ApplicationArea = All;
                }
                field(Address; Address)
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}
```

Collapsed or Expanded FastTabs

Dynamics 365 Business Central automatically determines whether FastTabs are initially displayed as expanded or collapsed. For example, when a document page is opened the first time, the first two parts or FastTabs are automatically expanded. All other parts or FastTabs are shown as collapsed to provide an optimal starting experience. Users can change the state of a FastTab to be expanded or collapsed directly within the user interface, but developers can't specify the starting state.

Choosing how to show fields in a FastTab

If a FastTab is expanded, you see all the fields. If it's collapsed, you just see the summary line. The summary line is the header that displays a name for the group, such as 'Communication'. It can also include important fields such as 'E-mail'. Promoting fields to the summary line enables you to present key information to the user, even if the FastTab is collapsed. You can also specify fields that only appear when the users select the **Show more** action on the FastTab. You promote a field or display it only when **Show more** is selected by setting the Importance property of the field.

> **NOTE**
>
> If a group doesn't specify the **CaptionML** property or this is set to an empty value, it's considered to be a group used only for structural purposes. This includes FastTabs. Structural FastTabs look and behave differently, for example, they can't be collapsed by users unless they include the **Show more** action.

Organizing compound content in a FastTab

By using a Group control within a FastTab, you can group fields by similarity. This pattern also gives you control over how fields are distributed between the left and right columns. When you group fields on a FastTab, the groups are distributed evenly between the left and right columns. Fields aren't.

Similarly, page parts placed within a FastTab are distributed evenly between the columns. For example, a FastTab containing two ListParts may be used to display two lists side by side for easier comparison. Learn more about

Page parts.

Manually arranging fields in multiple rows and columns

Using the GridLayout or FixedLayout controls, you can arrange fields in multiple rows and columns in a grid-like format. These controls define a static layout that doesn't automatically adapt to the available space on the screen. It's recommended to use these controls only when you're sure your users will access them on larger screens. For more information, see Arranging Fields Using Grid and Fixed Controls.

See Also

Arranging Fields Using Grid and Fixed Controls
Pages Overview
Using Designer
Table in Dynamics 365 Business Central

Arranging Fields in Rows and Columns Using the Grid Control

2/17/2021 • 3 minutes to read • Edit Online

By default, fields in a FastTab are arranged automatically in two columns that are based on the number of fields. For more information, see Field Arrangement on a FastTab. You can use a Grid control or a Fixed control to arrange fields in rows and columns on a page and design it to look like a grid-like format or a matrix-like format. To understand the differences between the two controls to help you determine which control to use, see Comparing Grid and Fixed controls.

> **NOTE**
> Grid control for arranging page fields is partially supported.

Using the **Grid** control, you can arrange the fields manually in one or more rows and columns. The **Grid** control gives you the following options:

- Set up your grid row-by-row or column-by-column.
- Span a field across multiple rows and columns.
- Show or hide field captions.

Setting-up fields in rows and columns in a FastTab

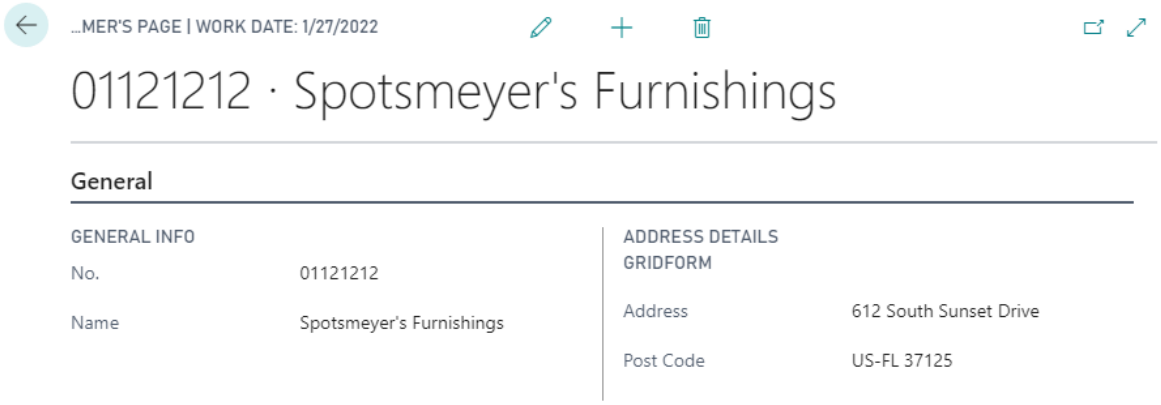
To set up a grid in row-by-row or column-by-column format, you define the **Grid** control in a FastTab of a page. You must define the **Grid** control in a group and specify how you want to arrange the fields by using the **GridLayout** property. For more information, see GridLayout Property.

Example

The following example demonstrates how to structure a page in a grid-like format.

```
page 50113 "Customers Page"
{
    PageType = Card;
    SourceTable = Customer;
    layout
    {
        area(content)
        {
            group(General)
            {
                grid(MyGrid)
                {
                    group("General Info")
                    {
                        field("No."; "No.")
                        {
                            ApplicationArea = All;
                        }
                        field(Name; Name)
                        {
                            ApplicationArea = All;
                        }
                        field("E-Mail"; "E-Mail")
                        {
                            ShowCaption = false;
                            ApplicationArea = All;
                        }
                    }
                    group("Address Details")
                    {
                        grid(Grid2)
                        {
                            group(GridForm)
                            {
                                field(Address; Address)
                                {
                                    ApplicationArea = All;
                                }
                                field("Post Code"; "Post Code")
                                {
                                    ApplicationArea = All;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

The following screenshot shows how the resulting page looks like from the Web client.

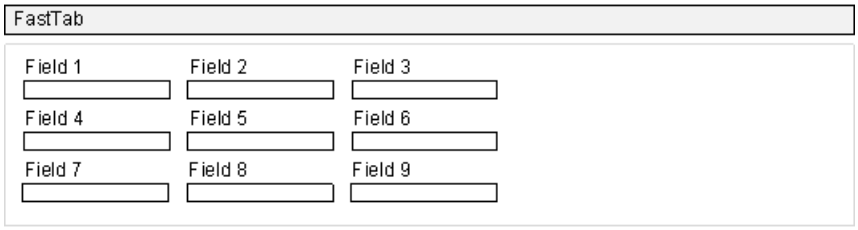


Setting fields to span multiple rows and columns

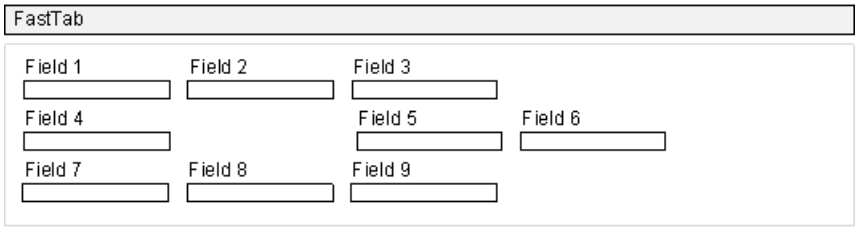
You can set a field to span multiple rows or columns. When you set a field to span multiple rows, the field occupies the cells in the rows below it, and existing fields in the occupied cells are moved to the right. When you set a field to span multiple columns, the field occupies the cells in the columns to the left, and existing fields in the occupied cells are moved to the right. You can also set a field to span multiple rows and columns.

> **IMPORTANT**
> The Dynamics 365 Business Central web client does not support row and column spanning for fields. If the page displays in the Dynamics 365 Business Central web client, the fields appear without spanning.

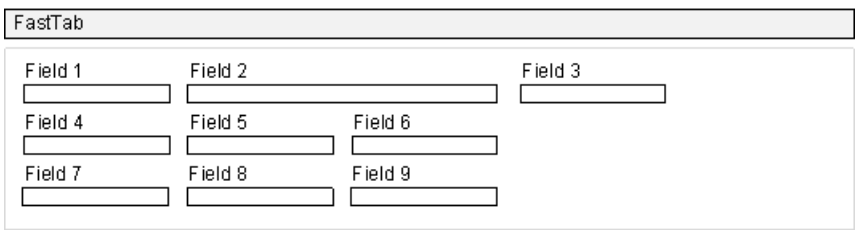
For example, the following figure illustrates a **Grid** control that consists of six fields arranged in three rows.



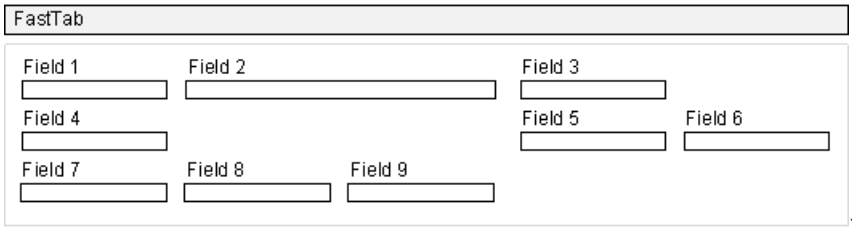
If you set **Field 2** to span two rows, then the following layout is displayed:



When you set a field to span multiple columns, the field occupies the cells in the columns to the right, and existing fields in the occupied cells are moved to the right. Using the previous **Grid** example, if you set **Field 2** to span two columns instead of two rows, the following layout is displayed:



You can also set a field to span multiple rows and columns. For example, if you set **Field 2** to span two rows and two columns, the following layout is displayed:



To set a field to span rows and columns

When you set the **Grid** control, the fields of that group can be set to span rows or columns.

- To set a field to span one or more rows, set the value of the **RowSpan** property to the number of rows. For more information, see RowSpan Property.
- To set a field to span one or more columns, set the value of the **ColumnSpan** property to the number of columns. For more information, see ColumnSpan Property.

> **NOTE**
>
> The **RowSpan** and **ColumnSpan** properties on fields in the grid layout are not supported in the Dynamics 365 Business Central web client. The **Rows** layout on the grid control itself is not supported.

Hiding field captions

You can hide the caption of a group or a field. To hide the caption of a field, set the value of the **ShowCaption** property to **False**. For more information, see ShowCaption Property.

See Also

Field Arrangement on FastTabs
Arranging Fields Using Grid and Fixed Controls
Arranging Fields in Rows and Columns Using the Fixed Control.

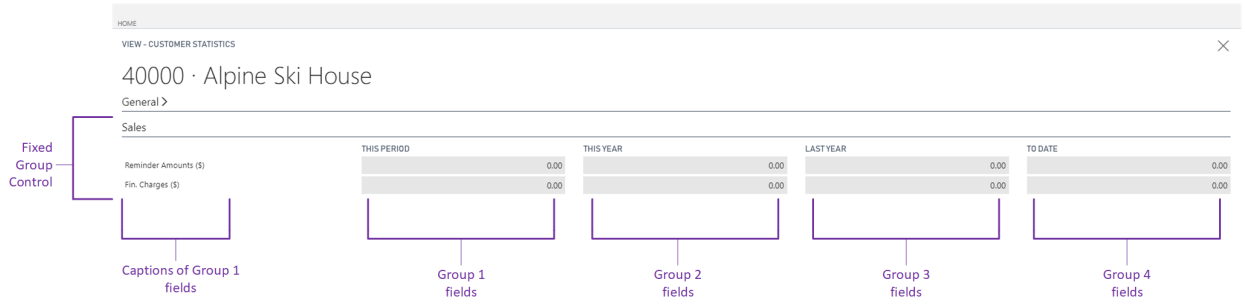
Arranging Fields in Rows and Columns Using the Fixed Control

2/17/2021 • 3 minutes to read • Edit Online

By default, fields on a FastTab are arranged automatically in two columns based on the number of fields. For more information on how the fields are placed on a page, see Field Arrangement on a Fasttab. To manually arrange fields, you can either use a Grid control to design the page to look like a grid-like format, or a Fixed control to design the page to look like a matrix-like format. To understand the differences between the two controls to help you determine which control to use, see Comparing Grid and Fixed controls.

How Fixed control works

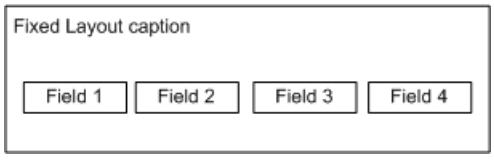
You use the Fixed control to arrange page fields in rows and columns to form a matrix-like layout except that the Fixed control contains a specific number of fields, and a matrix can contain an unspecified number of fields. A Fixed group control is typically used to display statistical data. The following illustration shows an example of a page that uses a Fixed control to show sales totals for different time periods.



You can also use a Fixed control to display information in the details section of a Worksheet page. If you are using the CRONUS International Ltd. demonstration database, then you can see examples of these uses in page 151, Customer Statistics, and page 40, Item Journal.

Adding fields

You can add fields directly in the Fixed control. However, when you add fields directly in the Fixed control, all the fields will display in an equal size and the larger fields will get compressed. The following illustration shows the resulting field layout on a page.



Grouping fields in a Fixed control

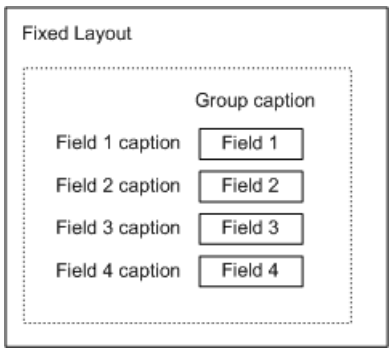
By placing the fields in a Fixed control throughout a group control, you can define separate rows and columns to create a matrix-like arrangement. The group control caption appears as the column header, and the field control captions appear as the row headers. If you add two more group controls that contain fields, then the layout on the page will display like a table format.

Example

The following AL code uses **Fixed** control to display four fields on a page inside the group called Fixed Layout.

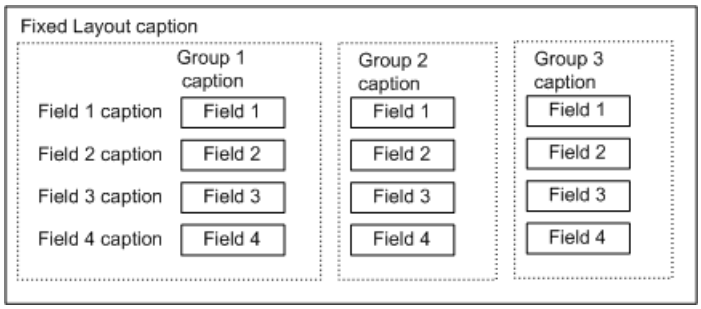
```
page 50114 "Fixed Control Example"
{
    layout
    {
        area(content)
        {
            group("Fixed Layout")
            {
                fixed(DefiningFixedControl)
                {
                    group("Group Caption")
                    {
                        field("Field 1"; "Field 1")
                        {
                            ApplicationArea = All;
                        }
                        field("Field 2"; "Field 2")
                        {
                            ApplicationArea = All;
                        }
                        field("Field 3"; "Field 3")
                        {
                            ApplicationArea = All;
                        }
                        field("Field 4"; "Field 4")
                        {
                           ApplicationArea = All;
                        }
                    }
                }
            }
        }
    }
   var
        "Field 1": Integer;
        "Field 2": Integer;
        "Field 3": Integer;
        "Field 4": Integer;
}
```

The following illustration shows the resulting field layout on a page.



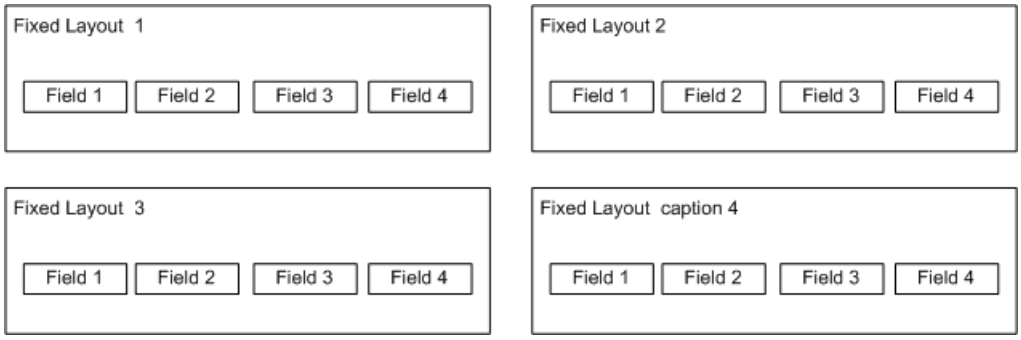
The group control caption appears as the column header, and the field control captions appear as the row headers. If you add two more group controls that contain fields, then the layout on the page will resemble the following illustration.

> **NOTE**
>
> Only the captions of fields in the first column define the row headings. Therefore, only the field captions for the first group control appear. The field captions in other group controls are ignored.



Using multiple Fixed controls

You can also set up more than one Fixed control in a group control. The page area will then divide the fields into two columns that contain the separate Fixed fields. For example, the following illustration shows the page layout if you have four Fixed controls.



> **NOTE**
>
> The fields in the Fixed controls in the illustration are not in a group control. If they were in a group control, then they would follow the same principle as described in the previous section about how to group fields.

> **IMPORTANT**
>
> In previous versions, having a Fixed control directly under a content area was supported. However, in Dynamics 365 Business Central, you must make sure that the Fixed control is nested in a Group control. For more information, see Supported Structure for Using the Grid and Fixed Controls.

Editing fields in a Fixed control

Fields in a fixed layout are not editable even if the **Editable** property is set to **true**. However, if the field drills down to a page where the field source is defined, then you can modify the field. For more information, see Editable Property.

See Also

Field Arrangement on a FastTab
Pages Overview
Arranging Fields Using Grid and Fixed Controls
Arranging Fields in Rows and Columns Using the GridLayout Control

Field Groups (Drop-Down Controls)

2/17/2021 • 2 minutes to read • Edit Online

A field group in table or table extension objects defines the fields to display in a drop-down control on pages that use the table.

In a table object, you define field groups by first adding a `fieldgroups` control, and then adding one or more `fieldgroup(<Name>; <Field>)` keyword for each group, where:

- `<Name>` can be either `DropDown`, for adding fields to the drop-down control, or `Brick` to display data as tiles.
- `<Field>` is a comma-separated list of the fields, by name, to include in the group.

> **NOTE**
> A field group can also be used to specify fields that display when list type pages are shown in the tile view. For more information, see Displaying Data as Tiles.

```
fieldgroups
{
  fieldgroup(DropDown; Field1, Field2)
  {

  }
  fieldgroup(Brick; Field1, Field2)
  {

  }
}
```

> **NOTE**
> The `fieldgroups` keyword cannot be inserted before the `key` control.

> **IMPORTANT**
> The syntax for using a DropDown, must be exactly `DropDown` with the right capitalization.

In a table extension object, the `fieldgroups` control allows you to add more fields to a field group defined for the table object. This can be done by using the `addlast(<name>; <field>)` keyword.

> **WARNING**
> The server will remove the duplicates, if multiple extensions attempt to add the same field more than once. A field can only be added to the field group once.

Define fields for a drop-down control

You define a field to include in a drop-down control by using the `DropDown` field group name in the keyword.

The following example illustrates how to add the field

```
tableextension 50100 CustomerExercise extends Customer
{
    fields
    {
        field(50100; "V02Max"; Decimal) { }
    }

    fieldgroups
    {
        addlast(DropDown; V02Max) { }
    }
}
```

See Also

Debugging in AL
Developing Extensions
Microsoft .NET Interoperability from AL

Field Calculation Methods

2/17/2021 • 5 minutes to read • Edit Online

The following methods perform various actions on fields:

- CalcFields
- CalcSums
- FieldError
- FieldName
- Init
- TestField
- Validate

CalcFields method

CalcFields updates FlowFields. FlowFields are automatically updated when they are the direct source expressions of controls, but they must be explicitly calculated when they are part of a more complex expression. For more information about Flowfields, see FlowFields.

CalcFields has the following syntax.

```
[Ok :=] Record.CalcFields(Field1, [Field2],...);
```

When you use FlowFields in AL methods, you must use the CalcFields method to update them.

In the following example, the SETRANGE method sets a filter and then the CalcFields method calculates the Balance and Balance Due fields by using the current filter and performing the calculations that are defined as the CalcFormula properties of the FlowFields. This example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
Customer	Record	Customer

```
Customer.Get('01454545');
Customer.SetRange("Date Filter",0D,TODAY);
Customer.CalcFields(Balance,"Balance Due");
Message('The Balance is %1 and your Balance Due is %2',Customer.Balance,Customer."Balance Due");
```

The following message is displayed:

The Balance is 342,529.44 and your Balance Due is 342,529.44

CalcSums method

CalcSums calculates the sum of one or more fields that are SumIndexFields in the record.

CalcSums has the following syntax.

```
[Ok :=] Record.CalcSums (Field1, [Field2],...);
```

For CalcSums, a key that contains the SumIndexFields must be selected as the current key. Similar to CalcFields, CalcSums uses the current filter settings when it performs the calculation.

In the following example, an appropriate key is selected, some filters are set, the calculation is performed and then a message is displayed. This example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
custledgerentry	Record	Cust. Ledger Entry

```
custledgerentry.SetCurrentKey("Customer No.");
custledgerentry.SetRange("Customer No.",'10000','50000');
custledgerentry.SetRange("Posting Date",0D,TODAY);
custledgerentry.CalcSums("Sales (LCY)");
Message('%1 calculated sales',custledgerentry."Sales (LCY)");
```

FieldError method

FieldError triggers a run-time error after it displays a field-related error message.

FieldError has the following syntax.

```
Record.FieldError(Field, [Text]);
```

This method is very similar to the Error method. However, in the FieldError method, if the name of a field is changed, for example, translated to another language, in the Table Designer, the message from the FieldError method will reflect the current name of the field.

The following examples show how to use the FieldError method. These examples require that you create the following variable.

VARIABLE	DATE TYPE	SUBTYPE
Item	Record	Item

```
Item.Get('70000');
If Item.Class <> 'HARDWARE' then
   Item.FieldError(Class);
```

If item 70000 has a Class other than HARDWARE, then you receive the following error message:

Class must not be OTHER in Item No. ='70000'.

If the text or code field contains the empty string, then you receive the following error message:

You must specify Class in Item No.='70000'.

If the field is a numeric field and is empty, it is treated as if it contains the value 0 (zero), and then you receive the following error message:

Class must not be 0 in Item No.='70000'.

You can change the default text that is displayed in the error message. The following example shows how to use the FieldError method and change the default text. This example requires that you create the following variable.

VARIABLE	DATA TYPE
Class	Code

```
if Item.Class < '4711' then
   Item.FieldError(Class,'must be greater than 4711');
```

The following error message is displayed:

Class must be greater than 4711 in Item No.='70000'.

FieldName

FieldName returns the name of a field. It has the following syntax.

```
Name := Record.FieldName(Field);
```

You could just use the name of the field. However, using FieldName lets you create messages that always contain the name of the field, even if the name of the field is changed.

This example shows how to use FieldName together with FieldError.

```
FieldError(Quantity,'must not be less than ' + FieldName("Quantity Shipped"));
```

Init

Init initializes a record. It has the following syntax.

```
Record.Init();
```

If a default value for a field has been defined by using the **InitValue** property, this value is used for the initialization. Otherwise, the default value of each data type is used.

> **NOTE**
> Init does not initialize the fields of the primary key.

TestField method

TestField tests whether a field contains a specific value. It has the following syntax.

```
Record.TestField(Field, [Value]);
```

If the test fails, that is, if the field does not contain the specified value, an error message is displayed and a run-time error is triggered. This means that any changes that were made to the record are discarded. If the value that you test against is an empty string, the field must have a value other than blank or 0 (zero).

The following example tests the Language Code field for customer number 10000 in the Customer table and tests whether the Language Code is ZX. This example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
customer	Record	Customer

```
customer.Get('10000');
customer.TestField("Language Code",'ZX');
```

Validate method

Validate calls the OnValidate trigger of a field. It has the following syntax.

```
Record.Validate(Field [, NewValue]);
```

When you enter an account number in a ledger, code in a table trigger is executed to transfer the name of the account from the chart of accounts. If you enter an account number in a batch job, the code which transfers the name of the account is not automatically executed. The following example executes the appropriate field-level trigger code. This example requires that you create the following variable.

VARIABLE	DATA TYPE	SUBTYPE
GeneralLedgerEntry	Record	G/L Entry

```
GeneralLedgerEntry.Validate("G/L AccountNo", '100');
```

This corresponds to the following code.

```
GeneralLedgerEntry."G/L AccountNo" := '100';
GeneralLedgerEntry.Validate("G/L AccountNo");
```

The `Validate` method is useful for centralizing processing, which makes your application easier to maintain.

For example, if the OnValidate trigger of the Total Amount field performs a calculation that uses values from three other fields as operands, the calculation must be performed again if the contents of any one of these fields changes. You should avoid entering the calculation formula in the OnValidate triggers of each field because this can create errors if the calculation formula has to be changed later and you have to update the code in all the triggers. Instead, you should enter the calculation formula in the OnValidate trigger of only one of the fields and call this trigger code from the OnValidate triggers of the other fields.

See Also

AL Methods

Formatting Decimal Values in Fields

2/17/2021 • 5 minutes to read • Edit Online

This article describes how you can format the decimal values that appear in fields on table, pages and reports. For example, you can change how the data appears in a Cue on the Role Center page. To format data, you use a combination of the AutoFormatType Property, AutoFormatExpression Property, and DecimalPlaces Property of the field. These properties work together to enable you to specify the following:

- Display amounts and unit amounts in another currency.
- Specify the number of decimal places.
- Specify whether to display a thousand separator.
- Specify characters before and after the value, such as currency signs or %.

Implementation overview

When a field is used on a page or report, you can set the **AutoFormatType** and **AutoFormatExpr** properties directly on the page field or report field (column), or you can set them on the underlying table field. If you specify the properties on the table field, then the format applies wherever the field is used. Specifying the properties on the page or report field will only apply the format to the specific page or report. If you specify the properties on the table field and the page or report field, then the settings on the page or report field take precedence.

When you use the **AutoFormatType** and **AutoFormatExpression** properties to format a field, two events are raised by the system codeunit **45 Auto Format**: OnResolveAutoFormat and OnAfterResolveAutoFormat.

Setting up data formatting

The settings for the **AutoFormatType**, **AutoFormatExpression**, and **DecimalPlaces** properties will depend on the type of data that is displayed, for example, this could be currency amounts, unit amounts, simple decimals, or ratios. For the most part, the **AutoFormatType** property is the primary setting, which in turn determines the options for setting the **DecimalPlaces** and **AutoFormatExpr** properties.

If the **AutoFormatType** is not set or is set to an incorrect property value, then the default setting is used, regardless of whether the **AutoFormatExpression** or **DecimalValues** properties are set. The default setting uses `AutoFormatType = 1` and `AutoFormatExpression = ''`.

The following tables describes how to set each of the properties to achieve the format that you want.

Setting the DecimalPlaces property

With the following set up, the **AutoFormatExpression** property is ignored.

AUTOFORMATTYPE PROPERTY	DECIMALPLACES PROPERTY	USAGE DESCRIPTION

AUTOFORMATTYPE PROPERTY	DECIMALPLACES PROPERTY	USAGE DESCRIPTION

0	Set to the number of decimal places that you want to display for the value.	Use this configuration when you want to format the decimal value according the Standard Format 0 (which is the default format) with a specific number of decimal places.
		
		For example, if the value is a US decimal `-76543.21` and you set the **DecimalPlaces** property to `0`, then the value appears as **76,543**. The properties will look like this:
		
		`AutoFormatType = 0;`
		`DecimalPlaces = '0';`

Setting the AutoFormatExpression property

With the following setup, the **DecimalPlaces** property is ignored.

AUTOFORMATTYPE PROPERTY	AUTOFORMATEXPRESSION PROPERTY	USAGE DESCRIPTION

1	Set to return a currency code, such as USD or IDR. The blank currency code `''` denotes LCY and is the default value.	Use this configuration when you want to format the data as an amount. For example, a sales order will use two decimals when the currency is defined as US dollar and no decimals when the currency is defined as IDR (Indonesian rupiah). For example:
		
		`AutoFormatType = 1;`
		`AutoFormatExpression = 'IDR';`

2	Set to return a currency code such as USD or IDR. The blank currency code `''` denotes LCY and is the default value.	This is similar to the previous configuration where the **AutoFormatType** property is set to `1`, except you use this configuration when you want to format the data as a unit amount.

AUTOFORMATTYPE PROPERTY	AUTOFORMATEXPRESSION PROPERTY	USAGE DESCRIPTION

10	Set to the property according to the following syntax:
	`'[SubType][,<currencycode or expression>[,<PrefixedText>]]'`
	`SubType` can be `1`, `2`, another number, or omitted:
	`1` sets the value to an amount type (see 1 above). `2` sets the value to a unit amount type (see 2 above). The syntax for these two settings is:
	`'SubType,<currencycode[, <PrefixedText>]'`
	If you omit the subtype or use a number other than 1 or 2, the syntax is:
	`'<CustomNumber>, <expression>[, <PrefixedText>]'`
	where `<expression>` sets the precision and one of the standard formats. For more information, see Standard Formats.
		Use SubType `1` to add the currency symbol and use the amount type precision. You use SubType `2` for unit amount precision. For example, set the property to `'1,USD'` to add the **$** symbol, like **$543.21**.
		`AutoFormatType = 10;`
		`AutoFormatExpression = '1,USD';`
		If you omit the SubType, you can use this configuration to customize the format based on one of the standard formats. This option enables you to specify characters before and after the decimal value, such as currency signs **$** and percent **%**.
		For example, if you want to prefix the decimal value with a **$**, include a thousand separator, and have a maximum of two decimal places, such as **$76,453.21**, then you can set the properties to:
		`AutoFormatType = 10;`
		`AutoFormatExpression = '$<precision, 2:2><standard format, 0>'`
		If you want to display the decimal value as a percentage, then you can add `%` at the end of the setting. For example:
		`AutoFormatType = 10;`
		`AutoFormatExpression = '<precision, 1:1><standard format,0>%'`
		When you include a `%` at the end of the setting, then the decimal value is assumed to be the ratio, and the decimal value will be multiplied by 100. For example, a value of 0.98 will be formatted to **98%**.

11	Set the property to the standard format as explained below. For example: `'<Precision,3:3><Standard Format,0>'`	Use this option when you want full control over the formatting. The format string will be applied exactly as specified in the AutoFormatExpr property.

Precision

The precision determines the minimum and maximum number of decimal points for values. The precision takes the format `<precision,minimum:maximum>`. For example, `<precision,minimum:maximum>` sets the data with a minimum of 2 and a maximum of 3 decimal places.

Standard formats

The following table describes the standard formats that are available for the **AutoFormatExpr** property when the **AutoFormatType** property is set to 10.

STANDARD FORMAT	FORMAT DESCRIPTION	EUROPE DECIMAL EXAMPLE	US DECIMAL EXAMPLE
0	<Sign> <Integer Thousand> <Point or Comma> <Decimals>	-76.543,21	-76,543.21
1	<Sign> <Integer> <Point or Comma> <Decimals>	-76543,21	-76543.21
2	<Sign> <Integer> <Point or Comma> <Decimals>	-76543.21	-76543.21
3	<Integer Thousand> <Point or Comma> <Decimals> <Sign>	76.543,21-	76,543.21-
4	<Integer> <Decimals> <Point or Comma> <Sign>	76543,21-	76543.21-
9	XML format	-76543.21	-76543.21

See Also

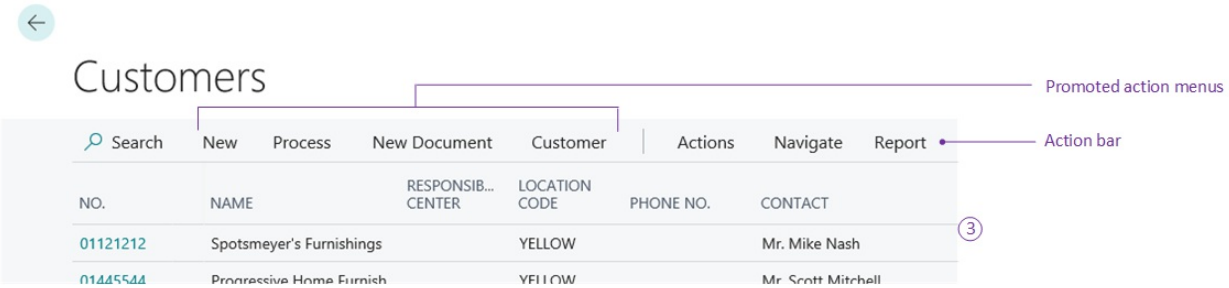
AutoFormatType Property
AutoFormatExpression Property
DecimalPlaces Property

Actions Overview

2/17/2021 • 6 minutes to read • Edit Online

In Dynamics 365 Business Central, actions are displayed at the top of each page, referred to as the action bar. In this topic, you learn about different types of actions, and how you can enable users to quickly locate the actions they want to use.

The actions can be displayed in different menus on the action bar.



You can choose from the following action menus to place the actions in the specified area.

AREA	SYNTAX	USED ON	DESCRIPTION	EXAMPLE
Actions menu	`area(processing)`	Role Center, list, card, and task pages	User tasks	Post a sales order
New document group in Actions menu	`area(creation)`	List, card, Role Center pages, and task pages	Actions that appear under the New group. Opens a new Dynamics 365 document.	New sales invoice
Navigate menu	`area(navigation)`	List, card, and task pages	Links to other pages in Dynamics 365 Business Central.	Prices
Report menu	`area(reporting)`	Role Center, list, card, and task pages	A list of available reports.	Customer Top 10 List

The following actions are related to the Role Center page.

AREA	SYNTAX	USED ON	DESCRIPTION	EXAMPLE
Navigation menus	`area(sections)`	Role Center pages	The top-level navigation consists of one or more root items that expand to display a submenu of links to other pages.	Posted sales invoices
Navigation bar	`area(embedding)`	Role Center pages	The second-level navigation displays a flat list of links to other pages.	Customers

For more information about actions used on the role center page, see Designing Role Centers.

> **TIP**
> If you used to work in Microsoft Dynamics NAV, you can get an overview of the mapping between actions in the Differences in the Development Environments topic.

Types of Actions

Each page has a different set of actions depending on the page type, and the processes that the page supports. In order to create the appropriate set of actions for a particular page, you should have a good understanding of your customer's business processes.

Each process in an organization has several actions associated with it. You should try to create a full set of actions that mirror all tasks and processes that are performed.

Example: The Sales Orders list page at CRONUS International contains all actions related to processing sales orders. During user configuration and personalization, some of these actions may be hidden or promoted to the ribbon. Therefore, you must create a full set of actions for the customer.
Pages can have the following actions as described in each section below.

Actions menu

The Actions menu is a displayed in the action bar on all page types, and contains relevant tasks for the current page. Typically, you add processing tasks and creation tasks in the Actions menu. To add processing actions such as posting a sale order, you must use the `processing` action area. They are regular daily tasks. Therefore, they must be on the Actions menu. For examples on how to add actions to the Actions menu, see Adding Actions to a Page.

Some examples from the Customer page are as follows:

- Sales Invoice
- Sales Quote
- Sales Credit Memo
- Ledger Entries
- Invoice Discounts
- Prices
- Line Discounts

You can add actions to the Actions menu, group actions together under action sub menus, or promote them to the ribbon. For examples of how to use actions, see Page Object and Page Extension Object.

New Document menu

The New Document menu is often displayed both as a top-level menu in the actions bar and as a sub menu in the Actions menu. You can use this menu to open new documents within Dynamics 365. You can add an action to create a new document such as creating a new sales invoice. This action displays in a separate menu called **New document** in the Actions menu. To add to the New document menu, you must use the `creation` action area.

Example: On the Customers page, if the order processor wants to create a new invoice, she can open the new page directly from the Actions menu. This is useful as she creates new sales invoices daily.

Navigate menu

The Navigate menu is displayed after the Actions menu in the action bar. Rather than providing tasks for the user, this menu provides additional information by taking the user to a specific page in Dynamics 365. To add a page link in the Navigate menu, you must use the `navigation` action area. These actions act like a bookmark to enable quick access to view a page.

> **NOTE**
> You should not add a Navigation action to a Role Center page.

Report menu

The Report menu is displayed after the Navigate menu in the action bar. The Reports menu lists the reports most relevant to a page. If a user does not require a Report menu, then the menu is hidden. Sometimes it is relevant to promote the most important reports to the top-level in the action bar to save the user from too many clicks. To create an action in the Report menu, you must use the `reporting` action area.

Promoted Actions

Promoted actions are actions that are set up on the Actions, Navigate, or Reports menus in the action bar, but are also configured to display in custom menus in the action bar. Although the actions are set up on the Actions, Navigate, or Report menus, you can choose to hide them on these menus and only show them in custom menus . For more information on how to add promoted actions, promoted categories and example, see Promoted Actions.

Grouping Actions in Sub-Menus

Within the different areas, you can create sub-menus to a group of actions and improve navigation. You create a sub-menu by adding a `group()` control, as shown in the following example:

```
actions
{
    area(Report)
    {
        action(Report 1)
        {
            RunObject = report "Report1";
        }

        // Adds a sub-menu called "Group1" to the Report menu.
        group(Group1)
        {
            // Adds the action "Report 2" to the My Label sub-menu.
            action(Action2)
            {
                RunObject = report "Report2";
            }
            // Adds a sub-menu called "Group2" to the Group1 sub-menu.
            group(Group2)
            {
                // Adds the action "Report 3" to the Group1 sub-menu.
                action(Action3)
                {
                    RunObject = report "Report3";
                }
            }
        }
    }
}
```

NOTE

Prior to Dynamics 365 Business Central 2019 Wave 2, in the client, sub-menus were automatically placed before single actions on the same level. This means, for example, group **Group2** appears before the action **Report 2**.

Actions at runtime

An action can trigger code to run, such as posting a document or otherwise modifying a record in a table. When a user chooses an action, one of the following pieces of logic will happen in addition to the code that the action itself triggers:

- If the page is empty and no longer shows any records, the page is re-initialized with default values.
- If the page does show records, and the current state is within the page filters boundary, the **OnAfterGetRecord** trigger is executed on the page.
- If the current record that the page showed is now outside the filter but there are other records within the filter, the **OnFindRecord** trigger is called and the **OnAfterGetRecord** trigger is run on the next record with the given filters.

The logic runs in the transaction that the action triggered. This can cause the application code to result in users locking the whole table when they thought they were only modifying one record.

To avoid users accidentally locking tables, you can use the SetSelectionFilter method before your code passes the record variable to the processing codeunit, for example. The following code example illustrates the code on the OnAction trigger on an action on a page.

```
if confirm('Are you sure you want to call this codeunit?', true) then begin
    CurrPage.SetSelectionFilter(Rec);
    codeunit.Run(50000, Rec);
end;
```

See Also

AL Development Environment
Developing Extensions in AL
Pages Overview

Adding Actions to a Page

2/17/2021 • 4 minutes to read • Edit Online

This topic shows how to create new actions, how to add actions to a page, and how to preview them in the Dynamics 365 web client. In Dynamics 365, actions can be displayed in the action bar of all pages and grouped together under the following actions menus:

- Promoted action categories
- Actions
- Navigate
- Report

Before putting an action on a page you should think about the business processes that the action supports. For example, on page 42, the Sales Orders list page, the Actions button contains actions for all tasks related to processing sales orders. Creating these actions can make it easier for the order processor to perform their daily tasks, such as posting sales orders and creating new customer orders.

For more information about different types of actions and where to use them, see Actions Overview.

> **TIP**
> After you have added actions to a page, you can use Designer to alter the actions, like moving an action to or from a promoted category, hiding and action or action group, and more. For more information, see Using Designer.

To add Actions to a Page

The page actions are displayed on the header section. There are multiple tabs to help navigate to the right item.

In order to add actions to the action bar, you must use the keywords with Anchors or Targets. These keywords are used to place and move the actions around in the tab groups. For more information about adding, moving, and modifying actions, see Using Keywords to Place Actions and Controls.

> **NOTE**
> Actions can only be linked to a page, or to a group control. Actions cannot be linked to fields, or parts on a page.

Set an icon to an action

Dynamics 365 Business Central includes images that you can use on actions in command bar menus and promoted actions on the ribbon. To add an image to an action, you add the **Image** property and you must provide the name of the image you that want to use from the Dynamics 365 Business Central Action icon library. By default, the size of images is 16 pixels high by 16 pixels wide. For promoted actions, you can choose to display larger images that are 32 pixels high and 32 pixels wide. For more information, see Image Property.

Example

The following example shows how to use different action areas on a **page object of the PageType Card**. These actions will display in the following menus in the action bar.

1. Actions menu: The `area(Processing)` action area is used to display the action in the Actions menu. This action uses the **Promoted** and **PromotedCategory** properties in order to display the action in the promoted

actions menu called **Process**.

2. New Document group: The `area(Creation)` action area is used to display the action in the **New document** group in the Actions menu. Also, this action uses the **Image** property to display a form icon instead of a default icon.
3. Navigate menu: The `area(Navigation)` action area is used to display the action in the Navigate tab. This action and other actions in this example uses the **RunObject** property to assign a page to the action.
4. Report menu: The `area(Reporting)` action area is used to display this action in the Report menu, and also a Group control is added as a submenu to this menu. It sets the **Caption** property to make the action group visible in the Reports menu.

```
page 50110 PageName
{
    PageType = Card;

    actions
    {
        // Adds the action called "My Actions" to the Action menu
        area(Processing)
        {
            action("My Actions")
            {
                Promoted = true;
                PromotedCategory = Process;
                ApplicationArea = All;
                trigger OnAction()
                begin
                    Message('Hello World');
                end;
            }
        }

        area(Creation)
        {
            // Adds the action "My New document" to the New Document group in the Actions menu.
            action("My New document")
            {
                ApplicationArea = All;
                RunObject = page "Customer Card";
                Image = "1099Form";
            }
        }

        area(Navigation)
        {
            // Adds the action called "My Navigate" to the Navigate menu.
            action("My Navigate")
            {
                ApplicationArea = All;
                RunObject = page "Customer Card";
            }
        }

        area(Reporting)
        {
            // Adds a submenu called "My Label" to the Report menu.
            group(NewSubGroup)
            {
                Caption = 'My label';
                group(MyGroup)
                {
                    // Adds the action "My Report" to the My Label submenu.
                    action("My Report")
                    {
                        ApplicationArea = All;
                        RunObject = page "Customer Card";
                    }
                }
            }
        }
    }
}
```

> **NOTE**
>
> Actions can be assigned to a page by setting the RunObject property, or by adding a trigger to a Codeunit. For more information, see RunObject Property and Codeunit Triggers.

The promoted action menus are always displayed first so the promoted actions provide quick access to common tasks, and users do not have to browse through a menu to access them. Add the Promoted property to add actions to the a promoted action menu. For more information on how to add promoted actions, promoted categories, and examples, see Promoted Actions.

You can assign different icons for your actions from the Dynamics 365 image library. For more information, see Image Property.

See Also

Actions Overview
Pages Overview
Promoted Actions

Promoted Actions

2/17/2021 • 4 minutes to read • Edit Online

Promoted actions are actions that are set up on the Actions, Navigate, or Report menus in the action bar, but are also configured to display on the Home tab. Although the actions are set up on the Actions, Navigate, or Report tabs, you can choose to hide them on these menus and only show them on the Home tab.

The following table describes where you can use promoted actions.

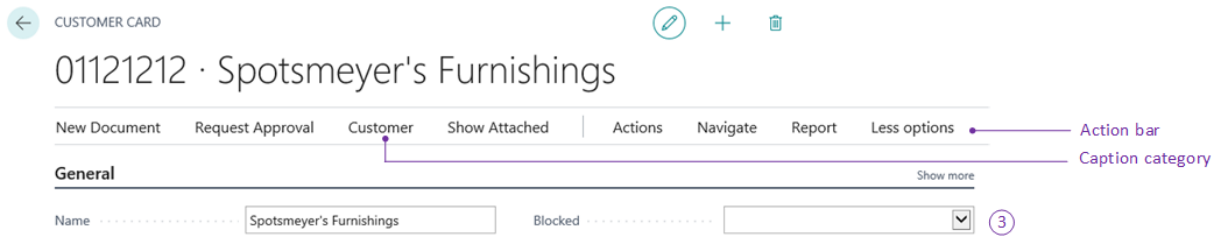
ACTION TYPE	USED ON	DESCRIPTION	EXAMPLE
Promoted Actions	List, card, Role Center pages, and task pages	Provide quick access to common tasks that appear under the Home tab.	Post and print a sales order

You can promote any command from the existing actions menus to the ribbon. If there are no promoted actions, the ribbon remains hidden. To promote an action on the Home tab, you set the **Promoted** property of the action. If you want to display the action only on the Home tab, then you add an additional step to set the **PromotedOnly** property. For more information, see Promoted Property and PromotedOnly Property.

Promote actions by category

Promoted actions can be grouped. You can add promoted actions by different grouped categories. Typically, promoted actions are displayed in the ribbon of the role center client. You can organize promoted actions into different categories, where each category is indicated by a caption in the ribbon. You define up to 10 categories for a page. The following figure illustrates a page that has promoted actions under the following categories.

- New Document
- Request Approval
- Customer



You assign a promoted action to a category by setting the **PromotedCategory** property of the action. By default, these category names correspond to the captions that are displayed for the category on the page in Dynamics 365 Business Central. You will typically want to change the captions, especially the Category4 through Category10 captions. See the table below for the default **PromotedCategory** values. To change the default captions, set the **PromotedActionCategories** property. You type the values of the **PromotedActionCategories** where each caption is separated with a comma as shown below:

```
PromotedActionCategories =
'New_caption,Process_caption,report_caption,category4_caption,category5_caption,category6_caption,category7_
caption,category8_caption,category9_caption,category10_caption';
```

The position of the caption in the list determines its corresponding category setting in the **PromotedCategory**

property for the actions as described in the table below.

PROMOTEDACTIONCATEGORIES CAPTION POSITION	DEFAULT PROMOTEDCATEGORY VALUES	EXAMPLE
First	New	*New_caption*
Second	Process	*Process_caption*
Third	Report	*Report_caption*
Fourth	Category4	*Category4_caption*
Fifth	Category5	*Category5_caption*
Sixth	Category6	*Category6_caption*
Seventh	Category7	*Category7_caption*
Eighth	Category8	*Category8_caption*
Ninth	Category9	*Category9_caption*
Tenth	Category10	*Category10_caption*

You can change category captions on a page-by-page basis and for each Dynamics 365 Business Central Windows client language.

For more information about these properties, see PromotedCategory Property and PromotedActionCategories Property.

Assigning an icon to the promoted actions

Each promoted action has an icon associated with it. You can accept a default icon, or choose your own from the Dynamics 365 Business Central image library by using the **Image** property, where each promoted action has an icon associated with it. Also, you can use a larger icon that makes it more prominent to the user by using the **PromotedIsBig** property. For more information, see Image Property and PromotedIsBig Property.

Example

The example shows how to promote actions on a Customers page using different properties:

1. The actions in the example are promoted to display in the **New Document**, **Request Approval** and **Customer** groups on the Home tab.
2. The **Sales Quote** and **Sales Invoice** actions are promoted to the ribbon and grouped in a category called **New Document**.
3. The `PromotedCategory` value; `Category5` corresponds the caption position in the `PromotedActionCateogories` value with the **New Document** caption.
4. Each promoted action in the example is assigned to a unique icon. Additionally, to display bigger icons, the **Sales Quote** and **Contact** actions are set with the `PromotedIsBig` property.
5. The **Sales Quote** and **Send Approval Request** actions are set to appear only on the Home tab.

```
page 50103 Customers
{
    PageType = Card;
    SourceTable = Customer;
```

```
    SourceTable = Customer;
    PromotedActionCategories = 'New,Process,Report,Manage,New Document,Request Approval,Customer,Page';

    actions
    {
        area(Creation)
        {
            action("Sales Quote")
            {
                Promoted = true;
                PromotedCategory = Category5;  // PromotedActionCategories = New Document
                PromotedOnly = true;
                PromotedIsBig = true;
                Image = NewSalesQuote;
                ApplicationArea = All;
                trigger OnAction()
                begin
                    Message('Create sales quote');
                end;
            }
            action("Sales Invoice")
            {
                Promoted = true;
                PromotedCategory = Category5;  // PromotedActionCategories = New Document
                Image = SalesInvoice;
                ApplicationArea = All;
                trigger OnAction()
                begin
                end;
            }
        }
        area(Processing)
        {
            action("Send Approval Request")
            {
                Promoted = true;
                PromotedOnly = true;
                PromotedCategory = Category6;  // PromotedActionCategories = Request Approval
                Image = SendApprovalRequest;
                ApplicationArea = All;
                trigger OnAction()
                begin
                end;
            }
            action("Cancel Approval Request")
            {
                Promoted = true;
                PromotedCategory = Category6;  // PromotedActionCategories = Request Approval
                Image = CancelApprovalRequest;
                ApplicationArea = All;
                trigger OnAction()
                begin
                end;
            }
        }
        area(Navigation)
        {
            action(Contact)
            {
                Promoted = true;
                PromotedCategory = Category7;  // PromotedActionCategories = Customer
                PromotedIsBig = true;
                Image = CustomerContact;
                ApplicationArea = All;
                trigger OnAction()
                begin
                end;
            }
            action("Account Details")
            {
```

```
            {
                Promoted = true;
                PromotedCategory = Category7;  // PromotedActionCategories = Customer
                Image = Account;
                ApplicationArea = All;
                trigger OnAction()
                begin
                end;
            }
        }
    }
}
```

For more examples of how to use actions, see Page Object and Page Extension Object.

See Also

Actions Overview
Adding Actions to a Page
AL Development Environment
Developing Extensions in AL
Pages Overview

Page Background Tasks

2/17/2021 • 17 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

To improve the performance of a page, you can develop the page to run read-only computations and long processes asynchronously in background tasks. Background tasks make a page quicker to open and more responsive, faster for users to enter information. Users aren't blocked from working while waiting for the computations to finish. Typical places where you might use background tasks are on cues and pages in FactBoxes.

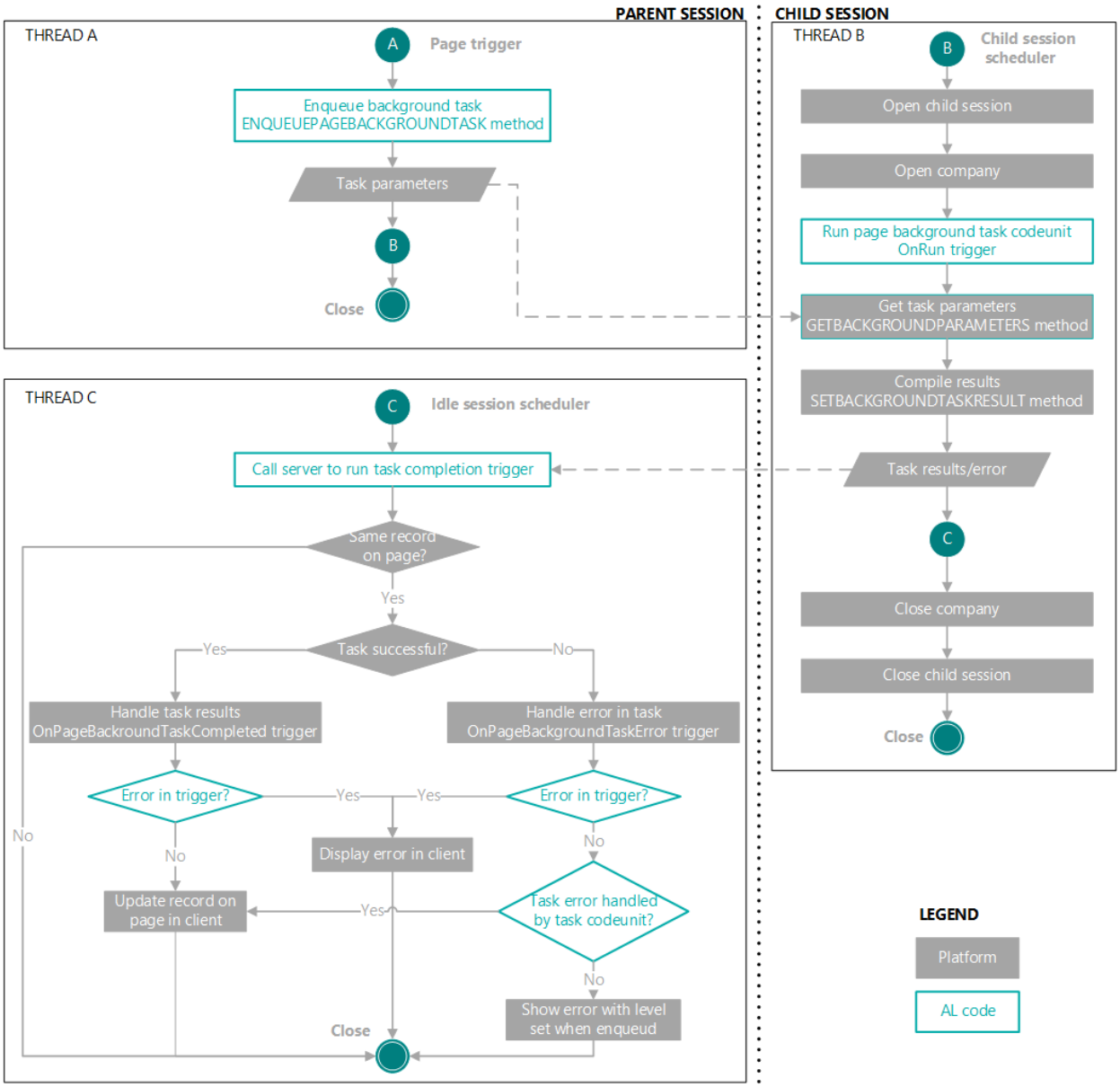
About background tasks

When a page opens in the client, a session is established between the page and the Business Central Server instance. Consider this session as the *parent session*. As a user works on the page, the user can sometimes be blocked from continuing until a process has completed. This situation is where background tasks can be beneficial.

A page background task is a *child session* that runs processes from a codeunit in the background. The user can continue to work on the page while the task runs. It's similar to other background sessions. The difference is that when the process completes, the child session is ended. The parent session is notified with results of the task. Then, the client will handle the results on the Business Central Server instance.

Background task flow

A background task is a multithread operation between the parent and child sessions. The following diagram illustrates the flow of a background task. In the illustration, the threads start in the order: THREAD A, THREAD B, THREAD C.



Background task characteristics and behavior

A page background task has the following characteristics:

- Does read-only operations; it can't write to or lock the database.
- Runs on the same Business Central Server instance as the parent session.
- The parameters that are passed to and returned from page background task are in the form of a `dictionary<string, string>`.
- Calls `OpenCompany` and executes in its own transaction.
- The callback triggers can't execute UI operations, except notifications and control updates.
- If the calling page or session closes or the current record is changed, the background task is canceled.
- It has a default and maximum timeout, which cancels the task automatically.
- Doesn't insert session event records; it relies on the parent session event records.
- Runs isolated from the parent session. Apart from the completion and error triggers, it can't call back to the parent session.
- There's a limit on the number of background tasks per session. If there are more tasks than the threshold specified per session, then the requests are queued.
- Executed synchronously from web services.
- Not counted as part of the license calculation.

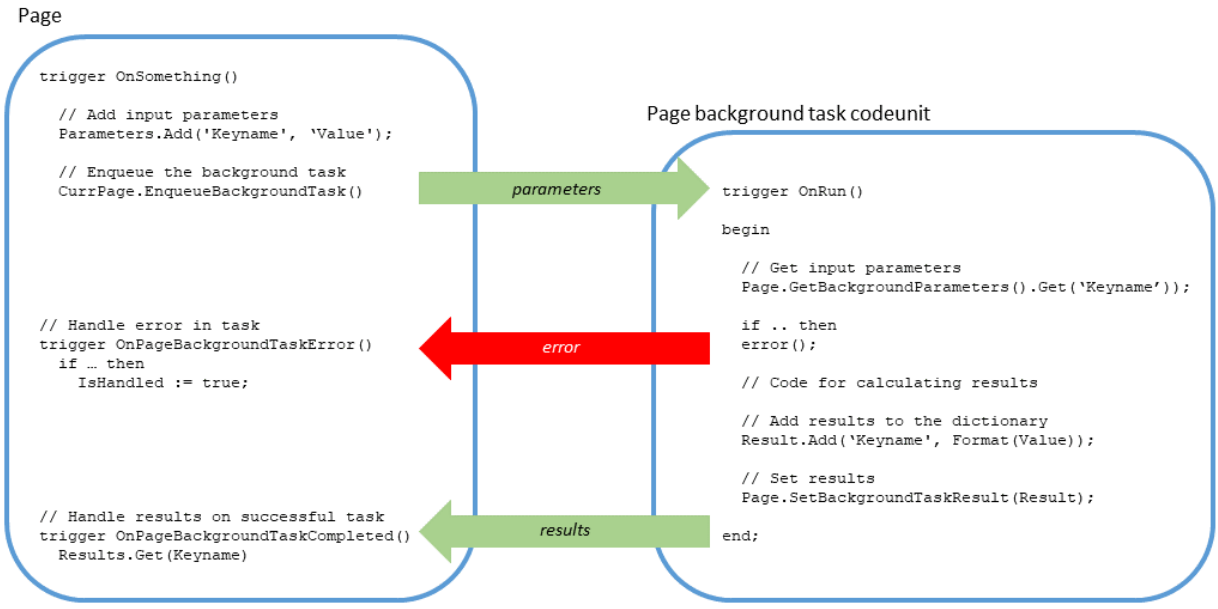
Background tasks API

The API for background tasks includes the following methods and triggers:

TYPE	NAME	DESCRIPTION
Methods	EnqueueBackgroundTask	Creates and queues a background task that runs the specified codeunit (without a UI) in a child session of the page session. If the task completes successfully, the **OnPageBackgroundTaskCompleted** trigger is invoked. If an error occurs, the **OnPageBackgroundTaskError** trigger is invoked. If the page is closed before the task completes, the task is canceled.
	GetBackgroundParameters	Gets the page background task input parameters.
	SetBackgroundTaskResult	Sets the page background task result as a dictionary. When the task is completed, the OnPageBackgroundCompleted trigger will be invoked on the page with this result dictionary
	RunPageBackgroundTask	Runs the page background task codeunit in the current session.
	CancelBackgroundTask	Attempt to cancel a page background task.
Triggers	OnPageBackgroundTaskCompleted	Runs after a page background task has successfully completed.
	OnPageBackgroundTaskError	Runs when an error occurs in a page background task.

How to create a page background task

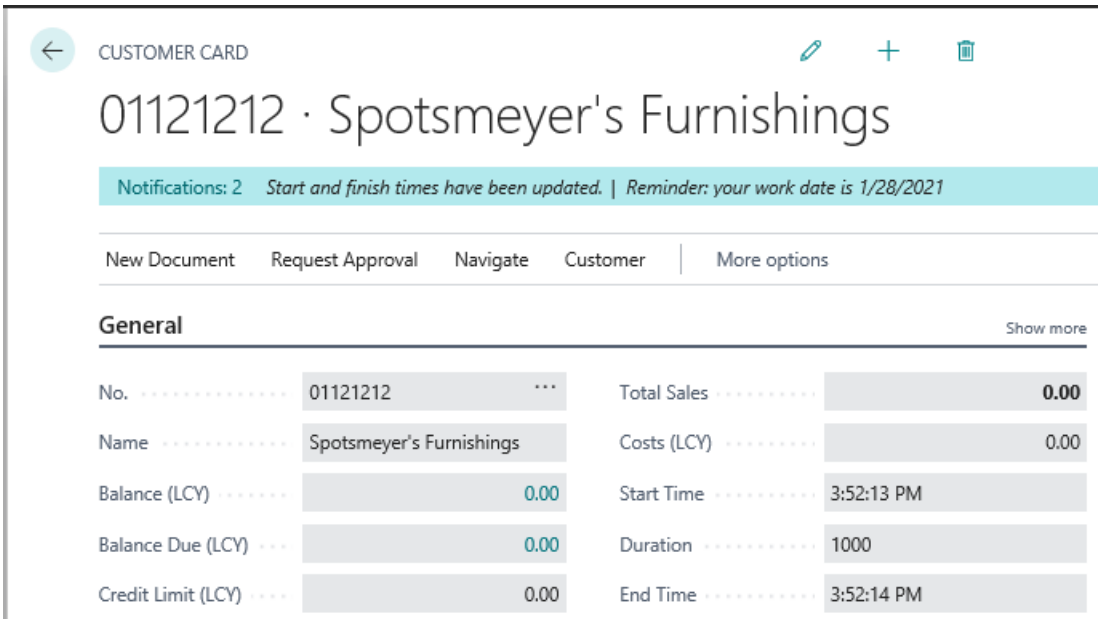
The following figure illustrates the application objects and code involved in creating a background task. The code has been simplified for demonstration purposes.



The general steps are as follows:

1. Create a background task codeunit that includes the logic that you want to run in the background.
2. On the page, complete the following steps:
   a. Add code that creates (or *enqueues*) the page background task at runtime.
   b. Add code to the OnPageBackgroundTaskCompleted trigger to handle the results of the background task and update the UI.
   c. Add code to the OnPageBackgroundTaskError trigger to handle errors that occur in the background task.

These steps are described in more details in the following sections. To help explain page background tasks, the sections use a simple example. The example extends the **Customer** card page to include a page background task. The task gets the current system time, waits a specified number of milliseconds, and gets the system time again. The page is extended with three new fields: **Start Time**, **Duration**, and **End Time**. In the page UI, these fields are updated with results of the background task, along with a notification when the task completes.



Creating a background task codeunit

You create a codeunit that does the computations that you want to run in the background. You'll also have to include code that collects the results of computations and passes them back to the calling page for handling.

The background task codeunit is a standard codeunit, which you create like any other codeunit, except it has the following characteristics:

- The OnRun() trigger is invoked without a record.
- It can't display any UI.
- It can only read from the database; not write to the database. If there's code that attempts to write to the database at runtime, an error occurs. The error informs the user that they don't have permission to do the operation on the table. For example, if the code tried to insert a record, an error similar to the following message would occur: `You don't have the following permissions on TableData 50125: Insert`.
- Casting must be manually written in code by using Format() and Evaluate() methods.

For general information about creating a codeunit, see Codeunit Object.

Getting the input parameters from the page background task

When a page background task is enqueued, it can include a set of parameters that can be used in the computations that are done in the background task codeunit. The parameters are a collection of key and value pairs. The parameters are passed as a dictionary of text to the codeunit's OnRun trigger when the page background task session starts.

To get the parameters, call the GETBACKGROUNDPARAMETERS method:

```
Parameters :=  Page.GetBackgroundParameters()
```

Use the EVALUATE method to convert the parameters to the required data type calculations.

Defining and setting the results

The results that are computed by the codeunit must be in the form of a dictionary of text. When the codeunit completes successfully, the results are passed to the parent session in a call to the `OnPageBackgroundCompleted` trigger, which will be explained later in this article.

The basic steps for defining the results are as follows:

1. Define a variable of the data type `Dictionary of [Text, Text]` for holding the results.
2. Use the Add to add key-value pairs for the results to the dictionary.
3. Call the SETBACKGROUNDTASKRESULT method to set the results in the background task.

   ```
   Page.SetBackgroundTaskResult(Result: Dictionary[Text, Text])
   ```

Example

In this example, the page background task codeunit is used to get the current system time. Then, after waiting a short period of time, it gets the system time again. The waiting period is defined by an input parameter (called `Wait`) that was passed to the background task from the parent session.

```
codeunit 50100 PBTWaitCodeunit
{
    trigger OnRun()
    var
        Result: Dictionary of [Text, Text];
        StartTime: Time;
        WaitParam: Text;
        WaitTime: Integer;
        EndTime: Time;
    begin
        if not Evaluate(WaitTime, Page.GetBackgroundParameters().Get('Wait')) then
            Error('Could not parse parameter WaitParam');

        StartTime := System.Time();
        Sleep(WaitTime);
        EndTime := System.Time();

        Result.Add('started', Format(StartTime));
        Result.Add('waited', Format(WaitTime));
        Result.Add('finished', Format(EndTime));

        Page.SetBackgroundTaskResult(Result);
    end;
}
```

Enqueuing a background task on the page

To create a page background task, you call the ENQUEUEBACKGROUNDTASK method from the page code to run the page background task codeunit. The basics steps are as follows:

1. Define any input parameters for the background task.

   The ENQUEUEBACKGROUNDTASK method can pass parameters to the background task that can be used as input to the task. Input parameters must have the data type `Dictionary of [Text, Text]`. For example, the following code defines a `Dictionary of [Text, Text]` variable and adds a single key and value pair to dictionary on the variable.

   ```
   var
       TaskParameters: Dictionary of [Text, Text];
   begin

       TaskParameters.Add('Wait', '1000');
     ...
   end
   ```

2. Define a global variable of the data type `Integer` that will be used to assign the background task an identification number.

   You don't have to assign a value to this variable. The ID is assigned automatically when the background task is enqueued.

3. Call the ENQUEUEBACKGROUNDTASK method.

   First, determine where in the code that you want to call the background task from. Typically, you call the ENQUEUEBACKGROUNDTASK method from a page trigger.

IMPORTANT

It's important that the ID of the current record of the page remains static after the ENQUEUEBACKGROUNDTASK method call is made and while the background task is running; otherwise the task will be cancelled. For this reason, we recommend that you don't enqueue the background task from the `OnOpenPage` or `OnValidate` triggers. Instead, use the `OnAfterGetCurrRecord` trigger.

Once you've determined the location, add the following code to enqueue the background task:

```
CurrPage.EnqueueBackgroundTask(TaskID, CodeunitId, Parameters, Timeout, ErrorLevel)
```

PARAMETER	DESCRIPTION	REQUIRED
`TaskId`	The variable that is defined for the task ID. This variable must be a global variable.	Yes
`CodeunitId`	The ID of the background task codeunit.	Yes
`Parameters`	The dictionary variable that is defined for the input parameters to the background task	No
`Timeout`	The number of milliseconds that the page background task can run before it's automatically canceled. You can set it to run for a maximum of 600,000 ms (10 minutes). When the task is canceled, the `OnPageBackgroundTaskError` trigger is called. By default, the task timeout is controlled by the **Page Background Task Default Timeout** and **Page Background Task Max Timeout** settings on the server instance. For more information, see Timeout. **Note:** You can add code to re-enqueue a task when an error occurs. For more information, see Re-enqueuing background tasks.	No

PARAMETER	DESCRIPTION	REQUIRED
`ErrorLevel`	The severity level for unhandled errors that occur in the background task. The severity level will determine how the errors are shown in the client UI. Values include: • `PageBackgroundTaskErrorLevel::Ignore` specifies that errors are ignored and have no effect in the client. • `PageBackgroundTaskErrorLevel::Warning` gives errors a severity level of warning. Warnings appear as a notification in the client. • `PageBackgroundTaskErrorLevel::Error` gives errors a severity level of error. Errors appear as a notification in the client. This value is the default.	No

Re-enqueuing background tasks

There are some scenarios where you want to enqueue a page background task again, after it's been initially enqueued. For example:

- You want to refresh the data on the page, like a part in a FactBox.
- You want to re-enqueue a task if it times out. When a page background task exceeds the specified timeout, the background task is canceled and an error with error code **ChildSessionTaskTimeout** occurs.

To re-enqueue a page background task, call the ENQUEUEBACKGROUNDTASK method on either the `OnPageBackgroundTaskCompleted` or `OnPageBackgroundTaskError` triggers, depending on your scenario. For detailed examples, see the PageBackgroundTask.AutoRefresh project in the BCTech GitHub repository.

> **NOTE**
>
> Use this pattern cautiously to avoid endless looping and applying excessive load on the server. Also, consider the **Child Sessions Max Queue Length** limit of the server instance. If this limit is exceeded, enqueuing will fail.

Design considerations and limitations

- The enqueued page background task stores the record ID of the current page. If the current record ID on the page changes, or the page is closed, the task is canceled.
- On list pages, it's recommended not to enqueue a page background task from `OnAfterGetRecord` trigger, unless you're aware of the consequences. If you enqueue a page background task from the `OnAfterGetRecord` , the task will be immediately canceled after the first row is retrieved. The reason is that the `OnAfterGetRecord` trigger is called on every row. Because the record changes for each row, the page background task is canceled when the trigger runs after the first row.
- By default, only five page background tasks can be run simultaneously for a parent session. If there are more than five, they're queued and run when a slot becomes available as other tasks finish. If you're using version 15.2 or later, you can increase or decrease this value by changing the **Child Sessions Max Concurrency** setting of the server instance. You can also change the **Child Sessions Max Queue Length** setting to specify the maximum number of child sessions that can be queued per parent session of a page background

task. If this value is exceeded, an error occurs. For more information, see Configuring Business Central Server - Asynchronous Processing. the Page.Rec is different.

Example

The following example extends **Customer Card** page of the base application to include three fields for displaying times that are calculated in a page background task.

```
pageextension 50100 CustomerCardExt extends "Customer Card"
{
    layout
    {
        addlast(General)
        {

            field(starttime;starttime)
            {
                ApplicationArea = All;
                Caption = 'Start Time';
                Editable = false;
            }

            field(durationtime;durationtime)
            {
                ApplicationArea = All;
                Caption = 'Duration';
                Editable = false;
            }

            field(endtime;endtime)
            {
                ApplicationArea = All;
                Caption = 'End Time';
                Editable = false;
            }
        }
    }

    var
        // Global variable used for the TaskID
        WaitTaskId: Integer;

        // Variables for the three fields on the page
        starttime: Text;
        durationtime: Text;
        endtime: Text;

    trigger OnAfterGetCurrRecord()
    var
        //Defines a variable for passing parameters to the background task
        TaskParameters: Dictionary of [Text, Text];
    begin
         TaskParameters.Add('Wait', '1000');

        CurrPage.EnqueueBackgroundTask(WaitTaskId, Codeunit::PBTWaitCodeunit, TaskParameters, 1000,
PageBackgroundTaskErrorLevel::Warning);
    end;
}
```

Coding the background task completion trigger to handle the results

When a page background task completes successfully, the `OnPageBackgroundTaskComplete` trigger of the page in the parent session is called, and the results of the task are passed to the trigger. The results are passed as a

dictionary of text. You add code to the trigger to handle the results. This operation typically includes updating the record in the page UI with the calculated values and caching the results in the database. The `OnPageBackgroundTaskComplete` trigger has the following signature:

```
trigger OnPageBackgroundTaskCompleted(TaskId: Integer; Results: Dictionary of [Text, Text])
```

PARAMETER	DESCRIPTION
`TaskId`	The ID that is assigned to the background task.
`Results`	The results of the background task

Design considerations and limitations

- Use the value of `TaskID` parameter assigned by the ENQUEUEBACKGROUNDTASK method call to identify a specific task.
- The client user must have the appropriate write permission to cache results in the database.
- Calling the UPDATE method has no effect in the trigger.
- Except for notifications, the trigger can't render UI in the client.

Example

The following example modifies the `OnPageBackgroundTaskCompleted` trigger to update the page with the started and finished times that were calculated in the page background task, and displays a notification that the times have been updated.

```
trigger OnPageBackgroundTaskCompleted(TaskId: Integer; Results: Dictionary of [Text, Text])
var
    started: Text;
    waited: Text;
    finished: Text;
    PBTNotification: Notification;
begin
    if (TaskId = WaitTaskId) then begin
        Evaluate(started, Results.Get('started'));
        Evaluate(waited, Results.Get('waited'));
        Evaluate(finished, Results.Get('finished'));

        starttime := started;
        durationtime := waited;
        endtime := finished;
        PBTNotification.Message('Start and finish times have been updated.');
        PBTNotification.Send();
    end;
end;
```

Handling errors

Within the page background task flow, errors can occur in three different locations:

- The page background task codeunit.

  > **NOTE**
  > A background task timeout will also result in an error.

- The `OnPageBackgroundTaskError` trigger of the page.

- The `OnPageBackgroundTaskCompleted` trigger of the page.

With errors that occur in the page background codeunit, you can control how the errors affect the client UI and the resultant data. For example, some errors are more severe than others. Users should be notified when an error occurs in some cases. Other times, the error can be ignored.

Errors that occur while executing the `OnPageBackgroundTaskError` or `OnPageBackgroundTaskCompleted` always display in the client with the severity level of error (`PageBackgroundTaskErrorLevel:Error`).

Handling errors that occur in the background task

When an error occurs in the page background task codeunit, the `OnPageBackgroundTaskError` trigger of the page in the parent session is automatically called with information about the error. To handle these errors, you can either use the `OnPageBackgroundTaskError` trigger as-is, that is with no custom code, or you can add custom code to the trigger to handle the errors separately.

The `OnPageBackgroundTaskError` trigger has the following signature:

```
trigger OnPageBackgroundTaskError(TaskId: Integer; ErrorCode: Text; ErrorText: Text; ErrorCallStack: Text;
var IsHandled: Boolean)
```

The following table describes the parameters of the trigger:

PARAMETER	DESCRIPTION
`TaskId`	Specifies the ID assigned to the background task.
`ErrorCode`	Specifies the error code assigned to the error that occurred in the background task, for example, `NDBCS:Deadlock` or `DB:FatalCode`. The error code is assigned by the Business Central Server instance.
`ErrorText`	Specifies the error message of the error that occurred in the background task.
`ErrorCallStack`	Specifies the error's call stack on the Business Central Server instance.
`IsHandled`	Specifies whether the error is handled. The default is `false`.

Using the `OnPageBackgroundTaskError` trigger as-is

If you want all errors that occur in the background task to be handled according to the severity level that was specified for the background task when it was enqueued, then don't add any code to the `OnPageBackgroundTaskError` trigger. In this case, the `ErrorLevel` parameter of the `ENQUEUEBACKGROUNDTASK` method call determines how the error is handled in the client.

- If the `ErrorLevel` is `PageBackgroundTaskErrorLevel:Ignore`, then error doesn't affect the client, and there's no indication on the page that the error occurred.
- If the `ErrorLevel` is `PageBackgroundTaskErrorLevel:Warning` or `PageBackgroundTaskErrorLevel:Error`, the error message displays as a notification on the page. There's currently no distinction in the notification as to whether the error is a warning or error.

For this implementation, you can either add the empty `OnPageBackgroundTaskError` trigger or omit the trigger entirely.

Customizing the OnPageBackgroundTaskError trigger

When `OnPageBackgroundTaskError` is called, it includes information about the error, such as the error code, error text, and call stack. You can add code to the trigger that handles errors based on this information. You do this by using the `IsHandled` boolean variable in your code:

- If `IsHandled` is set to `true`, the error thrown in the background task is ignored and handled by code that you add in the trigger. Here, you can add code to update the UI and associated record, similar to what can be done by `OnPageBackgroundTaskCompleted` trigger.
- If `IsHandled` is set to `false`, which is the default, the error in the background task is displayed in the client with the severity level that was specified by `ErrorLevel` parameter when the background task when was enqueued.

Example

The following example modifies the `OnPageBackgroundTaskError` trigger to display a more user-friendly notification in the client when the error `Could not parse parameter WaitParam` or timeout occurs in the page background task.

```
trigger OnPageBackgroundTaskError(TaskId: Integer; ErrorCode: Text; ErrorText: Text; ErrorCallStack: Text;
var IsHandled: Boolean)
var
    PBTErrorNotification: Notification;
begin
    if (ErrorCode = 'ChildSessionTaskTimeout') then begin
        IsHandled := true;
    PBTErrorNotification.Message('Something went wrong. The start and finish times haven''t been updated.');
    PBTErrorNotification.Send();
    end

    else if (ErrorText = 'Child Session task was terminated because of a timeout.') then begin
        IsHandled := true;
        PBTErrorNotification.Message('It took too long to get results. Try again.');
        PBTErrorNotification.Send();
    end
end;
```

Testing page background tasks

The TestPage data type includes the RUNBACKGROUNDTASK method that allows you to run unit tests for page background task codeunit. The following code is an example of a text codeunit that uses the RUNBACKGROUNDTASK method to test the page background task codeunit used in this article:

```
codeunit 50122 MyPBTCodeunit
{
    Subtype = Test;
    trigger OnRun()
    begin
        CustomerCard.OpenEdit();

        // Adds the parameters to be used as input to the background task
        TaskParameters.Add('Wait', '1000');

        // Runs the background task codeunit
        Results := CustomerCard.RunPageBackgroundTask(50100, TaskParameters);

        // Returns the results in the client
        Message('Start time: ' + '%1' + ', Duration :' + '%2' + ', Finished time: ' + '%3',
Results.Get('started'), Results.Get('waited'), Results.Get('finished'));
    end;

    var
        CustomerCard: TestPage "Customer Card";

        Results: Dictionary of [Text, Text];
        TaskParameters: Dictionary of [Text, Text];
}
```

Debugging page background tasks

Like with other code in the application, you debug page background tasks by using the integrated debugger (see Debugging). However, there are a couple limitations to be aware of:

- Only one session can be debugged at a time.
- If a breakpoint in a child session is hit, the child session will become the debugged session.
- Other sessions will continue to run normally.
- In other sessions, the breakpoints and error sessions will be ignored until the child session is completed.

Monitoring page background tasks

To monitor page background tasks, you can use the **# Active child session** performance counter and the event log.

The **# Active child session** performance counter monitors the number of active child sessions (page background tasks) on the Business Central Server instance.

In the event log, events that occur in the child session are recorded in the **Server > Admin** channel log and tagged with **Session type: ChildSession**. Events that occur in the parent session, such as in the `OnPageBackgroundTaskError` or `OnPageBackgroundTaskCompleted` triggers, are tagged with **Session type: Background**.

Designing part pages for page background tasks

Parts are a special category of page designed to be embedded within another page. Part type pages include ListPart, CardPart, and HeadlinePart. Like other page types, you can design a part page to use one or more page background tasks. However, unlike other page types, a part page won't display any data until all page background tasks have completed. This condition applies to synchronous data that's not reliant on the background tasks. In the user-interface, dashes (-) appear for field values while the page background tasks run. To work around this behavior, separate the synchronous data into a separate page part.

See Also

Configuring Business Central Server - Asynchronous Processing
Business Central Performance Counters
Monitoring Business Central Server Events
Page Parts Overview

API Page Type

2/17/2021 • 2 minutes to read • Edit Online

Pages of the type `API` are used to create versioned, webhook-supported, OData v4 enabled REST web services. This type of page cannot be displayed in the user interface, but is intended for building reliable integration services. When creating this page type, you must specify a number of properties that provide information for the web service endpoint. Use the snippet `tpage - Page of type API` to get the right template and the list of these properties automatically filled in. This page type cannot be extended by creating a page extension object. Instead, you must create a new API by adding a page object.

Pages of the type `API` can be used to develop a custom API. For more information, see Developing a Custom API.

Naming conventions

For the API page type, the following naming conventions exist:

- camelCase for naming attributes, tables, as well as APIPublisher, APIGroup, EntityName, and EntitySetName.
- Alphanumeric characters allowed (A-Z+a-z+0-9) in above elements.
- APIVersion follows the pattern vX.Y or beta.

At design time, the compiler will show warnings on casing violations and errors on naming violations. Once an API page is deployed, the corresponding $metadata is exposed on the endpoint of the page.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Example of the API page type

The following page example publishes an API available at: `../contoso/app1/v2.0/companies({id})/customers`. The `APIVersion` can be specified as one version, or a list of versions, if the API is supported through multiple versions.

```
page 50120 MyCustomerApi
{
    PageType = API;
    Caption = 'My Customer API';
    APIPublisher = 'contoso';
    APIGroup = 'app1';
    APIVersion = 'v2.0', 'v1.0';
    EntityName = 'customer';
    EntitySetName = 'customers';
    SourceTable = Customer;
    DelayedInsert = true;

    layout
    {
        area(Content)
        {
            repeater(GroupName)
            {
                field(id; Id)
                {
                    Caption = 'ID';
                }
                field(name; Name)
                {
                    Caption = 'Name';
                }
            }
        }
    }
}
```

See Also

AL Development Environment
API Query Type
Developing a Custom API
Page Extension Object
APIPublisher Property
APIGroup Property
APIVersion Property
EntityName Property
EntitySetName Property
Developing Extensions

Inspecting and Troubleshooting Pages

2/17/2021 • 5 minutes to read • Edit Online

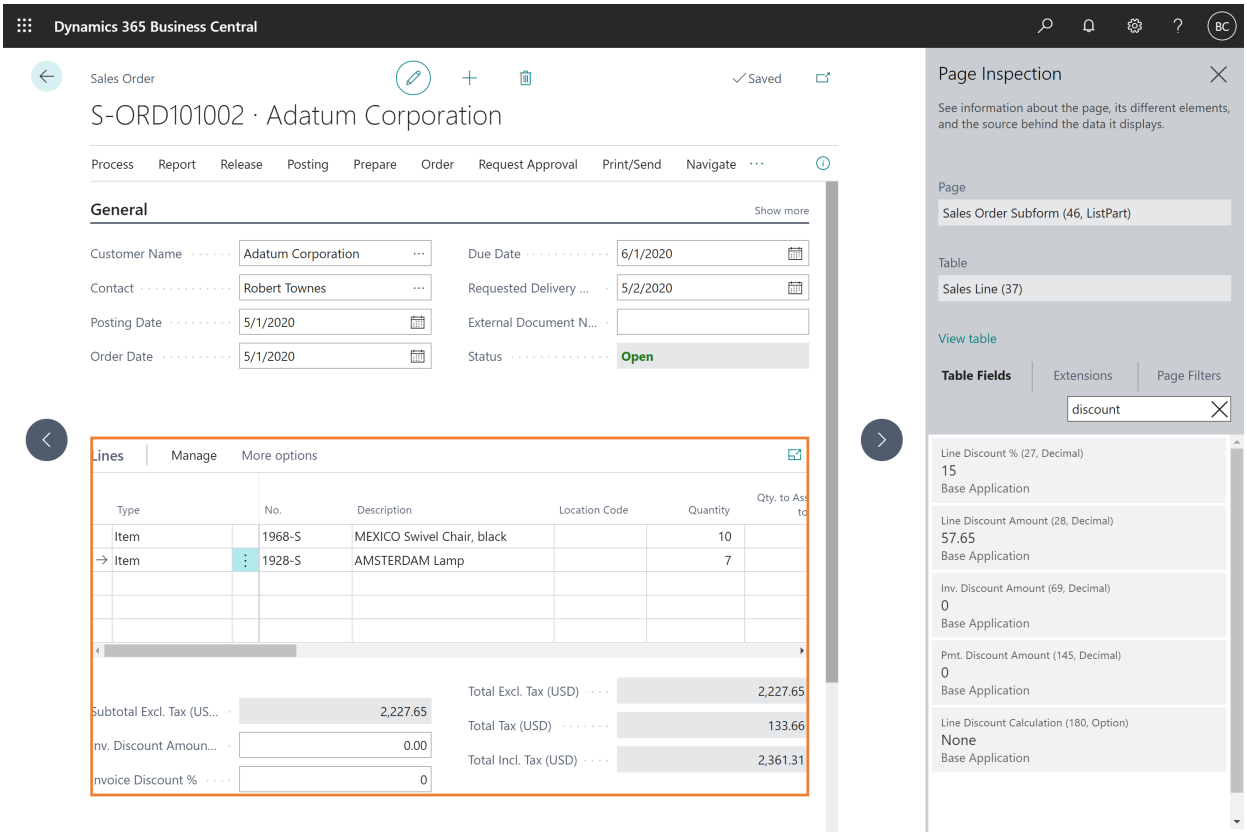
The Business Central Web client includes a page inspection feature that lets you get details about a page. Page inspection provides insight into the page design, the different elements that form the page, and the source behind the data it displays. Page inspection helps you:

- Learn the data model behind a page.
- Discover pages and parts that can be reused in your application design.
- Troubleshoot data issues without having to do tasks like copying the production database, viewing the entire source table, or digging into SQL.
- Debug the application, complementing Designer.

Working with Page Inspection

You start page inspection from the **Help & Support** page. Choose the question mark in the top-right corner, choose **Help & Support**, and then choose **Inspect pages and data**. Or, you can just use the keyboard shortcut **Ctrl+Alt+F1**.

The **Page inspection** pane opens on the side. The following figure illustrates the **Page Inspection** pane on the **Sales Order** page.



When the **Page Inspection** pane first opens, it shows information that pertains to the main page object.

Use the keyboard or pointing device to move focus to different elements on the page. When you select a FactBox or a part on the main page, a border will highlight the area. The **Page Inspection** pane then shows information about the selected element. For example, the previous figure shows information about the list part in the **Sales Order** page.

As you navigate to other pages in the application, the **Page Inspection** pane will automatically update with page information as you move along.

What Page Inspection Shows

The page inspection pane shows the information for the main page or page part, including:

- The page's source table (if any) and fields.
- Extensions that affect the page.
- Current filters applied to the page.

The following sections describe details about what is shown.

> **NOTE**
> If you do not see all details described below, you might not have the required permissions. For more information, see Controlling Access to Page Inspection Details.

> **TIP**
> To copy the values of a field or entity under one of the tabs to the clip board, select the field or entity and press Ctrl+C.

- Page
- Table
- Table Fields
- Extensions
- Page Filters

The **Page** field shows information about the main page or a selected (highlighted) subpage in a part. The field shows the following information:

- The name, as specified by its Name property
- The ID as specified by the ID property.
- The type, as specified by the PageType property.

Elements shown with limited information

- Role Center pages

  If a page has the type Role Center, the **Table** field doesn't appear. Because the Role Center consists of several parts, there's no more information shown. To see more details, select the different parts that make up the Role Center.

- Report request pages and previews

  If you open a report request page or preview for inspection, the only information shown in the Page Inspection pane is the report's name and ID.

- System parts, such as Links or Notes, and parts containing charts.

Control Add-in Style Guide

2/17/2021 • 5 minutes to read • Edit Online

This article offers a variety of stylistic definitions that are used throughout Dynamics 365, which you can apply to your control add-ins to create an experience that complements Dynamics 365.

Introduction

Control add-ins for Dynamics 365 extend a business solution by surfacing contextual functionality alongside business data. Control add-ins empower users to get more done without costly context switching, no matter which device they access Dynamics 365 from. Typical uses of control add-ins include unique data visualizations, surfacing controls from a third party service, or displaying related content from another data source.

Apart from the functionality, an important aspect of creating a control add-in is making sure the control add-in looks good and blends seamlessly into Dynamics 365. To achieve this, you should follow these basic principles:

- Apply similar patterns for command, navigation and presentation of data.
- Favor content over chrome
- Design for all platforms and input methods.
- Make it accessible to all users.
- Make it enjoyable and keep users in control.

Dynamics 365 uses a set of specific colors and fonts. You can employ these colors and fonts in your control add-ins to give it a style that matches the rest of client's user interface.

Colors

Choosing the right color gives the interface visual continuity. Color can be used to convey information to users, indicate interactivity, give feedback, and more. The following sections describe the colors used in Dynamics 365. The colors can be used on all aspects of a UI element, such background, border, text, and more.

Main colors

The following colors represent the Dynamics 365 theme main palette.

COLOR	NAME	USE	HEX VALUE
	Primary color	Prominent UI elements and areas.	#00B7C3
	Secondary color	UI elements and areas in default or subdued state.	#505C6D

Style colors

The following colors are used to express or accent conditions or user activity in the UI. For example, these colors are used as sentiments, or color indication, on Cues.

COLOR	DESCRIPTION	HEX VALUE
	Standard	#212121
	Accent	#00B7C3
	Strong	#212121
	Favorable	#35AB22
	Ambiguous	#9F9700
	Unfavorable	#EB6965
	Attention	#EB6965
	Subordinate	#A7ADB6

More palette colors

The following table includes additional colors that you can use in the UI.

COLOR	DESCRIPTION	HEX VALUE
	Yellow	#C9C472
	Green	#88CE81

COLOR	DESCRIPTION	HEX VALUE
	Red	#E97768
	Blue	#75B5E7
	Light green	#59CCB4
	Sky	75D8E7
	Egg	EEEA86
	Orange	#E89E63
	Violet	#DBBDEB
	Teal	#39B294
	Grass	#73BA5A
	Scarlet	#E65E6D

Chart colors

The following table describes the colors used in charts.

COLOR	DESCRIPTION	HEX VALUE
	-	#505C6D
	-	#008089
	Primary color	#00B7C3
	Yellow	#C9C472
	Red	#E97768
	Blue	#75B5E7
	Light green	#59CCB4
	Sky	75D8E7
	Egg	EEEA86
	Violet	#DBBDEB
	Teal	#39B294
	Grass	#73BA5A

Applying colors

To apply a color scheme to the control add-in, you specify CSS rule-sets that use the following properties:

PROPERTY	DESCRIPTION
`color`	Specifies font color.
`background-color`	Specifies background colors.
`border-color`	Specifies Border colors.

For example, to change the background of a part of your UI to use the `Secondary (#505C6D)` color, write the following CSS:

```
.my-ui-part {
    background-color: #505C6D;
}
```

If you want to change the text color of a caption to the Primary (#00B7C3) color, use the following CSS:

```
.my-caption {
    color: #00B7C3;
}
```

Typography

The main goal of typography is to provide clean and readable text in the user interface. Similar to colors, typography can also be used to convey or communicate conditions to the user.

Font Families

Dynamics 365 uses the following font families to specify the typeface and weight for text elements, such as headings, captions, messages, and so on:

EXAMPLE	NAME	VALUE
This is the Segoe UI font	Segoe UI	`"Segoe UI", "Segoe WP", Segoe, device-segoe, Tahoma, Helvetica, Arial, sans-serif`
This is the Segoe UI Light font	Segoe UI Light	`"Segoe UI Light", "Segoe WP Light", device-segoe-light, "Segoe WP Semilight", "Segoe UI", "Segoe WP", Segoe, Tahoma, Helvetica, Arial, sans-serif`
This is the Segoe UI Semi Light font	Segoe UI Semilight	`"Segoe UI Semilight", "Segoe WP Semilight", device-segoe-semilight, "Segoe UI", "Segoe WP", Segoe, Tahoma, Helvetica, Arial, sans-serif`
This is the Segoe UI Semibold font	Segoe UI Semibold	`"Segoe UI Semibold", "Segoe WP Semibold", device-segoe-semibold, "Segoe UI", "Segoe WP", Segoe, Tahoma, Helvetica, Arial, sans-serif`

Sizes

Dynamics 365 uses the following font sizes for text. The same font family on different clients may apply different sizes.

EXAMPLE	NAME	VALUE
This is 37.5pt	largest-plus-font-size	`37.5pt`
This is 30pt	largest-font-size	`30pt`
This is 22.5pt	large-plus-font-size	`22.5pt`
This is 18pt	large-font-size	`18pt`
This is 15pt	medium-plus-font-size	`15pt`
This is 13.5pt	medium-font-size	`13.5pt`
This is 12pt	small-plus-font-size	`12pt`
This is 10.5pt	small-font-size	`10.5pt`
This is 9pt	smallest-font-size	`9pt`

Applying Font Families and Sizes

To apply fonts and sizes to text elements in the UI, you need specify the following CSS properties:

- Font family. use property `font-family`.
- Font size. use property `font-size`.

For example, to change a UI element for the Web client to use the font family *Segoe UI Light* and the size *Small* (10.5pt), write the following CSS:

```
.my-ui-part {
    font-family: "Segoe UI Light", "Segoe WP Light", device-segoe-light, "Segoe WP Semilight", "Segoe UI",
"Segoe WP", Segoe, Tahoma, Helvetica, Arial, sans-serif;
    font-size: 10.5pt;
}
```

IMPORTANT

To ensure that the correct fonts are used on devices, do not omit fonts or change the order of the fonts.

Example

This examples illustrates how to use CSS to style a simple HTML UI part of a control add-in. The example includes three UI controls, as shown in the following HTML code:

```
<div class="addin">
    <div class="control">
        <div class="caption">Name:</div>
        <div class="value">
            <input type="text" name="name">
        </div>
    </div>

    <div class="control">
        <div class="caption">Surname:</div>
        <div class="value">
            <input type="text" name="name">
        </div>
    </div>

    <div class="control">
        <div class="submit">Submit</div>
    </div>
</div>
```

The following is CSS code for styling the controls, including padding, background colors, and fonts:

```
.addin {
    padding: 1em;
    background-color: #505C6D; /* Sets the background color to "Secondary" */
}

.addin .control {
     border-color: #00B7C3; /* Sets the border color to "Primary" */
}

.addin .control .caption {
    color: #00B7C3; /* Sets the captions to "Primary" */

    /* Segoe UI Light, small */
    font-family: "Segoe UI Light", "Segoe WP Light", device-segoe-light, "Segoe WP Semilight", "Segoe UI",
"Segoe WP", Segoe, Tahoma, Helvetica, Arial, sans-serif; /* Sets the font of the caption to ""Segoe UI
Light" */
    font-size: 10.5pt;
}

.addin .control .value {
    color: #008089; /* Tertiary shade 2 */

    /* Segoe UI, medium */
    font-family: "Segoe UI", "Segoe WP", Segoe, device-segoe, Tahoma, Helvetica, Arial, sans-serif;
    font-size: 12pt;
}

.addin .control .submit {
    color: white; /* Sets the caption text to "white" */
    background-color: #00B7C3; /* Sets the background to "Primary */

    /* Segoe UI Semibold, medium */
    font-family: "Segoe UI Semibold", "Segoe WP Semibold", device-segoe-semibold, "Segoe UI", "Segoe WP",
Segoe, Tahoma, Helvetica, Arial, sans-serif;
    font-size: 12pt;
    text-transform: uppercase; /* Sets the caption to use all uppercase letters */
}
```

Reports Overview

2/17/2021 • 2 minutes to read • Edit Online

You can use reports to print or display information from a database. Use reports to structure and summarize information to print documents, such as invoices. For example, create a report that lists all customers and orders that have been added by each customer. Also, create a report that is automatically filled with the relevant information for an invoice.

Reports can also be used to process data without printing or displaying content. For example, use a report to automate updating all prices in an item list. It can be easier to create a report to process data instead of a codeunit to do the same processing because you can use:

- Request page functionality to select options and filters for data items, which are available in a report but are difficult to add to a codeunit. For more information, see Request Pages.
- Report data items instead of writing code to open tables and retrieve records.
- Data modeling, which is available when you design reports.

Creating reports

Creating a report involves two primary tasks. First, you create a report object and design the dataset. The dataset determines the data that is extracted or calculated from the Dynamics 365 Business Central database tables that can be used in a report. After the dataset has been designed, you design the visual layout of the report. There are two types of report layouts that you can create: layouts using report definition language (RDL) and Word report layouts.

Getting started

The following table includes links to help you get started with designing the reports.

TO	SEE
Learn the overview of the report design process	Report Design Overview
Understand the report structure and designing the layout for a report.	Report Object
Understanding the data model and dataset of a report	Defining a Report Dataset
Learn how to create a report using a Word layout	Creating a Word Layout Report
Learn how to create a report using an RDL layout report.	Creating an RDL Layout Report

See Also

Report Object
Creating a Report
Request Pages
Creating an RDL Layout Report
Utilizing Read Scale-Out for Better Performance

Report Design Overview

2/17/2021 • 2 minutes to read • Edit Online

A report is composed of the following items:

- A report object
- A report dataset
- A report layout
- A request page
- Properties, triggers, and code

You design a report by first defining the dataset, and then designing the visual layout.

Report object

You create a report object in the AL Language development environment to define the data model, or dataset of a report. You can structure and summarize information in a report and print documents, such as sales quotes and invoices. For more information, see Report Object.

Report dataset

In order to define the underlying data model, you use the report dataset. A report dataset determines the data that is extracted or calculated from the Dynamics 365 Business Central database tables that can be used in a report. You build the report dataset by adding data items and columns. For more information, see Report Dataset.

Report layouts

The visual layout determines the content and format of a report when it is viewed and printed. You build the layout of a report by arranging data items and columns and specifying the general format, such as text font and size. A report that is viewed, printed, or saved from a Dynamics 365 Business Central client must have a report layout. There are two types of report layouts: layouts using report definition language (RDL) and Word layouts.

RDL layout

To create an RDL layout report, you use Visual Studio Report Designer or Microsoft SQL Server Reporting Services Report Builder. For more information, see Creating an RDL Layout Report.

> **IMPORTANT**
>
> RDL layouts can result in slower performance with document reports, regarding actions that are related to the user interface (for example. like sending emails) compared to Word layouts. When developing layouts for document reports, we recommend that you design Word layouts instead of RDL. With Word layouts, reports are not impacted by the security constraints on sandbox appdomains like they are with RDL layouts. From a service perspective, RDL layouts are not trusted, so they will run in a sandbox appdomain that only lives for the current report invocation.

Word report layout

You create Word layouts by using a Word Document. Word layouts are based on a Word document that includes a custom XML parts that represents the report dataset. For more information, see Creating a Word Layout Report.

See Also

Reports
Report Object
Report Data Type
Creating an RDL Layout Report
Creating a Word Layout Report
Request Pages

Report Object

2/17/2021 • 3 minutes to read • Edit Online

Reports are used to print or display information from a database. You can use a report to structure and summarize information, and to print documents, such as sales quotes and invoices.

Creating a report consists of two primary tasks; the first task is to create the underlying data model and the next is to define the visual layout that displays the data. The report object defines the underlying data model and specifies which database tables and fields to pull data from. When the report is run, that data is displayed in a specified layout; the visual layout, which determines the content and format of a report when it is viewed and printed.

For more information about defining database tables and fields, see Defining a Report Dataset. For more information about the Report data type, see Report Data Type.

You build the layout of a report by arranging data items and columns, and specifying the general format, such as text font and size. There are two types of report layouts; client report definition, also called RDL layouts and Word layouts. RDL layouts are defined in Visual Studio Report Designer or Microsoft SQL Server Reporting Services Report Builder. Word layouts are created using Word. Word layouts are based on a Word document that includes a custom XML part representing the report dataset.

> **NOTE**
> Extension objects can have a name with a maximum length of 30 characters.

Snippet support

Typing the shortcut `treport` will create the basic layout for a report object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Report example

The following example is a report that prints the list of customers. The report object defines a dataset of columns from the Customer table. For more information on creating a Word Layout report, see Creating a Report.

```
report 50103 "Customer List"
{
  CaptionML=ENU='Customer List';
  DefaultLayout = RDLC; // if Word use WordLayout property
  RDLCLayout = 'MyRDLReport.rdl';

  dataset
  {
    dataitem(Customer;Customer)
    {
      RequestFilterFields="No.","Search Name","Customer Posting Group";
```

```
        column(COMPANYNAME;COMPANYNAME)
        {
        }
        column(CurrReport_PAGENO;Customer."no.")
        {
        }
        column(Customer_TABLECAPTION_CustFilter;TABLECAPTION + ': ' + CustFilter)
        {
        }
        column(CustFilter;CustFilter)
        {
        }
        column(Customer_No;"No.")
        {
        }
        column(Customer_Customer_Posting_Group;"Customer Posting Group")
        {
        }
        column(Customer_Customer_Disc_Group;"Customer Disc. Group")
        {
        }
        column(Customer_Invoice_Disc_Code;"Invoice Disc. Code")
        {
        }
        column(Customer_Customer_Price_Group;"Customer Price Group")
        {
        }
        column(Customer_Fin_Charge_Terms_Code;"Fin. Charge Terms Code")
        {
        }
        column(Customer_Payment_Terms_Code;"Payment Terms Code")
        {
        }
        column(Customer_Salesperson_Code;"Salesperson Code")
        {
        }
        column(Customer_Currency_Code;"Currency Code")
        {
        }
        column(Customer_Credit_Limit_LCY;"Credit Limit (LCY)")
        {
          DecimalPlaces=0:0;
        }
        column(Customer_Balance_LCY;"Balance (LCY)")
        {
        }
        column(CustAddr_1;CustAddr[1])
        {
        }
        column(CustAddr_2;CustAddr[2])
        {
        }
        column(CustAddr_3;CustAddr[3])
        {
        }
        column(CustAddr_4;CustAddr[4])
        {
        }
        column(CustAddr_5;CustAddr[5])
        {
        }
        column(Customer_Contact;Contact)
        {
        }
        column(Customer_Phone_No;"Phone No.")
        {
        }
        column(CustAddr_6;CustAddr[6])
        {
```

```
    }
    column(CustAddr_7;CustAddr[7])
    {
    }
    column(Customer_ListCaption;Customer_ListCaptionLbl)
    {
    }
    column(CurrReport_PAGENOCaption;CurrReport_PAGENOCaptionLbl)
    {
    }
    column(Customer_NoCaption;FIELDCAPTION("No."))
    {
    }
    column(Customer_Customer_Posting_GroupCaption;Customer_Customer_Posting_GroupCaptionLbl)
    {
    }
    column(Customer_Customer_Disc_GroupCaption;Customer_Customer_Disc_GroupCaptionLbl)
    {
    }
    column(Customer_Invoice_Disc_CodeCaption;Customer_Invoice_Disc_CodeCaptionLbl)
    {
    }
    column(Customer_Customer_Price_GroupCaption;Customer_Customer_Price_GroupCaptionLbl)
    {
    }
    column(Customer_Fin_Charge_Terms_CodeCaption;FIELDCAPTION("Fin. Charge Terms Code"))
    {
    }
    column(Customer_Payment_Terms_CodeCaption;Customer_Payment_Terms_CodeCaptionLbl)
    {
    }
    column(Customer_Salesperson_CodeCaption;FIELDCAPTION("Salesperson Code"))
    {
    }
    column(Customer_Currency_CodeCaption;Customer_Currency_CodeCaptionLbl)
    {
    }
    column(Customer_Credit_Limit_LCYCaption;FIELDCAPTION("Credit Limit (LCY)"))
    {
    }
    column(Customer_Balance_LCYCaption;FIELDCAPTION("Balance (LCY)"))
    {
    }
    column(Customer_ContactCaption;FIELDCAPTION(Contact))
    {
    }
    column(Customer_Phone_NoCaption;FIELDCAPTION("Phone No."))
    {
    }
    column(Total_LCY_Caption;Total_LCY_CaptionLbl)
    {
    }

    trigger OnAfterGetRecord();
    begin
      CalcFields("Balance (LCY)");
      FormatAddr.FormatAddr(
        CustAddr,Name,"Name 2",'',Address,"Address 2",
        City,"Post Code",County,"Country/Region Code");
    end;

  }
}

requestpage
{
  SaveValues=true;
  ContextSensitiveHelpPage = 'my-feature';
  layout
```

```
    layout
    {
    }

    actions
    {
    }
  }

  labels
  {
      LabelName = 'LabelText', Comment = 'Foo', MaxLength = 999, Locked = true;
  }

  trigger OnPreReport();
  var
    CaptionManagement : Codeunit 42;
  begin
    CustFilter := CaptionManagement.GetRecordFiltersWithCaptions(Customer);
  end;

  var
    FormatAddr : Codeunit 365;
    CustFilter : Text;
    CustAddr : ARRAY [8] OF Text[50];
    Customer_ListCaptionLbl : Label 'Customer - List';
    CurrReport_PAGENOCaptionLbl : Label 'Page';
    Customer_Customer_Posting_GroupCaptionLbl : Label 'Customer Posting Group';
    Customer_Customer_Disc_GroupCaptionLbl : Label 'Cust./Item Disc. Gr.';
    Customer_Invoice_Disc_CodeCaptionLbl : Label 'Invoice Disc. Code';
    Customer_Customer_Price_GroupCaptionLbl : Label 'Price Group Code';
    Customer_Payment_Terms_CodeCaptionLbl : Label 'Payment Terms Code';
    Customer_Currency_CodeCaptionLbl : Label 'Currency Code';
    Total_LCY_CaptionLbl : Label 'Total (LCY)';
}
```

See Also

Request Pages
Creating an RDL Layout Report
Creating a Word Layout Report
Adding Help Links from Pages, Reports, and XMLports
Page Extension Object
Page Properties
Developing Extensions
AL Development Environment

Defining a Report Dataset

2/17/2021 • 2 minutes to read • Edit Online

You use a report object in the AL Language development environment to define the data model, or dataset, of a report. The dataset determines the data that is extracted or calculated from the Dynamics 365 Business Central database tables that can be used in a report. For more information, see Report Object.

You build the report dataset from data items and columns. A data item is a table. A column can be:

- A field in a table
- A variable
- An expression
- A text constant

Typically, data items and columns correspond to fields in a table. When the report is run, each data item is iterated for all records in the underlying table. Filters are applied and the dataset is created. When a report is based on more than one table, you must set relations between the data items so that you can retrieve and organize the data.

Snippet support

Typing the shortcut `treport` will create the basic layout for a report object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Example

The following example adds the `Customer` table as the data item and the `CustomerName` and `CompanyName` as fields of a column to the report. For more information on creating a report, see Creating a Report.

```
dataset
{
    dataitem(Customer; Customer)
    {
        column(CustomerName; CustomerName)
        {
        }
        column(CompanyName; CompanyName)
        {
        }
    }
}
```

See Also

Report Object
Reports Overview

Request Pages

2/17/2021 • 4 minutes to read • Edit Online

A request page is a page that is run before the report or XMLport starts to execute. Request pages enable end users to specify options and filters for a report and an XMLport. Request pages are defined as part of designing a Report object and an XMLport object. The syntax is shown further down in this topic. You design the filters on request pages by using the following report and XMLport properties:

PROPERTY	DESCRIPTION
RequestFilterHeading Property	Sets a caption for the request page tab that is related to a report's data item or an XMLport's table element.
RequestFilterHeadingML Property	Sets the text used as a RequestFilterHeading Property for a request page tab.
RequestFilterFields Property	Specifies which fields are automatically included on the tab of the request page that is related to a report's data item or an XMLport's table element. The user can set filters on these fields.

NOTE

Request pages for XMLports are not supported by the Business Central Web client in versions prior to Dynamics 365 Business Central 2019 release wave 2. If you try to run an XMLport with a Request page from the web client in these versions, you receive an error that the XMLport page type is not supported. Alternatively, XMLport request pages do work in the Dynamics NAV Client connected to Business Central.

By default, a request page is displayed, unless the UseRequestPage is set to `false`; then the report or XMLport will start to print as soon as it is run. In this case, end users can't cancel the report or XMLport run. It is still possible to cancel the report or XMLport, but some pages may print.

By default, without having set anything else, a request page will always display the following buttons:

- Send to
- Print
- Preview
- Cancel

Additionally, you can add more options on the request page to allow the end user to filter the data displayed.

Filtering on request pages

The fields that you define as `RequestFilterFields` are shown on the request page and can be used for filtering the data before viewing or printing the report.

NOTE

Only on the Windows client, filtering is possible even if `RequestFilterFields` is not set.

Defining the `RequestFilterFields` property in the `dataitem()` part of the report code is done as illustrated in the following code example:

```
report 50103 "Customer List"
{
    CaptionML = ENU = 'Customer List';
    RDLCLayout = 'Customer List Report.rdl'; // if Word use WordLayout property
    dataset
    {
        dataitem(Customer; Customer)
        {
            RequestFilterFields = "No.", "Search Name", "Customer Posting Group";
...
```

> **NOTE**
>
> We recommend that you add fields that the end-users of the report will frequently set filters on.

For more information about the report object, see Report Object.

Defining the `RequestFilterFields` property in the `tableelement()` part of an XMLport is done in a similar way:

```
XMLport 50104 "Export Customer List"
{
    CaptionML = ENU = 'Export customer List';
    Direction = Export;
    schema
    {
        textelement(root)
        {
            XmlName = 'Root';
            tableelement(Customer; Customer)
            {
                RequestFilterFields = "No.", "Search Name", "Customer Posting Group";
...
```

For more information about the XMLport object, see XMLport Object.

By default, for every data item in the report and table element in a XMLport, a FastTab for defining filters and sorting is created on the request page. To remove a FastTab from a request page, do not define any `RequestFilterFields` for the data item or table element and set the DataItemTableView property in a report or the SourceTableView property in an XMLport to define sorting. The request page is displayed, but there is no tab for this data item or table element.

If a `DataItemTableView` or `SourceTableView` is not defined, then end users can select a sort field and sort order at runtime.

In a complex report or XMLport that uses data from several tables, the functionality may depend on a specific key and sort order. Design your reports and XMLports so that end users can't change the sort order in a way that affects their functionality.

For data items and table elements whose source table contains calculated fields, such as amounts and quantities, the **Filter totals by:** section is automatically included on the request page, which allows you to adjust various dimensions that influence calculations.

TIP

For information about how to enter filter criteria on the request page, see Filtering in the Business Central application help.

Defining a `requestpage` section

On reports, in addition to defining the filter options by setting the `RequestFilterFields` property, you can add a `requestpage` section. In this section, you can set the SaveValues property to `true` in order to save the values that the end user enters on the request page. When the report is run again, the end user will have the option to use previously defined filters. You can also add a `layout` to the request page, specifying an **Options** section to perform checks.

NOTE

With request pages that have `SaveValues = true`, users can preview the report multiple times in the client and the request page remains open. If `SaveValues = false` or omitted, a **Preview & Close** action appears on the request page. In this case, the request page closes once the report has been previewed.

```
...
requestpage
    {
        SaveValues = true;

        layout
        {
            area(content)
            {
                group(Options)
                {
                    Caption = 'Options';
                    field(PostingDate; PostingDateReq)
                    {
                        ApplicationArea = Basic, Suite;
                        Caption = 'Posting Date';
                        ToolTip = 'Specifies the posting date for the invoice(s) that the batch job creates.
This field must be filled in.';
                    }
                }
            }
        }

        trigger OnOpenPage()
        begin
            if PostingDateReq = 0D then
                PostingDateReq := WorkDate;
        end;

        var
            PostingDateReq: Date;
    }
...
```

See Also

Report Object
XMLport Object

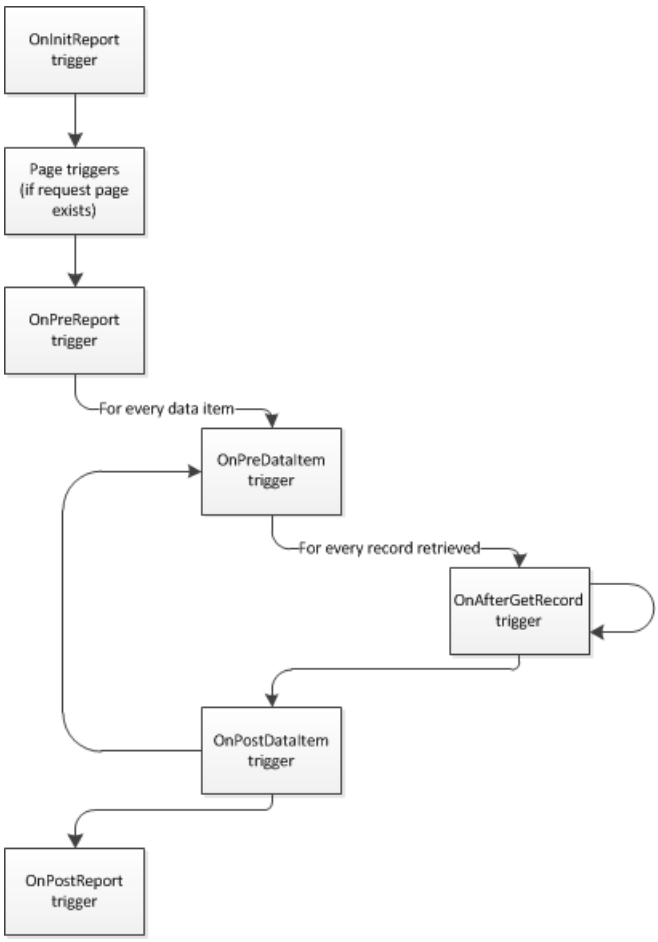
Reports Overview
Report Design Overview
RunRequestPage Method
RequestFilterHeading Property
RequestFilterHeadingML Property
RequestFilterFields Property
DataItemTableView

Report Triggers

2/17/2021 • 2 minutes to read • Edit Online

When you run a report, the report triggers are called in a specific order.

The following illustration describes the order of report triggers.



Report triggers

When you start the report run, the OnInitReport Trigger is called. If the OnInitReport doesn't end the processing of the report, then the request page for the report is run, if it's defined. The page triggers for the request page are called. On the request page, you select the options that you want for this report. You can also decide to cancel the report run. If you decide to continue, then the OnPreReport Trigger is called. At this point, no data has yet been processed. When the OnPreReport trigger has been run, the first data item is processed unless the processing of the report was ended in the OnPreReport trigger.

When the first data item has been processed, the next data item, if there's any, is processed in the same way. When there are no more data items, the OnPostReport Trigger is called to do any necessary post processing, for example, removing temporary files.

> **IMPORTANT**
>
> If you define two methods that have the same name, one defined in a report and the other in a table that is referenced by the report, you cannot invoke the defined in the report directly. By default, a call to the invokes the that is defined in the table. This behavior occurs when the is called from a source expression or a trigger.

How a data item is processed

Before the first record is retrieved, the OnPreDataItem Trigger is called, and after the last record has been processed, the OnPostDataItem Trigger is called.

Between these two triggers, the data item records are processed. Processing a record means executing the record triggers and outputting data. After a record is retrieved from the data item, the OnAfterGetRecord (Data Items) Trigger is called.

If there's an indented data item, then the records in this data item are processed.

When there are no more records to be processed in a data item, control returns to the point from which processing was started. For an indented data item, it's the next record of the data item on the next highest level. If the data item is already on the highest level, then control returns to the report.

Trigger execution with multiple previews

Business Central 2020 release wave 2 (version 17) introduced the ability to preview a report multiple times, without closing the request page. With this change, the execution flow of the report triggers is different.

When previewing a report, a new report instance is scheduled to run immediately. This new instance will rerun all of the report triggers, but won't execute any of the request page triggers.

This condition leads to the two important differences in the flow, which may influence your design:

1. The **OnInitReport** trigger is run once when opening the request page and again for every preview of the report. In some cases, it can mean that before the report is even executed, the **OnInitReport** trigger has already run twice.
2. The request page and the request page triggers won't be executed on every preview. The new report instances will receive the request page parameters from the initial request page.

For some reports, their design may make it impossible to run them as scheduled reports. It's possible to disable multiple previews and revert to the legacy preview behavior by setting the AllowScheduling property on the report object to `false`. This property was introduced in version 17.2.

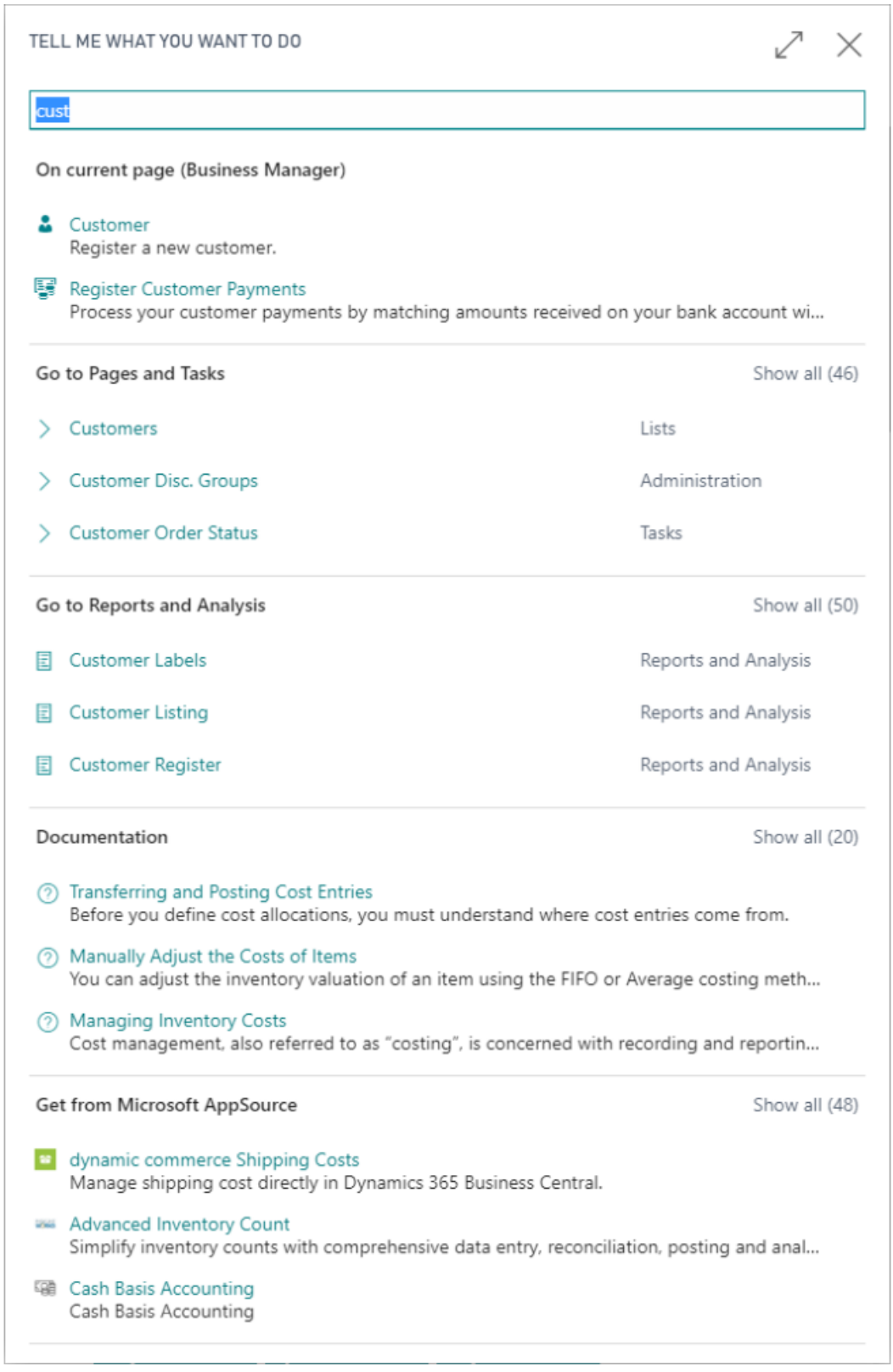
See Also

Report and DataI
Report Object
Triggers

Adding Pages and Reports to Tell me

2/17/2021 • 3 minutes to read • Edit Online

The Business Central client includes the **Tell me** feature that lets users find objects and online help articles by entering search terms. When you have added a page or a report in your extension, you most likely want it to be discoverable to users in **Tell me**. In AL, you make a page or report searchable from **Tell me** by setting the UsageCategory property in code. The **UsageCategory** setting will make the page or report searchable, and the value chosen for the setting will further sub categorize the item.



Tell me finds pages and reports by searching the captions that are specified on page and report objects by the CaptionML property.

Working with the UsageCategory property

When you create a Page or a Report, you add the UsageCategory Property. If the **UsageCategory** is set to **None**, or if you do not specify **UsageCategory**, the page or report will not show up when you search in Dynamics 365 Business Central.

UsageCategory property values

The values for the **UsageCategory** property are listed below. The sub category will help the user navigate through the search results and it is a best practice to be consistent when categorizing the pages and the reports that you add. A consistent approach will help guiding the user and improve productivity.

VALUE	DESCRIPTION
None	The page or report is not included in search.
Lists	The page or report is listed as **Lists** under the **Pages and Tasks** category.
Tasks	The page or report is listed as **Tasks** under the **Pages and Tasks** category.
ReportsAndAnalysis	The page or report is listed as **Reports and Analysis** under the **Reports and Analysis** category.
Documents	The page or report is listed as **Documents** under the **Reports and Analysis** category.
History	The page or report is listed as **Archive** under the **Reports and Analysis** category.
Administration	The page or report is listed as **Administration** under the **Pages and Tasks** category.

Adding additional search terms

You can specify other words or phrases that can help users find a page or report by using the AdditionalSearchTerms and AdditionalSearchTermsML properties. If the page or report is searchable by **Tell me** (that is, th **UsageCategory** property is set a value other than `None`), the search terms specified by these properties are used in addition to the caption of the page or report. These properties are useful when the caption does not always reflect what users will look for. A good example of this in Business Central is pages and reports associated with **Item**. Users unfamiliar with Business Central might look for 'product' or 'merchandise' instead of 'item'.

> **NOTE**
>
> For Business Central on-premises, the Business Central Web Server configuration file (navsettings.json) includes a setting called `UseAdditionalSearchTerms` that enables or disables the use of additional search terms by the **Tell me**. For more information, see Configuring Business Central Web Server Instances.

Example

The following example creates a `SimpleItemList` page and sets a `UsageCategory` property to the page, so that the `SimpleItemList` page is discoverable through search using the **Tell me** feature. Also, the example sets the `AdditionalSearchTerms` property to add two search terms for the page.

```
page 50210 SimpleItemList
{
    PageType = List;
    SourceTable = Item;
    UsageCategory = Lists;
    AccessByPermission = page SimpleItemList = X;
    ApplicationArea = All;
    AdditionalSearchTerms = 'product, merchandise';

    layout
    {
        area(content)
        {
            group(General)
            {
                field("No.";"No.") {}
                field(Name;Name) {}
                field(Description;Description) {}
            }
        }
    }
}
```

Optional settings

In addition to making a page or report searchable, you can control the access of an object by providing **Read**, **Insert**, **Modify**, **Delete**, and **Execute** (RIMDX) permissions by adding the AccessByPermission property. Likewise, control the application area access on the specified object by adding the ApplicationArea Property.

The **AccessByPermission** property and **ApplicationArea** property are the optional settings, which can be applied with the **UsageCategory** property. These settings are used to set restrictions on an object when you enable the Search functionality.

Working in the Dynamics NAV Development Environment

> **NOTE**
> Dynamics NAV Development Environment is **DISCONTINUED AFTER:** Business Central Spring 2019.

If you are using the Dynamics NAV Development Environment, you can also set **UsageCategory**, **AdditionalSearchTerms**, **AccessByPermission**, and **ApplicationArea** properties on pages and reports to control their search.

After you change these properties by using the Dynamics NAV Development Environment, before the changes take effect in the client, you must run **Build Object Search Index** from the **Tools** menu.

See Also

Adding Menus to the Navigation Pane
UsageCategory Property
Page Object
Report Object
AL Development Environment

Substituting Reports

2/17/2021 • 2 minutes to read • Edit Online

Contrary to pages and tables, extensibility is not yet supported for report objects in Business Central. Therefore, if you want to make any changes to the dataset or the layout of a base application report, you must create a new version of the report and apply the changes on the new object. Then you can override the base report with your own customized version by subscribing to the **OnAfterSubstituteReport** event published by **Codeunit 44 – ReportManagement**.

How to substitute a report for another report

To substitute a report, you create a method and subscribe it to the **OnAfterSubstituteReport** event, as shown in the code below. The `OnSubstituteReport` method replaces the report specified by the `ReportId` with the one given by the `NewReportId` parameter. In this example the `"Customer - List"` report will be substituted for `"My New Customer - List"`.

```
codeunit 50100 "Substitute Report"
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::ReportManagement, 'OnAfterSubstituteReport', '', false,
false)]
    local procedure OnSubstituteReport(ReportId: Integer; var NewReportId: Integer)
    begin
        if ReportId = Report::"Customer - List" then
            NewReportId := Report::"My New Customer - List";
    end;
}
```

For more information on how to subscribe to events, see Subscribing to Events.

When the **OnAfterSubstituteReport** event is raised, the event subscriber method is called and the substitution takes place.

> **NOTE**
> The event is called **OnAfterSubstituteReport** to match the pattern followed by other events in the **ReportManagement** codeunit, but the subscriber will be invoked before the substitution takes place.

The **OnAfterSubstituteReport** event is raised when:

1. The user activates a page action that runs the report to be substituted, that is, an action that has the RunObject Property set to the report.
2. The report is invoked from the **Tell Me** window.
3. The report is called by one of the following *static* methods:
   - Run Method
   - RunModal Method
   - SaveAsHtml Method
   - SaveAsXml Method
   - SaveAsPdf Method
   - SaveAsExcel Method

- SaveAsWord Method
- RunRequestPage Method
- Execute Method
- Print Method
- SaveAs Method

For more information about raising events, see Raising Events.

Good practices

- Consider using the same caption for both reports, given by the Caption Property. Consequently, any links and action captions that lead to the report will match the report itself. This is also relevant for bookmarks linked to a report, since they maintain the caption of the original report, even if it has been substituted for one with another caption.

- Consider enhancing the code of the subscriber method to check if the report has already been replaced with another extension. This is done by comparing the `ReportId` and `NewReportId` parameters before making the change, such that if the value of the `NewReportId` parameter is different from the value of the `ReportId` parameter and different from -1, it means that the report has already been substituted for another subscriber of the **OnAfterSubstituteReport** event.

> **IMPORTANT**
> Make sure that if a report is called on code, you use a compatible report to replace it to avoid run time errors.

See Also

Report Data Type
Subscribing to Events
Events in AL
GetSubstituteReport Method
Getting Started with AL

Testing Reports

2/17/2021 • 2 minutes to read • Edit Online

Testing your report requires you to run it and to verify the data output. This practice helps you ensure that your customers are presented with complete and accurate data.

Before extensions, the output of a report was saved to a file, but extensions deployed to Dynamics 365 Business Central cannot access the file system and therefore must save the output of a report to a stream. Codeunit 131007 `Library - Report Dataset` offers a high-level API for running and testing the output of reports that does not require direct access to the file system.

Example

The following example shows how to initialize the codeunit 131007 `Library - Report Dataset` by using the `RunReportAndLoad` method. This method is preferred as it will run the report and initialize the `Library - Report DataSet` codeunit. To verify the output, call either the `AssertElementWithValueExists` or the `AssertElementWithValueNotExist` method. The other methods in the library should work as well if they do not contain "Tag" in the name. `RUNREQUESTPAGE` and `[RequestPageHandler]` are optional and you can use them when you want to open the request page.

> **TIP**
> If you want to run the report separately and load the data from the input stream manually, you can use the `LoadDataFromInstream` method.

```
codeunit 50105 MyReportTesting
{
    Subtype = Test;

    [Test]
    [HandlerFunctions('RemittanceAdviceJournalRequestPageHandler')]
    procedure TestingReports();
    var
        XmlParameters: Text;
        LibraryReportDataset: Codeunit "Library - Report Dataset";
        GenJournalLine: Record "Gen. Journal Line";
    begin
        // Run the Report Remittance Advice - Journal.
        XmlParameters := Report.RunRequestPage(Report::"Remittance Advice - Journal");
        LibraryReportDataset.RunReportAndLoad(Report::"Remittance Advice - Journal", GenJournalLine,
XmlParameters);

        // Verifying Total Amount on Report.
        LibraryReportDataset.AssertElementWithValueExists('Amt_GenJournalLine', GenJournalLine.Amount);
    end;

    [RequestPageHandler]
    procedure RemittanceAdviceJournalRequestPageHandler(var RemittanceAdviceJournal: TestRequestPage 399);
    begin
        // Empty handler used to close the request page. We use default settings.
    end;
}
```

Any changes done in the handler above will result in the `XmlParameters` being changed and applied

automatically when the report runs. Examples of the implementation in the existing tests are in `Codeunit 133770` and `Codeunit 134141`.

Remarks

TestRequestPage.SaveAsXML uses a different format than Report.SaveAsXML or Report.SaveAs by serializing the output of **Report Previewer**. This is a component that will be deprecated in the future and replaced with the new methods that can be used for the new tests. Another difference is that `TestRequestPage.SaveAsXML` requires files to be saved to disk and loaded, while other methods work in memory, making them more efficient.

> **NOTE**
>
> The existing tests still need support and the codeunit solves this problem by supporting both formats for now. `TestRequestPage.SaveAsXML` uses Tags for values, while the new format uses attributes. This means that you cannot use any public method that contains "Tag" in the name to test the reports generated in the memory.

See Also

Reports Overview
Testing Pages
Test Codeunits and Test Methods

Creating a Word Layout Report

2/17/2021 • 2 minutes to read • Edit Online

When you create a new report, there are two things you have to think about; defining the report dataset of data items and columns, and then designing the report layout. These steps will show how to create a very simple report based on a Word layout. For more information about the report object, see Report Object.

Create a Word layout report

To facilitate testing your report layout, the following simple example extends the Customer List page with a trigger that runs the report as soon as the Customer List page is opened.

1. Create a new extension to the **Customer List** page that contains code to run the report, as well as a simple report object by adding the following lines of code:

```
pageextension 50100 MyExtension extends "Customer List"
{
    trigger OnOpenPage();
    begin
        report.Run(Report::MyWordReport);
    end;
}

report 50124 MyWordReport
{
    DefaultLayout = Word;
    WordLayout = 'MyWordReport.docx';
}
```

2. Build the extension (**Ctrl+Shift+B**) to generate the MyWordReport.docx file.
3. Add the **Customer** table as the data item and the **Name** field as a column to the report by adding the following lines of code to the report. For more information about defining a dataset, see Report Dataset.

```
report 50124 MyWordReport
{
    DefaultLayout = Word;
    WordLayout = 'MyWordReport.docx';

    dataset
    {
        dataitem(Customer; Customer)
        {
            column(Name; Name)
            {

            }
        }
    }
}
```

4. Build the extension (**Ctrl+Shift+B**).
5. Open the generated report layout file in Word.
6. In Word, edit the layout using the **XML Mapping Pane** on the **Developer** tab.

> **NOTE**
>
> If you do not see the Developer tab, go to **Options**, then **Customize Ribbon**, and in the **Main tabs** section, select the **Developer** check box.

7. In Word, to the right, in the **Custom XML part** lookup, locate the report, and then open the layout.
8. Right-click on the **Customer** table, and in **Insert Content Control**, select **Repeating** to add the repeater data item.
9. Right-click on the **Name** field and in **Insert Content Control**, select **Plain Text** to add the column as a text box.
10. Save the report layout when you are done and then close it.
11. Back in Visual Studio Code, press **Ctrl+F5** to compile and run the report.

You will now see the generated report in preview mode.

> **NOTE**
>
> If the report layout is not generated, open the `settings.json` from Visual Studio Code. Use **Ctrl+Shift+P**, then choose **Preferences: Open User Settings**, locate the **AL Language extension**. Under **Compilation Options**, choose **Edit in settings.json** and add the following line:
>
> ```
> json "al.compilationOptions": { "generateReportLayout": true }
> ```

See Also

Report Design Overview
Report Object
Creating an RDL Layout Report

Creating an RDL Layout Report

2/17/2021 • 2 minutes to read • Edit Online

When you create a new report for Dynamics 365 Business Central, there are two things you have to think about; defining the report dataset of data items and columns, and then designing the report layout. These steps will show you how to create a very simple report based on an RDL layout. For more information about the report object, see Report Object.

> **IMPORTANT**
>
> RDL layouts can result in slower performance with document reports, regarding actions that are related to the user interface (for example. like sending emails) compared to Word layouts. When developing layouts for document reports, we recommend that you design Word layouts instead of RDL. With Word layouts, reports are not impacted by the security constraints on sandbox app domains like they are with RDL layouts. From a service perspective, RDL layouts are not trusted, so they will run in a sandbox app domain that only lives for the current report invocation.

To create and modify RDL report layouts, you use SQL Server Report Builder or Visual Studio Report Designer. For information about required versions of these tools, see System Requirements.

Create an RDL layout report

To facilitate testing your report layout, the following simple example extends the Customer List page with a trigger that runs the report as soon as the Customer List page is opened.

1. Create a new extension to the Customer List page that contains code to run the report, as well as a simple report object by adding the following lines of code:

```
pageextension 50123 MyExtension extends "Customer List"
{
    trigger OnOpenPage();
    begin
        report.Run(Report::MyRdlReport);
    end;
}

report 50123 MyRdlReport
{
    DefaultLayout = RDLC;
    RDLCLayout = 'MyRDLReport.rdl';

}
```

2. Build the extension (**Ctrl+Shift+B**) to generate the MyRDLReport.rdl file.

3. Add the **Customer** table as the data item and the **Name** field as a column to the report by adding the following lines of code to the report:

```
report 50123 MyRdlReport
{
    DefaultLayout = RDLC;
    RDLCLayout = 'MyRDLReport.rdl';

    dataset
    {
        dataitem(Customer; Customer)
        {
            column(Name; Name)
            {
            }
        }
    }
}
```

4. Build the extension (**Ctrl+Shift+B**). The `MyRDLReport.rdl` file will be created in the root of the current project.
5. Open the generated report layout file in **Microsoft SQL Server Report Builder**.
6. Edit the layout by inserting a table.
7. Add the **Name** column from the **Datasets** folder into the table and save the .rdl file.
8. Back in Visual Studio Code, press **Ctrl+F5** to compile and run the report in Dynamics 365 Business Central.

You will now see the generated report in preview mode.

> **NOTE**
>
> If the report layout is not generated, open the `settings.json` from Visual Studio Code. Use **Ctrl+Shift+P**, then choose **Preferences: Open User Settings**, locate the **AL Language extension**. Under **Compilation Options**, choose **Edit in settings.json** and add the following line:
>
> ```
> "al.compilationOptions": {
>         "generateReportLayout": true
>     }
> ```

See Also

Report Design Overview
Report Object
Creating a Word Layout Report

Walkthrough: Designing a Report from Multiple Tables

2/17/2021 • 22 minutes to read • Edit Online

A report object is composed of a report dataset and a visual layout. You design a report by first defining the dataset and then designing the visual layout. You define the dataset for reports directly in AL code. You design the layout in Visual Studio Report Designer or Microsoft SQL Server Reporting Services Report Builder for a RDL layout and in Microsoft Word for a Word layout. After you design a report, you can make it available to applications that are running on the Business Central Web client. A report can be designed from one table or multiple tables. This walkthrough demonstrates how to design a report from multiple tables.

About This Walkthrough

This walkthrough shows you how to design a report from the AL Language development environment and using Visual Studio Report Designer for designing an RDL layout.

This walkthrough illustrates the following tasks:

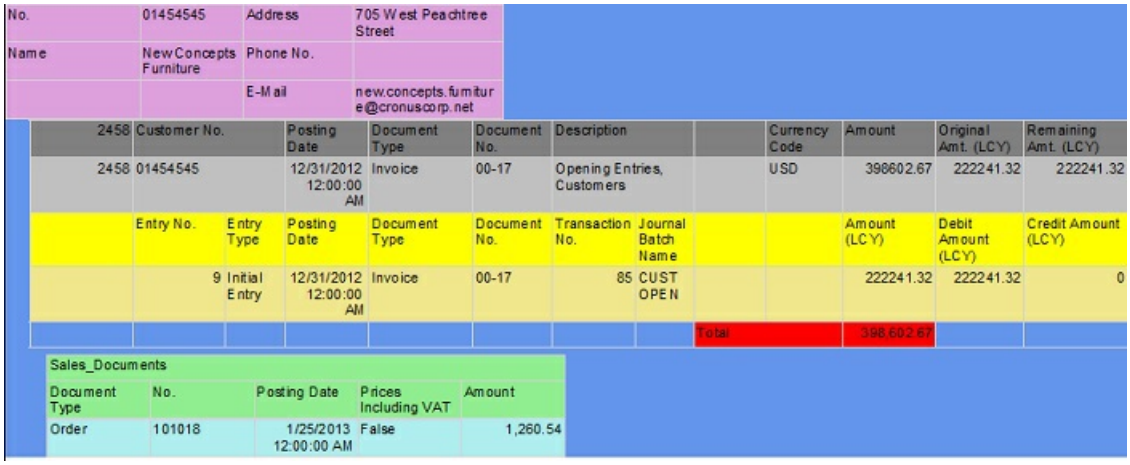
- Defining the dataset for multiple tables.
- Adding fields to a data item.
- Defining properties for the data items.
- Adding labels to a report.
- Design a client report definition (RDL) report layout in Visual Studio 2019.
- Setting filters to hide empty rows and fields in a report.
- Building and running a report.

Story

Viktor is a developer who is working for CRONUS International Ltd. Viktor has been asked by his manager to create a report that shows data from the `Customer` (ID 18), `Cust. Ledger Entry` (ID 21), `Detailed Cust. Ledger Entry` (ID 379), and the `Sales Header` (ID 36) tables. The report should meet the following requirements:

- The report must display customer information at the top of the report.
- For each customer, the report must show a list of ledger entries.
- For each ledger entry, the report must show a list of detailed ledger entries under the ledger entries.
- The report must display basic sales document headers information for the selected customer.
- Each section of the data for each customer must begin on a new page.
- The `Amount` field from the `Cust. Ledger Entry` table should be totaled and displayed for each customer.
- If there are no records to display, the report must not display that data sections. For example, if there are no sales documents for a customer, the sale header section must be skipped.
- Amount fields must not display zero values.
- The orientation of the report should be landscape.

The following illustration shows an example of the second page of the report.



No.	01454545	Address	705 West Peachtree Street
Name	New Concepts Furniture	Phone No.	
		E-Mail	new.concepts.furniture@cronuscorp.net

2458	Customer No.	Posting Date	Document Type	Document No.	Description		Currency Code	Amount	Original Amt. (LCY)	Remaining Amt. (LCY)
2458	01454545	12/31/2012 12:00:00 AM	Invoice	00-17	Opening Entries, Customers		USD	398602.67	222241.32	222241.32

	Entry No.	Entry Type	Posting Date	Document Type	Document No.	Transaction No.	Journal Batch Name			Amount (LCY)	Debit Amount (LCY)	Credit Amount (LCY)
	9	Initial Entry	12/31/2012 12:00:00 AM	Invoice	00-17	85	CUST OPEN			222241.32	222241.32	0
								Total		398,602.67		

Sales_Documents

Document Type	No.	Posting Date	Prices Including VAT	Amount
Order	101018	1/25/2013 12:00:00 AM	False	1,260.54

Defining the Dataset

Viktor starts by creating an empty report object by using the AL Language extension in Visual Studio Code. You can use the shortcut `treport` to create the basic layout for a report object.

He sets the DefaultLayout Property to **RDLC** to specify that he will use an RDL layout for the report and the RDLCLayout Property to `'MyRDLReport.rdl'`, the name of the rdl file he will use for the layout.

Viktor will now design the dataset to display customers and their transaction details. This is defined within the `dataset` part of the report object.

Adding Data Items and columns

The datasets for the data model will come from four tables: `Customer`, `Cust. Ledger Entry`, `Detailed Cust. Ledger Entry`, and `Sales Header`. Viktor will create a data item for each table with the `dataitem` control. Moreover, for each table, he will add the fields that he wants to display on the report. Each field is given by a `column` control, defined inside the corresponding data item.

The hierarchy of the `dataitem` and `column` controls is important because it will determine the sequence in which data items are linked, which in turn will control the results. Working from top-to-bottom, you start by adding the `dataitem` control for first table that you want in the dataset, then add column controls for each table field that you want to include in the dataset. For the next table, you add another `dataitem` control that is embedded within the first `dataitem` control, then add column controls as needed. You continue this pattern for additional tables and fields.

Defining Properties for the Data Items

Once Viktor has specified the dataitem and column elements he will define the appropriate properties. He sets the DataItemTableView Property in each data item to sort the table view based on a specific field.

He also sets the RequestFilterFields Property to automatically include a specific field on the filter tab of the request page. For more information about request pages, see Request Pages.

Now, Viktor uses the DataItemLink (Reports) Property to set a link between one or more fields of the dataitem tables. Links determine which records to include in the dataset based on the values of a common field between dataitems. This property must be set on the lower dataitem of the report object.

For each of the `column` controls he adds the IncludeCaption Property and sets it to **True**. This property specifies to include the caption of the fields in the dataset of a report.

Finally, he sets the PrintOnlyIfDetail Property to **True** on a data item to print data only if at least one of its child data items generates output.

Adding Labels to the Report

Viktor will now add labels to the report. You define the labels in the `label` part of the report. These labels will be used later as captions.

The following code exemplifies the code that Viktor has written for the report.

```
report 50101 "Report for Multiple Tables"
{
    //Make the report searchable from Tell me under the Administration category.
    UsageCategory = Administration;
    ApplicationArea = All;
    // Use an RDL layout.
    DefaultLayout = RDLC;
    // Specify the name of the file that the report will use for the layout.
    RDLCLayout = 'MyRDLReport.rdl';

    dataset
    {

        dataitem(Customer; Customer)
        {
            // Sort the table view based on the "No." field.
            DataItemTableView = Sorting("No.");
            // Include the "No." field on the filter tab of the request page.
            RequestFilterFields = "No.";
            // Print data only if at least one of the CustLedgerEntry and SalesHeader data items generates
output.
            PrintOnlyIfDetail = True;

            // For each field that you want to display you add a column control.
            column(No_Customer; "No.")
            {
                // Include the caption of the "No." field in the dataset of the report.
                IncludeCaption = true;
            }

            column(Name_Customer; Name)
            {
                IncludeCaption = true;

            }

            column(Phone_Customer; "Phone No.")
            {
                IncludeCaption = true;

            }

            column(Address_Customer; Address)
            {
                IncludeCaption = true;

            }

            column(EMail_Customer; "E-Mail")
            {
                IncludeCaption = true;

            }
            dataitem(CustLedger; "Cust. Ledger Entry")
            {

                DataItemTableView = sorting("Entry no.");
                // Set a filter on the child data item, **CustLedgerEntry** to select only the records where
the
                // value of `Customer."No."` field and the `"Customer Ledger Entry"."Customer No."` field
```

```
matches.
                DataItemLink = "Customer No." = field("No.");

                column(EntryNo_CustLedgerEntry; "Entry No.")
                {
                    IncludeCaption = true;

                }
                column(CustomerNo_CustLedgerEntry; "Customer No.")
                {
                    IncludeCaption = true;

                }
                column(PostingDate_CustLedgerEntry; "Posting Date")
                {
                    IncludeCaption = true;

                }
                column(DocumentType_CustLedgerEntry; "Document Type")
                {
                    IncludeCaption = true;

                }

                column(DocumentNo_CustLedgerEntry; "Document No.")
                {
                    IncludeCaption = true;

                }

                column(Description_CustLedgerEntry; Description)
                {
                    IncludeCaption = true;

                }

                column(CurrencyCode_CustLedgerEntry; "Currency Code")
                {
                    IncludeCaption = true;

                }

                column(Amount_CustLedgerEntry; Amount)
                {
                    IncludeCaption = true;

                }

                column(OriginalAmtLCY_CustLedgerEntry; "Original Amt. (LCY)")
                {
                    IncludeCaption = true;

                }

                column(RemainingAmtLCY_CustLedgEntry; "Remaining Amt. (LCY)")
                {
                    IncludeCaption = true;

                }


                dataitem(DetCustLedger; "Detailed Cust. Ledg. Entry")
                {

                    DataItemTableView = sorting("entry no.");
                    DataItemLink = "Cust. Ledger Entry No." = field("Entry No."), "Customer No." =
field("Customer No.");

                    column(EntryNo_DetailedCustLedgEntry; "Entry No.")
```

```
            {
                IncludeCaption = true;

            }

            column(EntryType_DetailedCustLedgEntry; "Entry Type")
            {
                IncludeCaption = true;

            }

            column(PostingDate_DetailedCustLedgEntry; "Posting Date")
            {
                IncludeCaption = true;

            }

            column(DocumentType_DetailedCustLedgEntry; "Document Type")
            {
                IncludeCaption = true;

            }

            column(DocumentNo_DetailedCustLedgEntry; "Document No.")
            {
                IncludeCaption = true;

            }

            column(AmountLCY_DetailedCustLedgEntry; "Amount (LCY)")
            {
                IncludeCaption = true;

            }

            column(TransactionNo_DetailedCustLedgEntry; "Transaction No.")
            {
                IncludeCaption = true;

            }

            column(JournalBatchName_DetailedCustLedgEntry; "Journal Batch Name")
            {
                IncludeCaption = true;

            }

            column(DebitAmountLCY_DetailedCustLedgEntry; "Debit Amount (LCY)")
            {
                IncludeCaption = true;

            }

            column(CreditAmountLCY_DetailedCustLedgEntry; "Credit Amount (LCY)")
            {
                IncludeCaption = true;

            }

        }
    }

    dataitem(SalesHeader; "Sales Header")
    {
        DataItemTableView = sorting("Document Type", "No.");
        DataItemLink = "Sell-to Customer No." = field("No.");

        column(DocumentType_SalesHeader; "Document Type")
        {
```

```
                {
                    IncludeCaption = true;

                }

                column(No_SalesHeader; "No.")
                {
                    IncludeCaption = true;

                }

                column(PostingDate_SalesHeader; "Posting Date")
                {
                    IncludeCaption = true;

                }

                column(PricesIncludingVAT_SalesHeader; "Prices Including VAT")
                {
                    IncludeCaption = true;

                }

                column(Amount_SalesHeader; Amount)
                {
                    IncludeCaption = true;

                }
            }

        }
    }

    // These labels will be used later as captions in the report layout.
    labels
    {
        Sales_Document_Caption = 'Sales Documents';
        Total_Caption = 'Total';
    }
}
```

Designing the Visual RDL Layout for the Report

Next, Viktor will design an RDL layout for the report by using Visual Studio Report Designer. He will set properties for the report and the report elements, format the report, and then add the data to the report.

To design the RDL layout for the report

1. Build the extension (Ctrl+Shift+B) to generate the `MyRDLReport.rdl` file, and then open the file with Visual Studio 2019.
2. Right-click anywhere outside the report (in the shaded area) and then choose **Add Page Header**.
3. Right-click anywhere outside the report (in the shaded area), and then choose **Report Properties**.
4. In the **Report Properties** window, choose the **Page Setup** tab. In the **Paper size** section, under **Orientation**, choose **Landscape**, and then choose the **OK** button.
5. On the **View** menu, choose **Toolbox**. Select the **List** control, and then choose the body of the report to add the **List** control to the report. This control will contain and group all the data.
6. Move the **List control** to the top of the report body and resize it to cover the whole report body.
7. Right-click the middle of the **List** controls, and then click on the box to open the **Rectangle Properties**.
8. In the **Rectangle Properties** window, choose the **Fill** tab, in the **Fill Color** list color pallet, select

Cornflower Blue from the color pallet, and then choose the **OK** button. You can choose any color.

> **NOTE**
> Changing the color of report elements helps you identify elements on the report preview. You can set different color properties for table header, detail rows, text boxes, and so on.

Viktor will set the properties of the **List** control to hold the dataset, group the data by Customer No. and set up how the groups should be displayed.

To set the list control properties

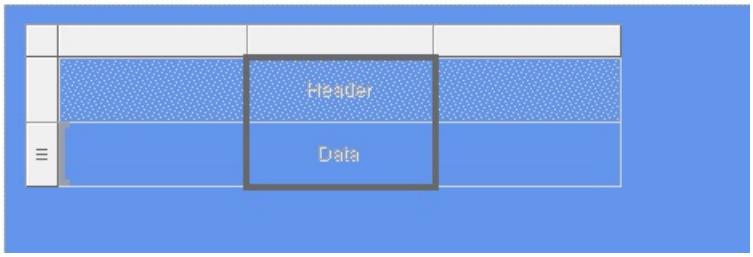
1. Select the **List** control, right-click the shaded border to the left of the **List** control, and then choose **Tablix Properties**.
2. In the **Tablix Properties** window, on the **General** tab, under **Dataset name**, select **DataSet_Result** from the drop-down list, and then choose the **OK** button.
3. Select the **List** control, right-click the shaded border to the left of the list control, choose **Row Group**, and then **Group Properties**.
4. In the **Group Properties** window, on the **General** tab, under **Group expressions:**, choose the **Add** button, and then select **[No_Customer]** from the **Group on:** drop-down list. This groups all the data in the List control by customer number.
5. On the **Page Breaks** tab, select **Between each instance of a group**, and then choose the **OK** button.

Viktor is now ready to add the customer data. The table will display one customer at a time, therefore Viktor must put all the fields into table header rows. The table data and footer rows will be disabled.

To add customer data

1. From the **Toolbox** pane, drag a **Table** control into the **List** control and resize the table to about the half the width of the list control. This table will contain the customer data.

   The following illustration shows the list control and the table.

   Note that the table contains two table rows, a header row (first row), and a data row (second row). The three parallel lines in the left border of the second row identify the data row.
2. Select any table row, right-click the shaded border, and then choose **Tablix Properties** to open the **Tablix Properties** window.
3. On the **General** tab, verify that the **Dataset name** field is set to **DataSet_Result**, and then choose the **OK** button.

   The table has three columns. Viktor will add a fourth column to the table to hold all the customer data.
4. Right-click the middle column header, choose **Insert Column**, and then select **Right** to insert the fourth column into the table.
5. Select the second table row (the data row), right-click the row, choose **Delete Rows** to delete the data row, and then choose the **OK** button in the **Delete Rows** window to delete the row and its associated

groups.

6. Select the remaining table row, right-click the shaded border on the left, choose **Insert Row**, and then choose **Below** to insert another table header row.
7. Repeat step 6 to insert a third table header row. There should now be three header rows in the table.
8. Right-click the first cell (row 1, column 1) in the table, and then choose **Expression** to open the **Expression** window.
9. In the **Category** column, select **Parameters**, in the **Item** column, verify that **All** is selected, and then in the **Values** column, double-click **No_CustomerCaption**. Verify that the **Set expression for: Value** box contains the following value: `=Parameters!No_CustomerCaption.Value`. This cell will display the customer No. caption in the report.
10. Modify the expression to `=First(Parameters!No_CustomerCaption.Value)`. Choose the **OK** button.

> **NOTE**
> All caption fields must begin with `=First` so that the first value for the caption fields in the data set is retrieved and used as caption. If the First function is not used, the report will return the current value for a field. The current value however may be incorrect. For example, the current value could be empty.

11. Right-click the second cell (row 1, column 2) in the table, and then choose **Expression** to open the **Expression** window.
12. In the **Category** column, select **Field(DataSet_Result)**, in the **Item** column verify that **All** is selected, and then in the **Values** column double-click **No_Customer**. Verify that the **Set expression for: Value** box contains the following value `=Fields!No_Customer.Value`. Choose the **OK** button. This cell will display the Customer No..
13. Repeat steps 8 through 12 to the enter captions and values in the following cells.

> **NOTE**
> Columns 1 and 3 will contain the captions and columns 2 and 4 will contain the values.

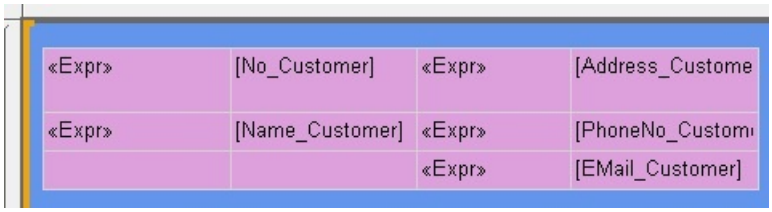
ROW	COLUMN	CAPTION	VALUE
2	1	Name_CustomerCaption	None
2	2	None	Name_Customer
1	3	Address_CustomerCaption	None
1	4	None	Address_Customer
2	3	PhoneNo_CustomerCaption	None
2	4	None	PhoneNo_Customer
3	3	Email_CustomerCaption	None

ROW	COLUMN	CAPTION	VALUE
3	4	None	Email_Customer

14. Select all table rows (not the whole table), and then on the **View** menu, choose **Properties Window** to open the **Properties** window in Visual Studio.

15. In the **Properties** window, under **Fill**, set the **BackgroundColor** property to **Plum**. You can choose any color.

    The layout that Viktor has designed to this point resembles the following illustration.



16. On the **Build** menu, choose **Build Web site** to build the project. Inspect the **Output** pane and make sure that there are no build errors. Close Visual Studio.

    > **NOTE**
    > It is a good practice to build the project periodically during the report design to make sure that there are no build errors.

    Viktor will run the report and preview what he has done to this point.

17. Go back to your project in Visual Studio Code and Reload the Window.

18. In the `launch.json` file set the `"startupObjectId"` to the **Id** of the report object and the `"startupObjectType"` to `Report`.

19. Press the `F5` key to run the report.

20. In the request page that is displayed, choose the **Preview** button to view the report. The first customer is displayed on the first page. If you page through the report, each customer is displayed on a separate page.

Viktor will now add the data for the customer ledger entries and detailed ledger entries. The entries will be put in a different table.

To add the data for ledger entry and detailed ledger entry

1. Open the `MyRDLReport.rdl` report in Microsoft Visual Studio.

2. From the **Toolbox**, drag a table control into the list control. Put the table under the table that contains the customer data.

   > **NOTE**
   > You may have to resize the report body and the list controls to make them larger.

3. Select the table, right-click the shaded border, choose **Tablix Properties**. On the **General** tab, verify that the **Dataset name** field is set to **DataSet_Result**, and then choose the **OK** button.

4. Select the table data row, choose **Insert Row** and then choose **Outside Group – Below**. This adds

another data row to the table. You now have one header row and two data rows.

5. Delete the first row (header row) in the table and then insert columns in the table so that the total number of columns is eleven.
6. Choose the first data row, right-click the shaded border to the left, choose **Add Group**, and then choose **Parent Group**.
7. In the **Tablix group** window, select **Group by**, select **EntryNo_CustLedgerEntry** from the drop-down list. Select **Add group header**, and then choose the **OK** button.
8. Right-click the first row, choose **Insert Row**, and then choose **Inside Group – Above**. This header will hold the captions for the Customer Ledger entries.
9. Right-click the cell in the row1, column 2, and then choose **Expression** to open the **Expression** window.
10. In the **Category** column, select **Parameters** and then in the **Values** column double-click **EntryNo_CustLedgerEntryCaption**. Note that the **Set expression for: Value** box contains the following value: `=Parameters!EntryNo_CustLedgerEntryCaption.Value`
11. Modify the expression to the following value: `=First(Parameters!EntryNo_CustLedgerEntryCaption.Value)`.
12. Repeat steps 9 through 11 to add captions for the table cells in the rest of the first row as shown in the following table.

COLUMN	CAPTION EXPRESSION
3	CustomerNo_CustLedgerEntryCaption
4	PostingDate_CustLedgerEntryCaption
5	DocumentType_CustLedgerEntryCaption
6	DocumentNo_CustLedgerEntryCaption
7	Description_CustLedgerEntryCaption
8	Skip this cell. You will use this cell later.
9	CurrencyCode_CustLedgerEntryCaption
10	Amount_CustLedgerEntryCaption
11	OriginalAmtLCY_CustLedgerEntryCaption
12	RemainingAmtLCY_CustLedgerEntryCaption

13. Right-click the left-most grouping cell (the cell that contains the **EntryNo_CustLedgerEntry** field) in the table, select **Text Box Properties**, in the Text Box Properties window, select the **Visibility** tab, under the **Change display options**, select the Hide option.
14. Select the first row in the table, in the **Properties** pane, under **Fill**, set the **BackgroundColor** property to **Dim Grey**.
15. Right-click the cell in the row2, column 2, and then choose **Expression** to open the **Expression** window.
16. In the **Category** column, select **Fields (DataSet_Result)**, in the **Values** column, double-click

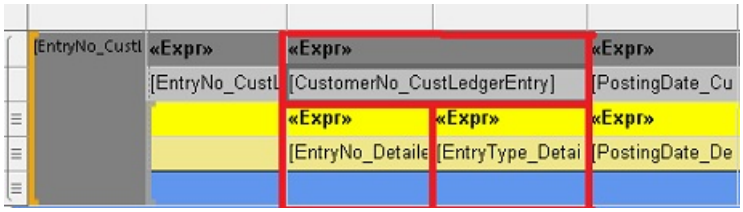
EntryNo_CustLedgerEntry, and then choose the **OK** button. Note that the **Set expression for: Value** box contains the following value: `=Fields!EntryNo_CustLedgerEntry.Value`

17. Repeat steps 15 and 16 for row 3 to add fields from the ledger entry dataset. Put the fields under the corresponding captions.
18. Select the row that you just filled and set the **BackgroundColor** property to **Silver**.
19. Build the project, inspect the **Output** pane, and make sure that there are no build errors.
20. Select the second table row, right-click the shaded border to the left, choose **Insert Row**, and then choose **Below**. The table should now have three group rows, one group data row, and one table footer row. This row will store the captions of **Detailed Cust. Ledg. Entry** data item.
21. Add the captions and fields for the **Detailed Cust. Ledger Entry table** as shown in the following table.

THIRD ROW (CAPTION)	FORTH ROW (FIELDS)
EntryNo_DetailedCustLedgEntryCaption	EntryNo_DetailedCustLedgEntry.Value
EntryType_DetailedCustLedgEntryCaption	EntryType_DetailedCustLedgEntry.Value
PostingDate_DetailedCustLedgEntryCaption.	PostingDate_DetailedCustLedgEntr.Value
DocumentType_DetailedCustLedgEntryCaption	DocumentType_DetailedCustLedgEntry.Value
DocumentNo_DetailedCustLedgEntryCaption	DocumentNo_DetailedCustLedgEnt.Value
TransactionNo_DetailedCustLedgEntryCaption	TransactionNo_DetailedCustLedgEntry.Value
JournalBatchName_DetailedCustLedgEntryCaption	JournalBatchName_DetailedCustLedgEntry.Value
AmountLCY_DetailedCustLedgEntryCaption	AmountLCY_DetailedCustLedgEntr.Value
DebitAmountLCY_DetailedCustLedgEntryCaption	DebitAmountLCY_DetailedCustLedgEntry.Value
CreditAmountLCY_DetailedCustLedgEntryCaption	CreditAmountLCY_DetailedCustLedgEntry.Value

22. Shrink the column that contains the **Customer No.** field of the Cust. Ledger Entry to about half of its size.
23. Right-click the column header that contains the **Customer No.** field, choose **Insert Column**, and then choose **Right**.
24. Select the cell that contains the **Customer No.** caption and the empty cell that you just created, and then choose **Merge Cells** to merge the two cells.
25. Repeat step 24 to merge the cell that contains the value of the **Customer No.** field and the empty cell that you created.
26. Assign the expression from the **EntryType** caption and field cells of the Detailed Cust. Ledg. Entry to the empty cell that you created to the right. You may have to have to cut the expressions and paste them into the empty cells.
27. Repeat 26 to move the **EntryNo** caption and field one cell to the right. This makes sure that the EntryNo and the EntryType data is located directly under the CustomerNo cell.

The following illustration shows EntryNo and the EntryType cells directly under the CustomerNo cell



28. Repeat steps through 27 to put the **Transaction No.** and **Journal Batch Name** captions and fields under the Description data. This creates a blank cell under the CurrencyCode field.

29. Select the third row and set the **BackgroundColor** property to **Yellow** and then set the **BackgroundColor** property of the fourth row to **Khaki**.

    Viktor will now hide all empty cells and add the totals to the footer row. To hide empty cells Viktor will add a filter that selects rows that have [EntryNo] value that are greater than zero.

To hide empty cells and add totals

1. Select the first row, right-click the shaded border to the left of the row, choose **Row Group**, and then choose **Group Properties**.

2. In the **Group Properties** window, select the **Filters** tab, and then choose the **Add** button.

3. Set **Expression** to **[EntryNo_CustLedgerEntry]**, change **Text** to **Integer**, set **Operator** to **>**, set **Value** to **0**, and then choose the **OK** button.

   The filter that is set applies to the other rows in the table.

4. In the **Group Properties** window, under the **Filters** tab, verify that the **Expression** box contains [EntryNo_CustLedgerEntry].

   Viktor will now add the total of the amount field to the footer row of the table, format the cells and hide the total cell if customer ledger entry is not available.

5. In the last row of the table, right-click the empty cell under the **Amount (LCY)** field, and then choose **Expression**.

6. In the **Category** column, select **Fields (DataSet_Result)**, in the **Values** column double-click **Amount_CustLedgerEntry**, and then change the expression in the **Set expression for: Value** box to the following value: `=Sum(Fields!Amount_CustLedgerEntry.Value)`. Choose the **OK** button.

7. In the **Properties** window, locate the **Format** property, choose the drop-down arrow and select **Expression**.

8. In the **Expression** window, enter the following formatting expression in **Set expression for: Value** box: `=Fields!Amount_CustLedgerEntryFormat.Value`. Choose the **OK** button.

   > **NOTE**
   > Alternatively, you set this value by double-clicking **Amount_CustLedgerEntryFormat** in the **Values** field of **Fields(DataSet)** category.

9. Select the two empty cells to the left of the total cell, right-click the cells, and then choose **Merge Cells**.

10. Right-click the merged cell, choose **Expression**, choose the **Parameters** category, and then set the caption to **Total_Caption**

11. Set the **BackgroundColor** property of the cells that contain the total and total caption to **Red**.

If you run the report now, the total amount cell will be displayed even if there are no ledger entries. Viktor will add an expression to hide the footer row when there are no ledger entries.

12. Select the last row, in the **Properties** window, locate the **Hidden** property, choose the drop-down arrow, and then choose **Expression**.
13. In the **Expression** window, in the **Set expression for: Hidden** box, enter the following expression to hide the row: `=Fields!EntryNo_CustLedgerEntry.Value = 0`. Choose the **OK** button. This hides the row if there are no entry values.
14. Right-click the left-most cell in the last table row, select **Text Box Properties**, select the **Visibility** tab, under **Change display options**, select the **Hide** option, and then choose the **OK** button.

The next step is to add the data from the **Sales Header** table.

To add the sales header data

1. From the **Toolbox**, drag a **Table** control to the **List** control, and then put the table control under the table that contains the **Cust. Ledger Entry** table
2. Right-click a column and add columns to create five columns for the table.
3. Delete the first header row from the table.
4. Right-click the data row, choose **Tablix Properties**, verify that the **DataSet name** is set to **DataSet_Result**, and then choose the **OK** button.
5. Right-click the data row, choose **Add Group**, and then choose **Parent Group** to open the **Tablix group** window.
6. Select the **Group by:** option and then choose the **fx** button to open the **Expression** window.
7. In the **Category** column, select **Parameters**, and then in the **Values**, column double-click **Sales_Document_Caption**. Verify that the **Set expression for: Value** box contains the following value: `=Parameters!Sales_Document_Caption.Value`. Choose the **OK** button.
8. In the **Tablix group** window, select **Add group header**, and then choose the **OK** button.
9. Right-click the first row in the table, choose **Insert Row**, and then choose **Inside Group – Above**.
10. Reduce the size of the first column, and then in the **Properties** window, under **Visibility**, set the **Hidden** property to **True**. This hides the first column.
11. In the first table row, merge all the cells except the first grouping cell.
12. Right-click the merged cell, and then choose **Expression**.
13. In the **Category** column, select **Parameters**, and then in the **Values** column double-click **Sales_Document_Caption**. Verify that the **Set expression for: Value** box contains the following value: `=Parameters!Sales_Document_Caption.Value`. Choose the **OK** button.
14. Modify the expression to `=First(Parameters!Sales_Document_Caption.Value)`.
15. Right-click the cell in row2, column 2, and then choose **Expression** to open the **Expression** window.
16. In the **Category** column, select **Parameters**, and then in the **Values** column double-click **DocumentType_SalesHeader**. Verify that the **Set expression for: Value** box contains the following value: `Parameters!DocumentType_SalesHeaderCaption.Value`.
17. Modify the expression to the following value: `=First(Parameters!DocumentType_SalesHeaderCaption.Value)` and then choose the **OK** button.

18. Right-click the cell that is under the caption that you just created, choose **Expression**. In the **Category** column, select **Fields (DataSet_Result)**, and then in the **Values** column double-click **DocumentType_SalesHeader**. Choose the **OK** button. Verify that the **Set expression for: Value** box contains the following value: `=Fields!DocumentType_SalesHeader.Value`

19. Repeat steps 15 through 18 and add the following captions and the corresponding fields.

CAPTION	CORRESPONDING FIELD
No_SalesHeaderCaption	No_SalesHeader
PostingDate_SalesHeaderCaption	PostingDate_SalesHeader
PricesIncludingVAT_SalesHeaderCaption	PricesIncludingVAT_SalesHeader
Amount_SalesHeaderCaption	Amount_SalesHeader

20. Select the first two rows and in the **Properties** window, set the **BackgroundColor** property to **Lime**.

21. Select the data row (last row), in the **Properties** window, set the **BackgroundColor** property to **Turquoise**.

    Viktor will now set a filter that hides empty rows.

To set a filter hide empty row

1. Select any row from this table, right-click the shaded border to the left of the table, and then choose **Tablix Properties**.

2. Choose the **Filters** tab and then choose the **Add** button.

3. In the **Expression** list box, select **No_SalesHeader**, set **Operator** to **>**, set **Value** to **0**, and then choose the **OK** button.

4. Save the report.

Building and Running the Report

Viktor will run the report to view how it looks like. For this, do the following:

1. Make sure that the `"startupObjectId"` is set to the **Id** of the report object and the `"startupObjectType"` to `Report` in the `launch.json` file.

2. Press the `F5` key to compile and run the report in Dynamics 365 Business Central.

3. If you have not disabled the UseRequestPage Property you will be shown a request page in the Web Client.

   The following illustration shows an example of the request page that is displayed when the report is run.

   If you choose the **Preview** button on the request page, the report will be displayed with the RLD layout just created.

Viktor can now add advanced features to the report. He can add features such as displaying the company name and logo on every page on the report. He might also want to add features that enable users to apply filters on the request page.

See Also

Report Overview
Defining a Report Dataset
Creating an RDL Layout Report

Formatting Decimal Values in Fields

2/17/2021 • 5 minutes to read • Edit Online

This article describes how you can format the decimal values that appear in fields on table, pages and reports. For example, you can change how the data appears in a Cue on the Role Center page. To format data, you use a combination of the AutoFormatType Property, AutoFormatExpression Property, and DecimalPlaces Property of the field. These properties work together to enable you to specify the following:

- Display amounts and unit amounts in another currency.
- Specify the number of decimal places.
- Specify whether to display a thousand separator.
- Specify characters before and after the value, such as currency signs or %.

Implementation overview

When a field is used on a page or report, you can set the **AutoFormatType** and **AutoFormatExpr** properties directly on the page field or report field (column), or you can set them on the underlying table field. If you specify the properties on the table field, then the format applies wherever the field is used. Specifying the properties on the page or report field will only apply the format to the specific page or report. If you specify the properties on the table field and the page or report field, then the settings on the page or report field take precedence.

When you use the **AutoFormatType** and **AutoFormatExpression** properties to format a field, two events are raised by the system codeunit **45 Auto Format**: OnResolveAutoFormat and OnAfterResolveAutoFormat.

Setting up data formatting

The settings for the **AutoFormatType**, **AutoFormatExpression**, and **DecimalPlaces** properties will depend on the type of data that is displayed, for example, this could be currency amounts, unit amounts, simple decimals, or ratios. For the most part, the **AutoFormatType** property is the primary setting, which in turn determines the options for setting the **DecimalPlaces** and **AutoFormatExpr** properties.

If the **AutoFormatType** is not set or is set to an incorrect property value, then the default setting is used, regardless of whether the **AutoFormatExpression** or **DecimalValues** properties are set. The default setting uses `AutoFormatType = 1` and `AutoFormatExpression = ''`.

The following tables describes how to set each of the properties to achieve the format that you want.

Setting the DecimalPlaces property

With the following set up, the **AutoFormatExpression** property is ignored.

AUTOFORMATTYPE PROPERTY	DECIMALPLACES PROPERTY	USAGE DESCRIPTION

AUTOFORMATTYPE PROPERTY	DECIMALPLACES PROPERTY	USAGE DESCRIPTION

0	Set to the number of decimal places that you want to display for the value.	Use this configuration when you want to format the decimal value according the Standard Format 0 (which is the default format) with a specific number of decimal places.
		
		For example, if the value is a US decimal `-76543.21` and you set the **DecimalPlaces** property to `0`, then the value appears as **76,543**. The properties will look like this:
		
		`AutoFormatType = 0;`
		`DecimalPlaces = '0';`

Setting the AutoFormatExpression property

With the following setup, the **DecimalPlaces** property is ignored.

AUTOFORMATTYPE PROPERTY	AUTOFORMATEXPRESSION PROPERTY	USAGE DESCRIPTION

1	Set to return a currency code, such as USD or IDR. The blank currency code `''` denotes LCY and is the default value.	Use this configuration when you want to format the data as an amount. For example, a sales order will use two decimals when the currency is defined as US dollar and no decimals when the currency is defined as IDR (Indonesian rupiah). For example:
		
		`AutoFormatType = 1;`
		`AutoFormatExpression = 'IDR';`

2	Set to return a currency code such as USD or IDR. The blank currency code `''` denotes LCY and is the default value.	This is similar to the previous configuration where the **AutoFormatType** property is set to `1`, except you use this configuration when you want to format the data as a unit amount.

AUTOFORMATTYPE PROPERTY	AUTOFORMATEXPRESSION PROPERTY	USAGE DESCRIPTION

10	Set to the property according to the following syntax:
	`'[SubType][,<currencycode or expression>[,<PrefixedText>]]'`
	`SubType` can be `1`, `2`, another number, or omitted:
	`1` sets the value to an amount type (see 1 above). `2` sets the value to a unit amount type (see 2 above). The syntax for these two settings is:
	`'SubType,<currencycode[, <PrefixedText>]'`
	If you omit the subtype or use a number other than 1 or 2, the syntax is:
	`'<CustomNumber>, <expression>[, <PrefixedText>]'`
	where `<expression>` sets the precision and one of the standard formats. For more information, see Standard Formats.
		Use SubType `1` to add the currency symbol and use the amount type precision. You use SubType `2` for unit amount precision. For example, set the property to `'1,USD'` to add the **$** symbol, like **$543.21**.
		`AutoFormatType = 10;`
		`AutoFormatExpression = '1,USD';`
		If you omit the SubType, you can use this configuration to customize the format based on one of the standard formats. This option enables you to specify characters before and after the decimal value, such as currency signs **$** and percent **%**.
		For example, if you want to prefix the decimal value with a **$**, include a thousand separator, and have a maximum of two decimal places, such as **$76,453.21**, then you can set the properties to:
		`AutoFormatType = 10;`
		`AutoFormatExpression = '$<precision, 2:2><standard format, 0>'`
		If you want to display the decimal value as a percentage, then you can add `%` at the end of the setting. For example:
		`AutoFormatType = 10;`
		`AutoFormatExpression = '<precision, 1:1><standard format,0>%'`
		When you include a `%` at the end of the setting, then the decimal value is assumed to be the ratio, and the decimal value will be multiplied by 100. For example, a value of 0.98 will be formatted to **98%**.

11	Set the property to the standard format as explained below. For example: `'<Precision,3:3><Standard Format,0>'`	Use this option when you want full control over the formatting. The format string will be applied exactly as specified in the AutoFormatExpr property.

Precision

The precision determines the minimum and maximum number of decimal points for values. The precision takes the format `<precision,minimum:maximum>`. For example, `<precision,minimum:maximum>` sets the data with a minimum of 2 and a maximum of 3 decimal places.

Standard formats

The following table describes the standard formats that are available for the **AutoFormatExpr** property when the **AutoFormatType** property is set to 10.

STANDARD FORMAT	FORMAT DESCRIPTION	EUROPE DECIMAL EXAMPLE	US DECIMAL EXAMPLE
0	<Sign><Integer Thousand><Point or Comma><Decimals>	-76.543,21	-76,543.21
1	<Sign><Integer><Point or Comma><Decimals>	-76543,21	-76543.21
2	<Sign><Integer><Point or Comma><Decimals>	-76543.21	-76543.21
3	<Integer Thousand><Point or Comma><Decimals><Sign>	76.543,21-	76,543.21-
4	<Integer><Decimals><Point or Comma><Sign>	76543,21-	76543.21-
9	XML format	-76543.21	-76543.21

See Also

AutoFormatType Property
AutoFormatExpression Property
DecimalPlaces Property

Developing Printer Extensions in Business Central

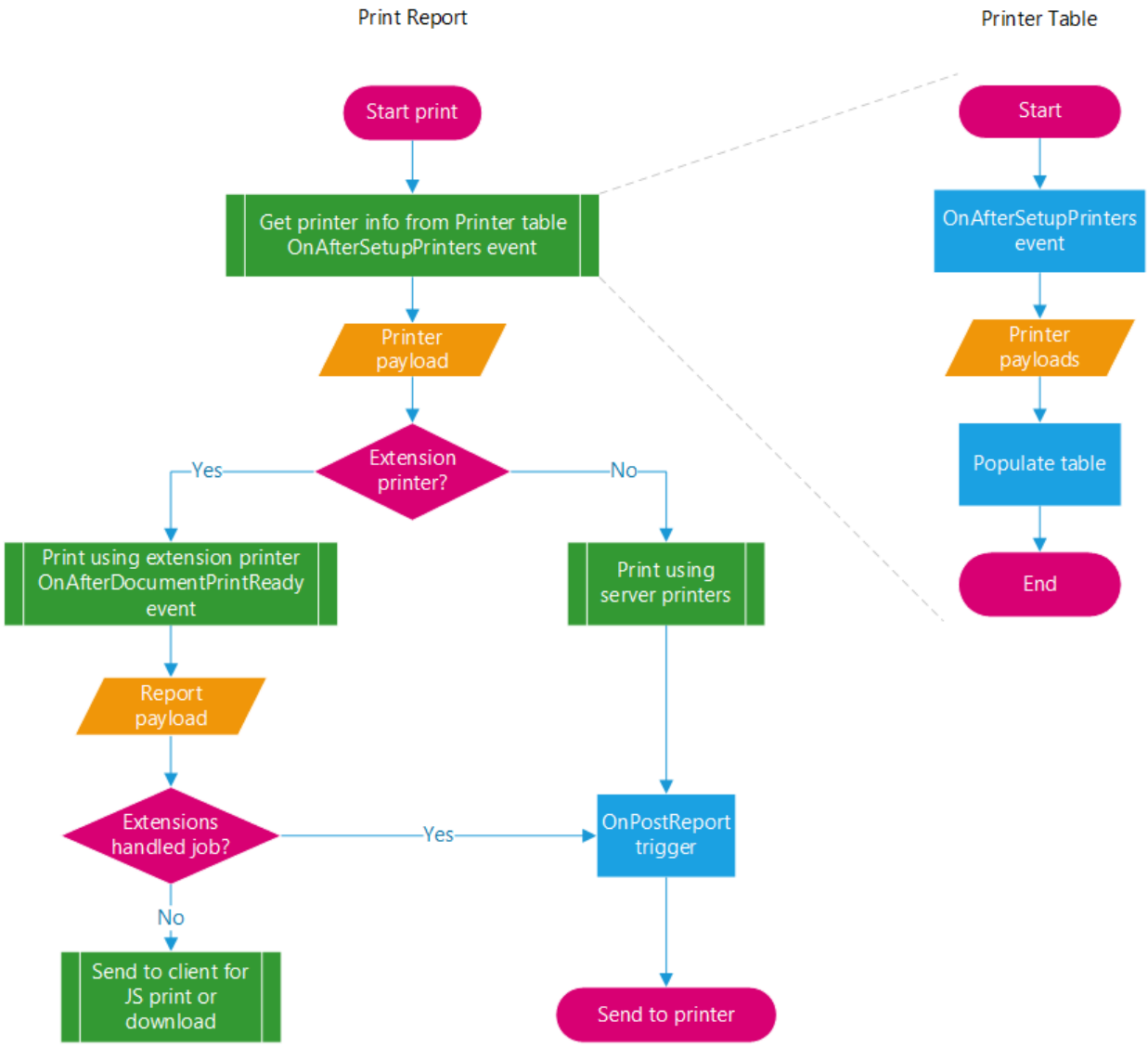
2/17/2021 • 3 minutes to read • Edit Online

This article provides an overview about how to add code that enables Business Central reports to be sent from the client directly to a web-connected printer, like an email printer or through a cloud printing service.

Without any customized code, there's no way to send a report directly to a printer for printing. The only way to print a report from the client is to download it first (as a .pdf file), and then send it to a printer. But you can create extensions that are designed to send reports to web-connected printers. For example, if you have an email printer, you create an extension that sends print jobs directly to the printer's email address.

Overview

To accommodate printing, the system publishes two events that you can subscribe to: OnAfterSetupPrinters and OnAfterDocumentPrintReady. The following figure illustrates the runtime execution of the events:



Throughout the print process, the system compiles the report into a PDF file and passes it in a stream object, then finally sends the PDF file the printer.

OnAfterSetupPrinters event

You subscribe to the OnAfterSetupPrinters event to set up different printers that users can use on reports, which

they select from the **Printer Selections** page. The OnAfterSetupPrinters event is a global integration event that is published from codeunit **44 Report Managements**. It has the following declaration:

```
[IntegrationEvent(false, false)]
local procedure OnAfterSetupPrinters(var Printers: Dictionary of [Text[250], JsonObject]);
```

The *Printers* parameter is a Dictionary of [Text, Text] type that includes a collection of key-value pairs that define different printer setups. The key specifies a name for the printer. The value is a JSON object that specifies settings supported by the printer. The settings include information like paper size, paper trays, default copies, and more. The JSON object is referred to as the *payload*.

The OnAfterSetupPrinters event is raised when the following operations occur:

- When you open a page that uses the virtual table **Printer** as its source.
- From page **64 Printer Selections** in the base application, when you access the **Printer Name** field for selecting a printer for a report. This field has a lookup to the **Printer** table.
- From a report request page, when you select the **Print** or **Preview** action for a report. But only if the report is set up to use a specific printer from the **Printer Selections** page.

OnAfterDocumentPrintReady

You subscribe to the OnAfterDocumentPrintReady event to specify what happens when the user selects the **Print** action on a report request page. You use OnAfterDocumentPrintReady event subscribers to provide instructions on how and where to send to the report for printing. For example, with email printing, the OnAfterDocumentPrintReady event subscriber would construct and send an email to a target printer.

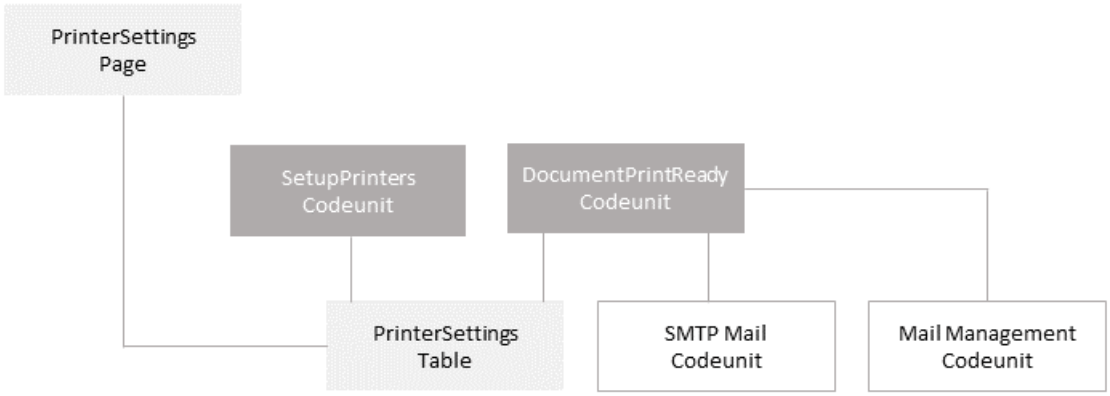
The OnAfterDocumentPrintReady event is also a global integration event that is published from codeunit **44 Report Managements**. It has the following declaration:

```
IntegrationEvent(false, false)]
local procedure OnAfterDocumentPrintReady(ObjectType: Option "Report","Page"; ObjectId: Integer;
ObjectPayload: JsonObject; DocumentStream: InStream; var Success: Boolean);
```

The event is raised when the use selects the **Print** action on the request page of a report. Two parameters of interest are *DocumentStream* and *ObjectPayload*. *DocumentStream* is an Upstream object that contains the report data to be printed. The *ObjectPayload* is a JsonObject type object that combines the printer payload and report metadata. The report metadata includes information like the company name, MIME type, views, and more. This combination is referred to as the *report payload*.

Development Overview

To develop a printer extension, you create codeunits that subscribe to the OnAfterSetupPrinters and OnAfterDocumentPrintReady events. Plus, you would typically add other objects to support the printing. For example, you could create a table that stores printer setups and a page that enables users to manage them.



For more information about using the OnAfterSetupPrinters and OnAfterDocumentPrintReady events to create a printer extension, see Creating a Printer Extension.

See Also

Events in AL
Publishing Events
Raising Events
Subscribing to Events

Creating a Printer Extension

2/17/2021 • 6 minutes to read • Edit Online

This article describes how to use the OnAfterSetupPrinters and OnAfterDocumentPrintReady events to set up different printers for reports.

Overview

This article uses a simplified printer extension example for sending reports to an email printer. The example creates a printer extension that sets up a single email printer. Users can then assign the printer to reports from the **Printer Selections** page in the client. In this article, you will:

- Create two codeunits, one that subscribes to the OnAfterSetupPrinters event and another that subscribes to the OnAfterDocumentPrintReady event.
- Use the **SMTP Mail** and **Mail Management** codeunits that are part of the base application for sending the report via email.

Set up printers using the OnAfterSetupPrinters event

This section describes how to use the OnAfterSetupPrinters event to set up a printer. When completed, users can select the printer from on the **Printer Selections** page.

Getting started

1. Create an AL project for the printer extension.

   See Getting Started with AL.

2. Create a codeunit to use for subscribing to the OnAfterSetupPrinters event.

3. Create a method that subscribes to the OnAfterSetupPrinters event.

   At this point, your code would look something like this:

   ```
   codeunit 50100 SetupPrinter
   {
       [EventSubscriber(ObjectType::Codeunit, Codeunit::"ReportManagement", 'OnAfterSetupPrinters', '',
   true, true)]
       procedure OnSetupPrinters(var Printers: Dictionary of [Text[250], JsonObject]);
       var

       begin

       end;
   }
   ```

   For more information about subscribing to events, see Subscribing to Events.

4. Now you can start adding code to the event subscriber method to create the printer payloads.

   In this example, you set up a printer named 'My Printer' that has two paper trays: A4 and Custom. There are two ways of setting values for a payload JSON object. You can add properties (key-value pairs) by using the Add method, as shown in the example. Or, you can use the ReadFrom method to read the JSON data from the string into a JsonObject variable (see Using ReadFrom to create the payload).

```
codeunit 50100 SetupPrinter
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"ReportManagement", 'OnAfterSetupPrinters', '',
true, true)]
    procedure OnSetupPrinters(var Printers: Dictionary of [Text[250], JsonObject]);
    var
        Payload: JsonObject;
        PaperTrays: JsonArray;
        PaperTrayA4: JsonObject;
        PaperTrayCustom: JsonObject;
    begin
        // Step 1: Create the paper trays
        PaperTrayA4.Add('papersourcekind', 'Upper');
        PaperTrayA4.Add('paperkind', 'A4');

        PaperTrayCustom.Add('papersourcekind', 'Custom');
        PaperTrayCustom.Add('paperkind', 'Custom');
        PaperTrayCustom.Add('width', 236);
        PaperTrayCustom.Add('height', 322);
        PaperTrayCustom.Add('units', 'mm');

        // Step 2: Add the paper trays to the list of paper trays
        PaperTrays.Add(PaperTrayA4);
        PaperTrays.Add(PaperTrayCustom);

        // Step 3: Add the required parameters for the payload
        Payload.Add('version', 1);
        Payload.Add('papertrays', PaperTrays);

        // Step 4: Add the printer to the dictionary
        Printers.Add('My Printer', Payload);
    end;
}
```

5. At this point, you can compile your project, and publish/install the extension on the tenant to test the payload.

   Open the **Printer Selections** page in the client. You should see **My Printer** as an option in the **Printer Name** field. If there are errors with payload, you'll get an error when you try to open the **Printer Name** field. For more information, see Troubleshooting Printer Payload Errors.

About the printer payload

The event subscriber method has one parameter, which is a dictionary of printers. The key is a name of the printer. The value is a JSON object that is referred to as the payload. The payload specifies information about a specific printer. The payload includes several attributes in the following structure:

```
{
    "version": 1,
    "description":[default=""],
    "duplex":[default=false],
    "color":[default=false],
    "defaultcopies":[default=1],
    "papertrays":
    [
        {
            "papersourcekind":'Upper' | 1,
            "paperkind":'A4' | 9,
            "units":[default='HI'],
            "height":[default=0],
            "width":[default=0],
            "landscape":[default=false]
        }
    ]
}
```

There are few required attributes, such as `version` and `papertrays`. The `papertrays` attribute must contain at least one paper tray setup, which in turn must include the attributes `papersourcekind` and `paperkind`. For more information about the attributes, see Printer Payload.

Using the ReadFrom method to create the payload

The following example illustrates how to create a payload by using the ReadFrom method.

```
codeunit 50101 SetupPrinter2
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"ReportManagement", 'OnAfterSetupPrinters', '', true,
true)]
    procedure SetupPrinters(var Printers: Dictionary of [Text[250], JsonObject]);
    var
        Payload: JsonObject;
    begin
        // Step 1: Read the payload from the text
        Payload.ReadFrom('{"version":1,"papertrays":[{"papersourcekind":"Upper","paperkind":"A4"},
{"papersourcekind":"Custom","paperkind":"Custom","width":236,"height":322,"units":"mm"}]}');

        // Step 2: Add the printer to the dictionary
        Printers.Add('My Printer', Payload);
    end;
}
```

Select the paper tray

A printer can have several paper trays. If a report doesn't specify which paper tray to use or the specified paper tray isn't present in the printer's setup, a default paper tray is used. The default paper tray is the first one defined in the `papertrays` list.

You can change the paper tray for an existing report by subscribing to the `OnAfterGetPaperTrayForReport` event and setting a value for a `DefaultPage` parameter.

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"ReportManagement", 'OnAfterGetPaperTrayForReport', '',
true, true)]
procedure GetPaperTrayForReport(ReportID: Integer; var FirstPage: Integer; var DefaultPage: Integer; var
LastPage: Integer)
```

As an alternative, if you're creating a new report, you can set a paper tray by specifying PaperSourceDefaultPage property.

```
report 50103 MyReport
{
    PaperSourceDefaultPage = Upper;
            ...
}
```

Handle print using the OnAfterDocumentPrintReady event

This section describes how to subscribe to the OnAfterDocumentPrintReady event. You subscribe to the OnAfterDocumentPrintReady event to define what happens when a user chooses to print a report. The event subscriber specifies how and where to send the report.

Getting started

1. Create a codeunit to use for subscribing to the OnAfterDocumentPrintReady event.
2. Create a method that subscribes to OnAfterDocumentPrintReady event.

   At this point, your code would look something like this:

   ```
   codeunit 50102 HandlePrintAction
   {
       [EventSubscriber(ObjectType::Codeunit, Codeunit::ReportManagement, 'OnAfterDocumentPrintReady',
   '', true, true)]
       procedure OnDocumentPrintReady(ObjectType: Option "Report","Page"; ObjectId: Integer;
   ObjectPayload: JsonObject; DocumentStream: InStream; var Success: Boolean);
       var

       begin

       end;
   }
   ```

   For more information about subscribing to events, see Subscribing to Events.

3. Now you can start adding code to the event subscriber method to handle the report payload and send the report.

   The following code sends a report by email to the printer that is named 'My Printer'. In this example, the printer's email address is 'myprinterb@businesscentral.onmicrosoft.com'.

```
codeunit 50101 SendReportByEmail
{
    [EventSubscriber(ObjectType::Codeunit, Codeunit::ReportManagement, 'OnAfterDocumentPrintReady',
'', true, true)]
    procedure OnDocumentPrintReady(ObjectType: Option "Report","Page"; ObjectId: Integer;
ObjectPayload: JsonObject; DocumentStream: InStream; var Success: Boolean);
    var
        SMTPMail: Codeunit "SMTP Mail";
        PrinterNameToken: JsonToken;
        PrinterName: Text;
        ObjectNameToken: JsonToken;
        ObjectName: Text;
        DocumentTypeToken: JsonToken;
        DocumentTypeParts: List of [Text];
        FileExtension: Text;
        MailManagement: Codeunit "Mail Management";
        SendFrom: Text;
        FileName: Text;
        Recipients: List of [Text];
    begin
        begin
            // Step 1: Before doing anything, it is important to check the current Success value
            if Success then
                exit;

            // Step 2: Make sure code only runs for reports
            if ObjectType = ObjectType::Report then begin

                // Step 3: Get report object payload information
                ObjectPayload.Get('printername', PrinterNameToken);
                PrinterName := PrinterNameToken.AsValue().AsText();
                if PrinterName = 'My Printer' then begin
                    ObjectPayload.Get('objectname', ObjectNameToken);
                    ObjectName := ObjectNameToken.AsValue().AsText();
                    ObjectPayload.Get('documenttype', DocumentTypeToken);

                    // Step 4: Build the email message
                    DocumentTypeParts := DocumentTypeToken.AsValue().AsText().Split('/');
                    FileExtension := DocumentTypeParts.Get(DocumentTypeParts.Count);
                    Recipients.Add('myprinterb@businesscentral.onmicrosoft.com');
                    SendFrom := MailManagement.GetSenderEmailAddress();
                    SMTPMail.CreateMessage('Sender', SendFrom, Recipients, 'Hello this is your
report', 'Please take a look');
                    SMTPMail.AddAttachmentStream(DocumentStream, ObjectName + '.' + FileExtension);

                    // Step 5: Send the email for print
                    SMTPMail.Send;
                    Success := true;
                    exit;
                end;
            end;
        end;
    end;
}
```

4. At this point, you can compile and publish/install the extension on a tenant to test.

   First, make sure that SMTP email is set up on the tenant (see Set Up Email in the Application Help).

   Then, on the **Printer Selections** page, set a report to use 'My Printer', and then run and print the report.

About the report payload

The event subscriber receives the printer payload and combines it with the report metadata, like the report's ID. This combination is the report payload. The content of the report itself is received as a stream object. You add code to define how and where to send the report payload for printing. In this example, it's sent as an email.

The report object payload is a JSON object that includes several parameters and values arranged in a specific structure, as shown in the following example


```
{
    "filterviews":
    [
        {"name":"Header","tableid":112,"view":"VERSION(1) SORTING(Field3) WHERE(Field3=1(103027))"},
        {"name":"Line","tableid":113,"view":"VERSION(1) SORTING(Field3,Field4) WHERE(Field4=1(0..10000))"},
        {"name":"ShipmentLine","tableid":7190,"view":"VERSION(1) SORTING(Field1,Field2,Field3)
WHERE(Field2=1(10000))"}
    ],
    "version":1,
    "objectname":"Standard Sales - Invoice",
    "objectid":1306,
    "documenttype":"application/pdf",
    "invokedby":"00000000-0000-0000-0000-000000000001",
    "invokeddatetime":"2020-01-17T15:33:52.48+01:00",
    "companyname":"CRONUS International Ltd.",
    "printername":"My Printer",
    "duplex":false,
    "color":false,
    "defaultcopies":1,
    "papertray":
    {
        "papersourcekind":257,
        "paperkind":0,
        "landscape":false,
        "units":0,
        "height":1268,
        "width":929
    }
}
```


The parameters can be read but not modified at runtime. For more information about the report payload, see Report Payload.

See Also

Working With and Troubleshooting Payloads
Developing Printer Extensions Overview
Events in AL
Publishing Events
Raising Events
Subscribing to Events

Queries in Business Central

2/17/2021 • 2 minutes to read • Edit Online

Queries enable you to retrieve records from one or more tables or data sources and then combine the data into rows and columns in a single dataset. Queries can also perform calculations on data, such as finding the sum or average of all values in a column of the dataset.

Query Types

There are two types of query objects: normal and API.

- A normal query retrieves records from business data tables in the Dynamics 365 Business Central database, and can be used to display data in the user interface. This type of query is created by a query object. For more information, Query Object.
- An API query is used to generate web service endpoints and this type of page cannot be displayed in the user interface. A query of the API type can be used to join data from different data sources. The data can only be viewed. For information about creating a query of the type API, see API Query Type.

Query usages

The following examples show how you can use queries in your Dynamics 365 Business Central application.

- Creating charts that are based on a query instead of a table.
- Saving a query as an .xml or .csv file. For example, you can use the SAVEASXML method to create an .xml file that contains the resulting dataset of a query. You can use the .xml file to integrate with external applications.
- Exposing data as an OData web service. You can register and publish a query as a web service in the same way that you can register and publish pages or codeunits as web services. You use the **Web Services** page to register and publish pages, codeunits, or queries. After you expose a query as a web service, you can import it into other applications.
- Using the query as a data source for a page. To do this, you have to copy the query resulting dataset into a temporary table and set it as the source table for the page.
- Using the query as a data source for a report. To do this, you have to copy the query resulting dataset into a temporary table which can then be used by the report.
- Performing calculations on data such as computing sums and averages. For more information see, Query Totals and Grouping.
- Replacing nested loops that use record variables to retrieve or to detect duplicate records. For more information, see Using Queries Instead of Record Variables.

Query Data Type

The query resulting dataset cannot be used directly by other objects such as pages and reports. Instead, you must create a Query Data Type instance and apply methods for handling the data from the query object. You must first call the OPEN Method to be able to perform actions such as reading the dataset or setting filters and you must use the CLOSE Method when you are finished. You can write this code in a codeunit, for example. The following example illustrates how to do this.

```
var
    MyQuery: Query "Customer SalesQuantity";
    Text000: Label 'Customer name = %1, Quantity = %2';
begin
    // Sets a filter to display only sales quantities greater than 20.
    MyQuery.SETFILTER(Quantity, '>20');
    // Runs the query.
    MyQuery.OPEN;
    // Reads each row in the dataset and displays a message with column values.
    // Stops reading when there are no more rows remaining in the dataset (READ is FALSE).
    while MyQuery.READ do
    begin
      MESSAGE(Text000, MyQuery.Name, MyQuery.Quantity);
    end;
    MyQuery.CLOSE;
end;
```

See Also

Query Object
Linking and Joining Data Items
Aggregating Data in Query Objects
Query Properties
Developing Extensions
AL Development Environment
Utilizing Read Scale-Out for Better Performance

Query Object

2/17/2021 • 3 minutes to read • Edit Online

Business Central query objects enable you to retrieve records from one or more tables and then combine the data into rows and columns in a single dataset. Query objects can also perform calculations on data, such finding the sum or average of all values in a column of the dataset.

There are two types of query objects: normal and API. This article describes normal query objects, which can be used to display data in the user interface. API query objects are used to generate web service endpoints and cannot be displayed in the user interface. For information about creating a query of the type API, see API Query Type.

Creating a query object

A query object is comprised mainly of two different types of elements: dataitems and columns. A dataitem specifies the table to retrieve records from. A column specifies a field of the table to include in the resulting dataset of a query. The basic steps to create a query object are as follows:

1. Add the `query` keyword, followed by the `elements` control.
2. Build the dataset by adding `dataitem` controls and `column` controls within the `elements` control.

   The hierarchy of the `dataitem` and `column` controls is important because it will determine the sequence in which data items are linked, which in turn will control the results. Working from top-to-bottom, you start by adding the `dataitem` control for first table that you want in the dataset, then add `column` controls for each table field that you want to include in the dataset. For the next table, you add another `dataitem` control that is embedded within the first `dataitem` control, then add `column` controls as needed. You continue this pattern for additional tables and fields.
3. When you have specified the dataitem and column elements, create links and joins between the `dataitem` elements.

   Dataitem links and joins determine which records to include in the dataset based on the values of a common field between dataitems. You set a link between one or more fields of the dataitem tables with the DataItemLink Property and you define the type of the link using the SQLJoinType Property. Both properties must be set on the lower dataitem of the query object. For more information, see Linking and Joining Data Items.

The following shows the basic structure of a query object.

```
query ID Name
{

    elements
    {
        dataitem(DataItem1; Table1)
        {
            column(Column1; Field1)
            {
            }
            column(Column2; Field2)
            {
            }
            dataitem(DataItem2; Table2)
            {
                // Sets a link between FieldY of Table2 and FieldX of Table1.
                DataItemLink = FieldY = DataItem1.FieldX;
                //The dataset contains records from Table1 and Table2 where a match is found between FieldY
and FieldX.
                SqlJoinType = InnerJoin;

                column(Column1; Field1)
                {
                }
                dataitem(DataItem3; Table3)
                {
                    DataItemLink = FieldZ = DataItem2.FieldY;
                    SqlJoinType = InnerJoin;
                    column(Column1; Field1)
                    {
                    }
                }
            }
        }
    }
}
```

NOTE

Extension objects can have a name with a maximum length of 30 characters.

Snippet support

Typing the shortcut `tquery` will create the basic layout for a Query object when using the AL Language extension in Visual Studio Code.

TIP

Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Query example

The following example shows a query that displays a list of customers with sales and profit figures. The query primarily retrieves fields from the **Customer** table, but also displays fields from the **Salesperson Purchaser** and **Country Region** tables.

The query also uses the DataItemLink property to create a link between the **Customer** table, **Salesperson**

Code field and the **Salesperson Purchaser** table, **Code** fields and a link between the **Customer** table, **Country/Region Code** field and the **Country/Region** table, **Code** field.

```
query 50102 "Top Customer Overview"
{
    QueryType = Normal;
    Caption = 'Top Customer Overview';

    elements
    {
        dataitem(Customer; Customer)
        {
            column(Name; Name)
            {
            }
            column(No; "No.")
            {
            }
            column(Sales_LCY; "Sales (LCY)")
            {
            }
            column(Profit_LCY; "Profit (LCY)")
            {
            }
            column(Country_Region_Code; "Country/Region Code")
            {
            }
            column(City; City)
            {
            }
            column(Global_Dimension_1_Code; "Global Dimension 1 Code")
            {
            }
            column(Global_Dimension_2_Code; "Global Dimension 2 Code")
            {
            }
            column(Salesperson_Code; "Salesperson Code")
            {
            }
            dataitem(Salesperson_Purchaser; "Salesperson/Purchaser")
            {
                DataItemLink = Code = Customer."Salesperson Code";
                column(SalesPersonName; Name)
                {
                }
                dataitem(Country_Region; "Country/Region")
                {
                    DataItemLink = Code = Customer."Country/Region Code";
                    column(CountryRegionName; Name)
                    {
                    }
                }
            }
        }
    }
}
```

IMPORTANT

You cannot run a query that gets data from both the application database and the business data database. This also applies to single-tenant deployments so that you do not have to rewrite queries if you decide to export the application. For a description of which tables are considered part of the application database, see Separating Application Data from Business Data.

See Also

Linking and Joining Data Items
Aggregating Data in Query Objects
Query Objects and Performance
Query Properties
Developing Extensions
AL Development Environment
API Query Type

Linking and Joining Data Items to Define the Query Dataset

2/17/2021 • 11 minutes to read • Edit Online

Business Central queries enable you to retrieve records from one or more tables and combine the specific records into rows in a single dataset. In AL, each table is specified as a data item. The data included in the dataset is a result of how the data items are linked and joined together.

> **TIP**
> The concept of linking and joining data items in AL is similar to Join clauses in SQL Select statements on tables in SQL Server. For those familiar with SQL Joins, when describing links and joins in AL, this article provides the equivalent SQL SELECT statement in most cases.

Sample Tables and Query

To demonstrate data item links and joins, this article uses the following sample tables and query.

Salesperson/Purchaser Table

The **Salesperson/Purchaser** table contains a list of salespersons. Each salesperson is identified by a unique code. The following is a simplified version of the **Salesperson/Purchaser** table for demonstration purposes.

CODE	NAME
AA	Annette
BB	Bart
DD	Debra
JJ	John

Sales Header Table

The **Sales Header** table contains a list of sales orders. Each sales order has a unique number, includes the name of the customer to sell to, and is assigned to a salesperson by the **Salesperson_Code** column. The following is a simplified version of the **Sales Header** table for demonstration purposes.

NO.	SELL-TO CUSTOMER NAME	SALESPERSON CODE
1000	Autohaus	AA
2000	Blanemark	DD
3000	Candoxy	JJ
4000	New Concepts	

Sample Query

The following query object links the **Sale Header** table with the **Salesperson/Purchaser** table on the **Salesperson_Code** and **Code** fields, as specified by the DataItemLink Property. In the example, the SQLJoinType Property is set to **InnerJoin**.

```
query 50100 "Sample Query"
{
    QueryType = Normal;
    Caption = 'Sales Overview';

    elements
    {
        dataitem(Salesperson_Purchaser; "Salesperson/Purchaser")
        {
            column(Salesperson; Name)
            {

            }
            dataitem(Sales_Header; "Sales Header")
            {
                DataItemLink = "Salesperson Code" = Salesperson_Purchaser.Code;
                // Change the SqlJoinType value to suit the desired results: LeftOuterJoin, InnerJoin,
RighOuterJoin, FullJoin, CrossJoin.
                SqlJoinType = InnerJoin;

                column(Order_Number; "No.")
                {

                }
                column(Sell_to_Customer; "Sell-to Customer Name")
                {

                }
            }
        }
    }
}
```

How to link and join data items

When you add data items to a query object in AL, you define them in a specific hierarchy, one after another, where each lower data item is embedded within the definition of the upper data item. The order of the data items determines the sequence in which data items are linked and joined to produce the results in the dataset.

In short, to join two data items, you set the DataItemLink and SqlJoinType properties on the lower data item in the query object.

Set the DataItemLink Property

The DataItemLink Property sets up a reference or association between one or more fields in the source table of a lower data item tables with a field in the source table of the upper data item. In a query, two data item tables typically will have columns that have values that are common to both tables. For example, the **Salesperson** table and **Sales Header** table have the **Code** column and **Salesperson_Code** column in common. To create a link between these two tables, you could set the DataItemLink property of the **Sales Header** data item as follows:

```
DataItemLink = "Salesperson Code" = Salesperson_Purchaser.Code;
```

The DataItemLink Property sets up an "equal to" (=) comparison condition between two columns of the data items. When the query is run, the query compares each row of the two data items to find records that having matching values for the columns. Records that have matching column values are combined into a row in the

resulting dataset. In some cases, there will be records that do not have matching values. You use the SqlJoinType Property to include records that do not have matching column values.

Set the SqlJoinType Property

The SqlJoinType Property the determines which records to combine into the results, based on the values of the fields that are linked by the DataItemLink property. You use this property to limit the records that are included in the resulting dataset based on the specified conditions. By default, the SqlJoinType property is `LeftOuterJoin`, so if you omit this property, a `LeftOuterJoin` is performed.

> **TIP**
> In SQL join statements, tables are designated as either left or right. In AL query objects, because data items are arranged vertically, when joining data items, the *left* corresponds to the upper data item (table) and *right* corresponds to the lower data item (table).

Linking More Than Two Data Items

A query links data items in the order that they appear in AL, starting from the top and then working downward. When you have more than two data items, lower data items are linked to the resulting dataset of the linked data items above it. For example, when you link the first two data items, the query generates a dataset. When you add another data item, it is linked to the dataset of the first linked pair of data items, where it applies the conditions that are specified by its DataItemLink Property and SqlJoinType Property. The following code adds another data item to the sample query:

```
query 50100 "Sample Query"
{
    QueryType = Normal;
    Caption = 'Sales Overview';

    elements
    {
        dataitem(Salesperson_Purchaser; "Salesperson/Purchaser")
        {
            column(Salesperson; Name)
            {

            }
            dataitem(Sales_Header; "Sales Header")
            {
                DataItemLink = "Salesperson Code" = Salesperson_Purchaser.Code;
                SqlJoinType = InnerJoin;

                column(Order_Number; "No.")
                {

                }
                column(Sell_to_Customer; "Sell-to Customer Name")
                {

                }
                dataitem(Sales_Line; "Sales Line")
                {
                    DataItemLink = "Sell-to Customer No." = Sales_Header."Sell-to Customer No.";
                }
            }
        }
    }
}
```

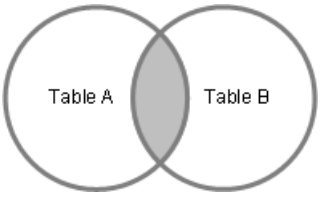
This pattern continues for each additional data item.

InnerJoin

`InnerJoin` creates a dataset by combining records from data item tables where a match is found between the columns that are linked by the DataItemLink Property of the lower data item. `Inner Join` uses an "equal to" comparison operator to match rows from the lower data item table with rows from the upper data item table that is based on the values of the linked columns.

- Each pair of matching records is combined into a row in the dataset.
- Records from the upper and lower data item tables that do not have a matching column in the lower data item table are excluded from the resulting dataset.

The following illustration shows an `InnerJoin` type between tables A and B. The shaded area indicates the records that are included in the resulting dataset.



Dataset Example

The following table shows the resulting dataset for an `InnerJoin` between the **Sales Header** table and **Salesperson/Purchaser** table in sample query.

SALESPERSON	ORDER_NUMBER	SELL_TO_CUSTOMER
Annette	1000	Autohaus
Debra	2000	Blanemark
John	3000	Candoxy

The records for **Bart** in the Salesperson table and **New Concepts** in the **Sales Header** table do not have matching records in the opposing table, so they are excluded from the resulting dataset.

SQL SELECT Statement for Inner Join

To specify an inner join with an SQL statement, you can do either of the following:

- Use a WHERE clause.
- Use the INNER JOIN condition with an ON clause.

The following two examples show how to create an inner join on the **Salesperson/Purchaser** and **Sales Header** tables with SQL statements. These two statements result in the same dataset.

```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser" INNER JOIN "Sales Header"
  ON "Salesperson/Purchaser".Code = "Sales Header"."Salesperson Code"
```

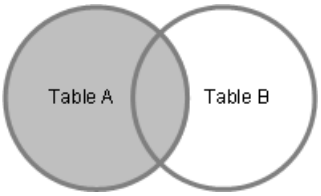
```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser", "Sales Header"
WHERE "Salesperson/Purchaser".Code = "Sales Header"."Salesperson Code"
```

LeftOuterJoin

A `LeftOuterJoin` resembles the `InnerJoin` except that the resulting dataset set contains every record from the upper data item table, even if a record does not have a matching value in the lower data item for columns that are linked by the DataItemLink Property.

- For each record in the upper data item, a row is added in the dataset that combines columns from the upper and lower data item.
- When a record in the upper data item table has no matching record in the lower data item table, columns coming from the lower data item table have null values.

The following illustration shows a `LeftOuterJoin` type between tables A and B. The shaded area indicates the records that are included in the resulting dataset. In the sample query, the **Salesperson/Purchaser** table is considered the left table.



Dataset Example

The following table shows the resulting dataset for a `LeftOuterJoin` between the **Sales Header** table and **Salesperson/Purchaser** table in sample query.

SALESPERSON	ORDER_NUMBER	SELL_TO_CUSTOMER
Annette	1000	Autohaus
Bart	null	null
Debra	2000	Blanemark
John	3000	Candoxy

The record for **Bart** in the Salesperson/Purchaser table does not have a matching record in the **Sales Header** table, so a row is included but the columns from the **Sale Header** table are given null values. The record for **New Concepts** in the **Sale Header** table is not included in the resulting dataset because it does not have a matching column in the **Salesperson/Purchaser** table.

SQL SELECT Statement for Left Outer Join

To specify a left outer join with an SQL statement, you use the LEFT OUTER JOIN condition.

The following example shows how to create a left outer join on the **Salesperson/Purchaser** and **Sales Header** tables by using a SQL statement.

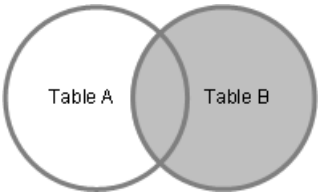
```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser" LEFT OUTER JOIN "Sales Header"
  ON "Salesperson/Purchaser".Code = "Sales Header"."Salesperson Code"
```

RightOuterJoin

A `RightOuterJoin` resembles the inner join except that the resulting dataset set contains every record from the lower data item table, even if a record does not have a matching value in the upper data item for columns that are linked by the DataItemLink Property.

- For each record in the lower data item, a row is added in the dataset that combines columns from the lower and upper data item tables.
- When a record in the lower data item table has no matching record in the upper data item table, columns coming from the upper data item table have null values.

The following illustration shows a `RightOuterJoin` type between tables A and B. The shaded area indicates the records that are included in the resulting dataset.



Dataset Example

The following table shows the resulting dataset for a `RightOuterJoin` between the **Salesperson/Purchaser** table and **Sales Header** table in the sample query. The **Sales Header** table is considered the right table.

SALESPERSON	ORDER_NUMBER	SELL_TO_CUSTOMER
Annette	1000	Autohaus
Debra	2000	Blanemark
John	3000	Candoxy
null	4000	New Concept

The record for **New Concepts** in the Sales Header table does not have a matching record in the **Salesperson/Purchaser** table, so a row is included but the columns from the **Salesperson/Purchaser** table are given null values. The record for **Bart** in the **Salesperson/Purchaser** table is not included in the resulting dataset because it does not have a matching column in the **Sales Header** table.

SQL SELECT Statement for Right Outer Join

To specify a right outer join with an SQL statement, you use the RIGHT OUTER JOIN condition.

The following example shows how to create a right outer join on the **Salesperson/Purchaser** and **Sales Header** tables by using a SQL statement.

```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser" RIGHT OUTER JOIN "Sales Header"
  ON "Salesperson/Purchaser".Code = "Sales Header"."Salesperson Code"
```

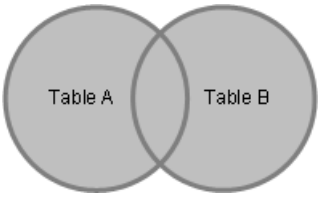
FullOuterJoin

A `FullOuterJoin` contains all the records from the upper data item table, and all records from the lower data item, including records that do not have a matching value for columns that are linked by the DataItemLink Property.

- Each pair of records from the data items that have matching column values are combined into a row in the dataset.
- Records from the upper data item table that do have a matching column are included in a row, where the columns from lower data item table have null values.

- Records from the lower data item table that do have a matching column are included in a row, where the columns from upper data item table have null values.

The following illustration shows a `FullOuterJoin` type between tables A and B. The shaded area indicates the records that are included in the resulting dataset.



Dataset Example

The following table shows the resulting dataset for a full outer join between the **Sales Header** table and **Salesperson/Purchaser** table in sample query.

SALESPERSON	ORDER_NUMBER	SELL_TO_CUSTOMER
Annette	1000	Autohaus
Bart	null	null
Debra	2000	Blanemark
John	3000	Candoxy
null	4000	New Concept

The records for **Bart** in the Salesperson/Purchaser table and **New Concepts** in the Sales Header table are included in a row, even though they not have matching values for columns.

SQL SELECT Statement for Full Outer Join

To specify a full outer join with an SQL statement, you use the FULL OUTER JOIN condition.

The following example shows how to create a full outer join on the **Salesperson/Purchaser** and **Sales Header** tables by using a SQL statement.

```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser" FULL OUTER JOIN "Sales Header"
  ON "Salesperson/Purchaser".Code = "Sales Header"."Salesperson Code"
```

CrossJoin

A `CrossJoin` contains rows that combine each row from the upper data item table with each row from a lower data item table. Cross joins are also called Cartesian products. A cross join does not apply any comparisons between columns of data items, so the DataItemLink Property is left blank.

Dataset Example

The following table shows the resulting dataset for a `CrossJoin` between the **Sales Header** table and **Salesperson/Purchaser** table in sample query.

SALESPERSON	ORDER_NUMBER	SELL_TO_CUSTOMER
Annette	1000	Autohaus

SALESPERSON	ORDER_NUMBER	SELL_TO_CUSTOMER
Annette	2000	Blanemark
Annette	3000	Candoxy
Annette	4000	New Concept
Bart	1000	Autohaus
Bart	2000	Blanemark
Bart	3000	Candoxy
Bart	4000	New Concept
Debra	1000	Autohaus
Debra	2000	Blanemark
Debra	3000	Candoxy
Debra	4000	New Concept
John	1000	Autohaus
John	2000	Blanemark
John	3000	Candoxy
John	4000	New Concept

SQL SELECT Statement for Cross Join

To specify a cross join with a SQL statement, you can do either of the following:

- Use the CROSS JOIN condition
- Create an implicit join, which has no join condition, without using a WHERE clause

The following examples shows how to create a cross join of the **Salesperson/Purchaser** and **Sales Header** tables by using a SQL statement.

```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser" CROSS JOIN "Sales Header"
```

```
SELECT "Salesperson/Purchaser".Name, "Sales Header"."No.", "Sales Header"."Sell-to Customer Name"
FROM "Salesperson/Purchaser", "Sales Header"
```

See Also

Query Object
Filtering Queries

Filtering in Query Objects

2/17/2021 • 8 minutes to read • Edit Online

You specify filters in a query to restrict the data in the resulting dataset. A filter applies conditions on fields in a table that is associated with the query. For a field to be included in the resulting dataset, a field must meet the conditions of the filter.

Overview

There are different ways to filter on fields of a query. You can set up filters on a field directly in the query object or use the AL filter methods that are outlined in the following table.

FILTER	DESCRIPTION
Filter directly on a data item in query object	You can set the DataItemTableFilter property of a data item to filter on a field in the table of the data item. You can apply the filter to any field in the table, not just fields that are defined as columns in the resulting dataset. A data item filter can't be overwritten from AL code. See Filtering on data items in a query object.
Filter directly on a column in a query object	You can set the ColumnFilter property of a `column` control to filter on the source field of the column. A filter on a column can be overwritten by the SETFILTER and SETRANGE methods from AL code. See Filtering on columns and filter rows in query object.
Add a filter row to a query object	A filter row lets you add a filter on a field that will not be included in the resulting dataset, but can be changed from AL code. To set up a row filter add a `filter` control referencing the filed that you want to filter and then set its ColumnFilter Property. A filter row is like a data item filter except a filter on a filter row can be overwritten by the SETFILTER and SETRANGE methods from AL code. See Filtering on columns and filter rows in query object.
Use SETFILTER or SETRANGE method calls	You can call the SETFILTER method method from AL code to set a filter on a field that is exposed through a column or filter row. The filter that is set by the **SETFILTER** method will overwrite any filter that is applied to a column or filter row on the same field by the ColumnFilter Property. You can call the SETRANGE method method from AL code to set a filter on a field that is exposed through a column or filter row. The filter that is set by the **SETRANGE** method will overwrite any filter that is applied to column or filter row on the same field. See Filtering using SETFILTER and SETRANGE methods.

Filtering on data items in a Query object

To specify filters on a data item, you set the DataItemTableFilter property of a data item. **DataItemTableFilter** property has the following syntax:

```
DataItemTableFilter = String;
```

Where `String` is the filter expression.

You can apply a filter on any field in a table, not just those fields that are represented by a column in the query object.

A data item filter is static which means that it can't be overwritten by the ColumnFilter Property, `filter` controls or by the SETFILTER or SETRANGE methods in AL code. If one of these filter types is applied to the same field as the data item filter, then the filters are combined. In logical terms, this combination corresponds to an "AND" operation. For example, if the data item filter applies a filter on a field to include values greater than 10 (>10) and a column filter applies a filter on the same field to include values less than fifty (<50), then the resultant filter includes values that are greater than 10 and less than fifty (10< value <50).

The DataItemTableFilter Property corresponds to a WHERE clause in an SQL SELECT statement. For more information, see Equivalent SQL SELECT Statements for Query Filters.

Example

The following query object links table `18 Customer` and table `37 Sales Line` to get the number of line items in each sales for customers. The **DataItemTableFilter** property is used to only include rows in which the number of line items is greater than 10.

```
query 50100 "Customer_Sales_Quantity"
{
    QueryType = Normal;

    elements
    {
        dataitem(C; Customer)
        {
            column(Customer_Number; "No.")
            {
            }

            column(Customer_Name; Name)
            {
            }

            dataitem(SL; "Sales Line")
            {
                DataItemLink = "Sell-to Customer No." = c."No.";
                SqlJoinType = InnerJoin;
                DataItemTableFilter = Quantity = filter(> 10);

                column(Qty; Quantity)
                {
                }
            }
        }
    }
}
```

Filtering on columns and filter rows in query object

Unlike data item filters, filters on a column or filter row are dynamic and can be overwritten from AL code at runtime by a call to the SETFILTER or SETRANGE method, if the method sets a filter on the same field.

You use filters on a column to filter on fields that are included in the dataset. To apply a filter on a column, you set the ColumnFilter Property of the column. You can apply a filter on any column, including aggregated columns that are applied an aggregate method by the Method Property. The **ColumnFilter** property has the following syntax:

```
ColumnFilter = String;
```

where `String` is the filter expression.

Adding a filter row

Use a filter row when you want to filter the query on a field, but you don't want to include the field in the dataset. For example, you might want to filter a date field on a specific date, but you don't want to include the date in the dataset. To set up a filter row, first add a `filter` element that specifies the table field on which you want to filter, Then, add the **ColumnFilter** property to set the conditions of the filter.

Example

The following query object links the `Customer` table and the `Sales Line` table and retrieves the total quantity of line items ordered for each customer. The query includes the following filters.

- A filter on the `Qty` column to include only records from the `Sales Line` table where the total quantity is less than 50.
- A filter on filter row for the `Location Code` field of the Sales Line table that includes only records where the location code is WHITE.

```
query 50100 "Customer_Sales_Quantity"
{
    QueryType = Normal;

    elements
    {
        dataitem(C; Customer)
        {
            column(Customer_Number; "No.")
            {
            }

            column(Customer_Name; Name)
            {
            }

            dataitem(SL; "Sales Line")
            {
                DataItemLink = "Sell-to Customer No." = c."No.";
                SqlJoinType = InnerJoin;
                DataItemTableFilter = Quantity = filter(> 10);

                column(Qty; Quantity)
                {
                    Method = Sum;
                    ColumnFilter = Qty = filter(< 50);
                }

                filter(Location_Code; "Location Code")
                {
                    ColumnFilter = Location_Code = const('White');
                }
            }
        }
    }
}
```

In an SQL SELECT statement, filters on a column or filter row that don't apply an aggregate method, as with the `Location_Code` filter row in the example, would correspond to a WHERE clause. Filters on a columns or filter rows that do apply a totals method, as with the `Quantity` column in the example, would correspond to a HAVING clause. For more information, see Equivalent SQL SELECT Statements for Query Filters.

Filtering using SETFILTER and SETRANGE methods

AL code includes the SETFILTER and SETRANGE methods that you can use to apply a filter on a field that is represented as a column or filter row in a query. The **SETFILTER** and **SETRANGE** methods enable you to set filters programmatically on a query at runtime. You use the **SETRANGE** method to filter on a range of values in a column or filter row. The **SETFILTER** method is more versatile than the **SETRANGE** method and enables you to filter a field based on a filter expression.

These methods will overwrite any filter on the same field that is set on a column or filter row by the `ColumnFilter` property. If a **SETFILTER** or **SETRANGE** method filters on the same field as a filter on a data item, as specified by the DataItemTableFilter Property, then the method filter and **DataItemTableFilter** property filter are combined.

Calling the SETFILTER and SETRANGE methods

You can call the **SETFILTER** and **SETRANGE** method from the AL code of the Business Central object that runs the query object or from the OnBeforeOpen Trigger of the query object.

To call the **SETFILTER** method, you use the following code.

```
Query.SetFilter(Column, String)
```

To call the **SETRANGE** method, you use the following code.

```
Query.SetRange(Column, FromValue, ToValue)
```

where:

- `Query` is a variable of the Query type that specifies the query object.
- `Column` is the name of the column or filter row as defined by its Name property.
- `String` is the filter expression.
- `FromValue` is the lower value of the range.
- `ToValue` is the higher value of the range.

For more information about these methods and important behavior, see SETFILTER method and SETRANGE method (Query).

Example

Referring to the query example in the previous sections, you can add the following code to the **OnBeforeOpen** trigger of the query object to change the filters on the `Quantity` column and the `Location\_Code` filter row to include quantities of in the range of 10 to 50 and a location code of RED.

```
trigger OnBeforeOpen()
begin
    currQuery.SETRANGE(Qty, 10, 50);
    currQuery.SETFILTER(Location_Code, '=RED');
end;
```

Equivalent SQL SELECT Statements for Query Filters

If you're familiar with SQL, then it is helpful to know how filtering in Business Central queries relates to SQL statements. To specify filters in an SQL statement, you use WHERE and HAVING clauses. The WHERE clause filters on fields. The HAVING clause filters on the results that have aggregated values as applied by of a totals method.

The following example shows the corresponding SQL SELECT statement for the previous data item filter example that links the `Customer` and `Sales Line` tables and filters on the `Quantity` field.

```
SELECT Customer."No.", Customer.Name, "Sales Line".Quantity
FROM Customer LEFT OUTER JOIN "Sales Line"
ON Customer."No." = "Sales Line".Sell-to Customer No.
WHERE "Sales Line"."Quantity" > 10
```

The following example shows the corresponding SQL SELECT statement for the previous column and filter row example that links the `Customer` and `Sales Line` tables and filters on the `Location Code` field and the total sum of the `Quantity` field.

```
SELECT Customer."No.", Customer.Name, SUM("Sales Line".Quantity) as Qty
FROM Customer LEFT OUTER JOIN "Sales Line"
  ON Customer."No." = "Sales Line".Sell-to Customer No.
WHERE "Sales Line"."Location Code" = WHITE
GROUP BY Customer."No."
HAVING Qty  50
```

See Also

Query Object
Aggregating Data
SETFILTER method
SETRANGE method

Aggregating Data in Query Objects

2/17/2021 • 6 minutes to read • Edit Online

In a query object, you can use the Method property to do a calculation on the fields of a column and return the calculated value in the dataset. For example, you can sum all the fields in a column or find the average value. The `Method` property is set on `column` controls and can be set to any of the following aggregate methods.

AGGREGATE METHOD	DESCRIPTION
Sum	Calculates the sum of the values of the field in the designated column for all records that are selected as part of the grouped set.
Average	Calculates the average value of the field in the designated column for all records that are selected as part of the grouped set. When averaging fields that have an integer data type (such as `Integer` or `BigInteger`), integer division is used. The result isn't rounded, and the remainder is discarded. For example, 5÷2=2 instead of 2.5 (or 2 1/2).
Min	Retrieves the lowest value of the field in the designated column for all records that are selected as part of the grouped set.
Max	Retrieves the highest value of the field in the designated column for all records that are selected as part of the grouped set.
Count	Returns the number of records that are selected as part of the grouped set.

Setting up an aggregate method for a query column

Except for the `Count` method, you can only use an aggregate method (`Sum`, `Average`, `Min`, and `Max`) on a field that has a numeric data type of Decimal, Integer, BigInteger, or Duration. To set up an aggregate on a column, you set the column's Method Property.

```
column(Name; Field)
{
    Method = Sum|Average|Min|Max|Count;
}
```

Setting an aggregate method on a column will automatically group the resultant data set by the other columns in the query. Records that have matching values for the other columns are grouped together into a single row in the results. The aggregate method is then applied against the group and a summary value returned in the row. It's similar to the GROUP BY clause in SQL SELECT statements (see Creating Queries with Aggregates in SQL).

The aggregate methods and grouping are further explained in the following sections.

Sample Query

The following sample query object retrieves the number of line items in every sales order for each customer. The query links the **Customer** table and the **Sales Line** table. In its current state, the `Method` property is commented out so it doesn't implement any aggregate method.

```
query 50101 "Customer_Sales_Quantity"
{
    QueryType = Normal;
    // Sorts the results in descending order
    OrderBy = descending(Qty);

    elements
    {
        dataitem(C; Customer)
        {
            column(Customer_Number; "No.")
            {
            }

            column(Customer_Name; Name)
            {
            }

            dataitem(SL; "Sales Line")
            {
                DataItemLink = "Sell-to Customer No." = c."No.";
                SqlJoinType = InnerJoin;

                column(Qty; Quantity)
                {
                    // Change the value of the property to perform a different aggregate method on grouped
columns:
                    // Sum, Average, Max, Min, or Count
                    // Method = Sum|Average|Min|Max|Count;
                }
            }
        }
    }
}
```

The following table represents a simplified version of the resulting dataset for the sample query.

CUSTOMER_NUMBER	CUSTOMER_NAME	QTY
20000	Selangorian Ltd.	400
30000	Blanemark Hifi	350
20000	Selangorian Ltd.	300
40000	Deerfield Graphics	250
20000	Selangorian Ltd.	200
30000	Blanemark Hifi	150

The following sections explain how you can modify the query to implement the different aggregate methods by changing the value of the Method property.

Sum

The `Sum` method adds the values of all fields for the specified column within a group. To set up a `Sum` method on the **Quantity** column of the sample query, set the `Method` property to `Sum`. The query is automatically grouped by the **No.** and **Name** columns.

```
...
column(Qty; Quantity)
{
    Method = Sum;
}
...
```

Looking at the sample query, you can use `Sum` method to get the total number of items in sales orders for each customer. The following table illustrates the resulting dataset for the query.

CUSTOMER_NUMBER	CUSTOMER_NAME	QTY
20000	Selangorian Ltd.	900
30000	Blanemark Hifi	500
40000	Deerfield Graphics	250

Average

The `Average` method calculates the average value of the fields in the column within a group. To set up an Average method on the **Quantity** column of the sample query, set the `Method` property to `Average`. The query is automatically grouped by the **No.** and **Name** columns:

```
...
column(Qty; Quantity)
{
    Method = Average;
}
...
```

Looking at the sample query, you can use `Average` method to get the average number of items in sales orders for each customer. The following table illustrates the resulting dataset for the query.

CUSTOMER_NUMBER	CUSTOMER_NAME	QTY
20000	Selangorian Ltd.	300
30000	Blanemark Hifi	250
40000	Deerfield Graphics	250

Min

The `Min` method retrieves the lowest value of fields in the column within a group. To set up a `Min` method on the **Quantity** column of the sample query, set the `Method` property to `Min`. The name of the **Quantity** column automatically changes to **Min_Quantity** and the query is automatically grouped by the **No.** and **Name**

columns:

```
...
column(Qty; Quantity)
{
    Method = Min;
}
...
```

Looking at the sample query, you can use `Min` method to get the least number of items in sales orders for each customer. The following table illustrates the resulting dataset for the query.

CUSTOMER_NUMBER	CUSTOMER_NAME	QTY
40000	Deerfield Graphics	250
20000	Selangorian Ltd.	200
30000	Blanemark Hifi	150

Max

The `Max` method retrieves the highest value of fields in the column within a group. To set up a `Max` method on the **Quantity** column of the sample query, set the `Method` property to `Max`. The name of the **Quantity** column automatically changes to **Max_Quantity** and the query is automatically grouped by the **No.** and **Name** columns:

```
...
column(Qty; Quantity)
{
    Method = Max;
}
...
```

Looking at the sample query, you can use `Max` method to get the greatest number of items in sales orders for each customer. The following table illustrates the resulting dataset for the query.

CUSTOMER_NUMBER	CUSTOMER_NAME	QTY
20000	Selangorian Ltd.	400
30000	Blanemark Hifi	350
40000	Deerfield Graphics	250

Count

The `Count` method returns the number of records from the data item table that comprise a group in the dataset. Unlike the other aggregate methods, the `Count` method is not associated with a specific column. Records are identified and counted based on the primary key of the data item table. Referring to the sample query, you can use a `Count` method to get the number of open sales orders per customer.

To set up a `Count` method in the sample query, the `column` element definition cannot include a source table; only a name. Therefore, you can delete the reference to the `Quantity` field in the `column(Qty; Quantity)`

element and set the `Method` property to `Count`:

```
...
column(Qty)
{
    Method = Count;
}
```

Looking at the sample query, you can use `Count` method the number of sales orders for each customer. The following table illustrates the resulting dataset for the query.

CUSTOMER_NUMBER	CUSTOMER_NAME	QTY
20000	Selangorian Ltd.	3
30000	Blanemark Hifi	2
40000	Deerfield Graphics	1

In SQL SELECT statements, the Count method corresponds to a COUNT(*) or COUNT(field) clause.

Creating queries with aggregates in SQL

If you're familiar with SQL, then it's helpful to know how the aggregate methods in Business Central relate to SQL statements. To specify an aggregate method in an SQL statement, you add the method to the SELECT statement and then add a GROUP BY clause. To group results on columns, you add the GROUPED BY statement.

The following example shows how to use an SQL statement to create an inner join of the Customer table and Sales Line table, and a sum of items for each customer. The result is grouped by the **No.** and **Name** columns.

```
SELECT Customer."No.", Customer.Name, SUM("Sales Line".Quantity)
FROM Customer INNER JOIN "Sales Line"
  ON Customer."No." = "Sales Line"."Sell-to Customer No."
GROUP BY Customer."No.", Customer.Name
```

See Also

Method Property
Query Object
Filtering Queries
Aggregating Data
Aggregate Functions (Transact-SQL)

Retrieving Date Data in Queries

2/17/2021 • 4 minutes to read • Edit Online

When you have fields in a table that contain dates, you can use a date method to retrieve only the year, month, or day instead of including the date in the resulting dataset of a query.

Setting up a Date method on a query column

To set up a date method on a query column, set the Method Property to `Day`, `Month`, and `Year`.

> **IMPORTANT**
> You can only use a date method on fields that have a Date or DateTime data type. For additional information about how to use a date method on a field that has the DateTime data type, see Working with DateTime Data Types.

For more information about how to set up query columns and properties, see Query Object.

Sample table and query

This article uses the following sample table and query to demonstrate the different date methods.

Sample Sales Header table

The following table contains data about sales orders for customers. The **Order Date** field has the data type of Date and the format DD-MM-YYYY, where DD is the day, MM is the month, and YYYY is the year.

NO.	BILL-TO NAME	ORDER DATE
1000	Autohaus Meilberg KG	18-01-2019
5000	Autohaus Meilberg KG	21-05-2019
4000	Beef House	30-09-2017
3000	Deerfield Graphics Company	05-04-2018
3000	Deerfield Graphics Company	29-04-2018

> **NOTE**
> This is a simplified subset of the data that is found in table 36 Sales Header of the CRONUS International Ltd. demonstration database.

Sample query

The following query object retrieves data from the sample Sales Header table. The query includes a totals method that counts the total the number of records from the table included in the dataset.

```
query 50100 "Sample Data Query"
{
    QueryType = Normal;

    elements
    {
        dataitem(Sales_Header; "Sales Header")
        {
            column(Bill_to_Name; "Bill-to Name")
            {
            }

            column(Order_Date; "Order Date")
            {
                // Change the value of the property to Day, Month, or Year
                Method = Day;
            }

            column(Count_)
            {
                Method = Count;
            }
        }
    }
}
```

NOTE

A column that applies a date method is still part of the group unlike columns that apply an aggregate method.

Day method

The `Day` method retrieves the day from the date expression of a field value in the query column. The day is returned as an integer, in the range of 1 to 31, which represents the day of the month. If the day in the date expression is 0, then 1 is returned.

Example

The following table displays the resulting dataset for the sample query with the `Method` property of the `Order Date` column set to `Day`.

BILL_TO_NAME	DAY_ORDER_DATE	COUNT_
Autohaus Meilberg KG	18	1
Autohaus Meilberg KG	21	1
Beef House	30	1
Deerfield Graphics Company	5	1
Deerfield Graphics Company	29	1

Month method

The `Month` method retrieves the month from the date expression of a field value in the query column. The month is returned as an integer, in the range of 1 to 12, where 1 represents January and 12 represents

December. If the month in the date expression is 0, then 1 is returned.

Example

The following table displays the resulting dataset for the sample query with the `Method` property of the `Order Date` column set to `Month`.

BILL_TO_NAME	MONTH_ORDER_DATE	COUNT_
Autohaus Meilberg KG	1	1
Autohaus Meilberg KG	5	1
Beef House	9	1
Deerfield Graphics Company	4	2

Year method

The `Year` method gets the year from the date expression of a field value in the query column. The year is returned as an integer. If the year in the date expression is 0, then 1900 is returned.

Example

The following table displays the resulting dataset for the sample query with the `Method` property of the `Order Date` column set to `Year`

CUSTOMER NAME	YEAR_ORDER_DATE	COUNT_
Autohaus Meilberg KG	2019	2
Beef House	2017	1
Deerfield Graphics Company	2018	2

Working with DateTime Data Types

On the SQL server, date and time values are processed using Coordinated Universal Time (UTC). If your Business Central solution uses a time zone other than UTC and the field on which you apply the date method has a data type of DateTime, then there might be a difference between the date value that is returned in the dataset for the field and the actual day, month, or year for the field in the table. This occurs when the corresponding UTC date for a field falls on the next day or previous day because of the time of day and the time zone of Business Central solution. The following table includes examples of DateTime values for two time zones that will return days, months, and years in a dataset that differ from the values in the table.

TIME ZONE	DATE AND TIME IN BUSINESS CENTRAL	DAY RETURNED BY DAY METHOD	MONTH RETURNED BY MONTH METHOD	YEAR RETURNED BY YEAR METHOD
Pacific Time (UTC – 8:00:00)	12-31-2011 17:00:00	31	12	2018
Middle European Time (UTC +1:00:00)	01-01-2012 00:59:00	1	1	2019

The differences in day, month, or year occur because when a date and time value is retrieved from the Business

Central database table, it is converted from the regional settings of the Business Central solution to the UTC date and time. The day, month, or year is calculated on the SQL server, and then returned to the query dataset as an integer, which does not consider the regional settings of the Business Central solution.

To avoid this condition, you should use the date method on fields that have a Date data type instead of a DateTime data type whenever possible. You can also return the DateTime value and implement post processing for the day, month, and year as needed.

See Also

Query Objects
Aggregating Data in Query Objects
Method Property

Using Queries Instead of Record Variables

2/17/2021 • 2 minutes to read • Edit Online

In scenarios where you want to read records from multiple tables, it can be a good idea to use a query instead of implementing code with record variables. Using a query can improve performance and also simplify the AL code that is required to perform the operation.

Code Example Using Record Variables

The following AL code shows an example of using record variables to retrieve and handle records from two tables. You could potentially use this code to track item movement. The code uses two record variables, `Item` and `ItemLedgerEntry`, to retrieve the first five records from table **27 Item** and table **32 Item Ledger Entry** where the **Entry Type** field equals **Sale**. The retrieved records are passed to and handled by the `OutputData` method.

```
begin
    count := 0;
    if Item.FINDSET then
        repeat
            PrevDate := 0D;
            TotalQty := 0;
            ItemLedgerEntry.SETCURRENTKEY("Item No.", "Posting Date");
            ItemLedgerEntry.SETRANGE("Item No.", Item."No.");
            ItemLedgerEntry.SETRANGE("Entry Type",
                ItemLedgerEntry."Entry Type"::Sale);
            if ItemLedgerEntry.FINDSET then
                repeat
                    if (ItemLedgerEntry."Posting Date" <> PrevDate) and (PrevDate <> 0D) then begin
                        OutputData(1, Item."No.", Item.Description, PrevDate, -TotalQty);
                        TotalQty := 0;
                        count := count + 1;
                    end;
                    PrevDate := ItemLedgerEntry."Posting Date";
                    TotalQty := TotalQty + ItemLedgerEntry.Quantity;
                until (ItemLedgerEntry.NEXT = 0) or (count >= 4);
            if PrevDate <> 0D then begin
                OutputData(1, Item."No.", Item.Description, PrevDate, -TotalQty);
                count := count + 1;
            end;
        until (Item.NEXT = 0) or (count >= 4);
end;
```

Corresponding Query Implementation

The following AL code represents a query object and additional code that will return the same results as the previous example that uses record variables.

```
query 50100 "Item Movements Query"
{
    QueryType = Normal;

    elements
    {
        // This dataitem corresponds to the `Item` record variable in the record variable example.
        dataitem(Item; Item)
        {
            column(No_; "No.")
            {
            }
            column(Description; Description)
            {
            }

            // This dataitem corresponds to the `ItemLedgerEntry` record variable in the record variable
example.
            dataitem(Item_Ledger_Entry; "Item Ledger Entry")
            {
                // The DataItemLink and  SqlJoinType settings correspond to the
`ItemLedgerEntry.SETRANGE("Item No.",Item."No.");` statement in the record variable example.
                DataItemLink = "Item No." = Item."No.";
                SqlJoinType = InnerJoin;

                filter(Entry_Type; "Entry Type")
                {
                }
                column(Posting_Date; "Posting Date")
                {
                }

                // The SUM corresponds to the `TotalQty := TotalQty + ItemLedgerEntry.Quantity;` statement
in the record variable example.
                column(Sum_Quantity; Quantity)
                {
                    Method = Sum;
                }
            }
        }
    }
}
```

Add the following code to a codeunit that will run the query.

```
var
    ItemMovements: Query "Item Movements"
begin
    ItemMovements.TopNumberOfRows(5);
    ItemMovements.SetRange(Entry_Type,ItemMovements.Entry_Type::Sale);
    ItemMovements.Open;
    while ItemMovements.Read do
      OutputData(2,
ItemMovements.Item_No,ItemMovements.Description,ItemMovements.Posting_Date,ItemMovements.Sum_Quantity);
end;
```

The `ItemMovements.TOPNUMBEROFROWS(5);` statement will include only the first 5 records in the resulting dataset and corresponds to implementing the `count` variable in the record-based code example.

The `OutputData` method performs the same operations as the `OutputData` method in the record variable example.

See Also

Accessing Columns of a Query Dataset

2/17/2021 • 2 minutes to read • Edit Online

If the query is in the reading state, you can retrieve the value of columns in the current active row of the dataset by using the following syntax in AL.

Syntax

```
ColumnValue := QueryVariable.ColumnName
```

- *QueryVariable* is a variable of the Query data type that specifies the query object.
- *ColumnName* is the name of the column in the query object.

Return Value

The data type of the field that is used by the column, unless the column applies a totaling method as specified by the Method Property. If the column applies a totaling method, then data type is an integer for the `Count` method and a decimal for all other totaling methods.

Returns the value of the column in the current active row.

Remarks

A column of a row can only be accessed after the query has been opened by using a call to the Open Method followed by a call to the Read Method. The current active row is the row that has been included in the query variable after the last call to Read Method.

Example

This example demonstrates how to access a column of a query dataset. When the query is run, each row in the dataset is read and message box is displayed that contains the value of a column in the row.

The following query object links table **18 Customer** and table **37 Sales Line**.

```
query 50123 "Customer_Sales_Quantity"
{
    QueryType = Normal;
    // Sets the results to only include the top forts the results in descending order
    TopNumberOfRows = 5;
    OrderBy = descending(Qty);

    elements
    {
        dataitem(C; Customer)
        {
            column(Customer_Number; "No.")
            {
            }

            column(Customer_Name; Name)
            {
            }

            dataitem(SL; "Sales Line")
            {
                DataItemLink = "Sell-to Customer No." = c."No.";
                SqlJoinType = InnerJoin;

                column(Qty; Quantity)
                {
                }
            }
        }
    }
}
```

The following codeunit opens the query, reads each row of dataset, and then displays a message that has the content of each row.

```
codeunit 50100 QueryColumnAccess
{
    trigger OnRun()
    begin

        // Sets a filter to display only sales quantities greater than 20.
        MyQuery.SETFILTER(Qty, '>20');
        // Runs the query.
        MyQuery.OPEN;
        // Reads each row in the dataset and displays a message with column values.
        // Stops reading when there are no more rows remaining in the dataset (READ is FALSE).
        while MyQuery.READ do begin
            MESSAGE(Text000, MyQuery.Customer_Name, MyQuery.Qty);
        end;
        // Closes the query.
        MyQuery.CLOSE;

    end;

    var
        MyQuery: Query "Customer_Sales_Quantity";

        Text000: Label 'Customer name = %1, Quantity = %2.';
}
```

See Also

Query Object

API Query Type

2/17/2021 • 2 minutes to read • Edit Online

Queries of the type `API` are used to generate web service endpoints and this type of query cannot be used to display data in the user interface. A query of the API type can be used to join data from different data sources. The data can only be viewed. When creating this query type, you must specify a number of properties that provide information for the web service endpoint. Use the snippet `tquery - Query of type API` to get the right template and the list of these properties automatically filled in.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Example of the API query type

The following query example publishes an API available at: `../contoso/app1/v1.0/companies({id})/customerSales`. The `APIVersion` can be specified as one version, or a list of versions, if the API is supported through multiple versions.

```
query 20000 "APIV1 - Customer Sales"
{
    QueryType = API;
    APIPublisher = 'contoso';
    APIGroup = 'app1';
    APIVersion = 'v1.0';
    Caption = 'customerSales', Locked = true;
    EntityName = 'customerSale';
    EntitySetName = 'customerSales';

    elements
    {
        dataitem(QueryElement1; Customer)
        {
            column(customerId; Id)
            {
                Caption = 'Id', Locked = true;
            }
            column(customerNumber; "No.")
            {
                Caption = 'No', Locked = true;
            }
            column(name; Name)
            {
                Caption = 'Name', Locked = true;
            }
            dataitem(QueryElement10; "Cust. Ledger Entry")
            {
                DataItemLink = "Customer No." = QueryElement1."No.";
                SqlJoinType = LeftOuterJoin;
                DataItemTableFilter = "Document Type" = FILTER (Invoice | "Credit Memo");
                column(totalSalesAmount; "Sales (LCY)")
                {
                    Caption = 'TotalSalesAmount', Locked = true;
                    Method = Sum;
                }
                filter(dateFilter; "Posting Date")
                {
                    Caption = 'DateFilter', Locked = true;
                }
            }
        }
    }
}
```

See Also

AL Development Environment
API Page Type
APIPublisher Property
APIGroup Property
APIVersion Property
EntityName Property
EntitySetName Property Query Object
Developing Extensions

Interfaces in AL

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

An interface in AL is similar to an interface in any other programming language; it is a syntactical contract that can be implemented by a non-abstract method. The interface is used to define which capabilities must be available for an object, while allowing actual implementations to differ, as long as they comply with the defined interface.

This allows for writing code that reduces the dependency on implementation details, makes it easier to reuse code, and supports a polymorphic way of calling object methods, which again can be used for substituting business logic.

The interface declares an interface name along with its methods, and codeunits that implement the interface methods, must use the `implements` keyword along with the interface name(s). The interface itself does not contain any code, only signatures, and cannot itself be called from code, but must be implemented by other objects.

The AL compiler checks to ensure that implementations adhere to assigned interfaces.

You can declare variables as a given interface to allow passing objects that implement the interface, and then call interface implementations on the passed object in a polymorphic manner.

Snippet support

Typing the shortcut `tinterface` will create the basic layout for an interface object when using the AL Language extension in Visual Studio Code.

Interface example

The following example defines an interface `IAddressProvider`, which has one method `getAddress` with a certain signature. The codeunits `CompanyAddressProvider` and `PrivateAddressProvider` both implement the `IAddressProvider` interface, and each define a different implementation of the `getAddress` method; in this case a simple variation of address value.

The `MyAddressPage` is a simple page with an action that captures the choice of address and calls, based on that choice, an implementation of the `IAddressProvider` interface.

```
interface IAddressProvider
{
    procedure GetAddress(): Text
}

codeunit 50200 CompanyAddressProvider implements IAddressProvider
{
    procedure GetAddress(): Text

    begin
        exit('Company address \ Denmark 2800')
    end;
}

codeunit 50201 PrivateAddressProvider implements IAddressProvider
{
```

```
    procedure GetAddress(): Text

    begin
        exit('My Home address \ Denmark 2800')
    end;
}

enum 50200 SendTo
{
    Extensible = true;

    value(0; Company)
    {
    }

    value(1; Private)
    {
    }
}

page 50200 MyAddressPage
{

    PageType = Card;
    ApplicationArea = All;
    UsageCategory = Administration;

    actions
    {
        area(Processing)
        {
            action(ActionName)
            {
                ApplicationArea = All;

                trigger OnAction()

                var
                    iAddressProvider: Interface IAddressProvider;

                begin
                    AddressproviderFactory(iAddressProvider);
                    Message(iAddressProvider.GetAddress());

                end;
            }

            action(SendToHome)
            {
                ApplicationArea = All;

                trigger OnAction()

                begin
                    sendTo := sendTo::Private
                end;
            }

            action(SendToWork)
            {
                ApplicationArea = All;

                trigger OnAction()

                begin
                    sendTo := sendTo::Company
                end;
            }
        }
```

```
    }

    local procedure AddressproviderFactory(var iAddressProvider: Interface IAddressProvider)
    var
        CompanyImplementer: Codeunit CompanyAddressProvider;
        PrivateImplementer: Codeunit PrivateAddressProvider;
    begin

        if sendTo = sendTo::Company then
            iAddressProvider := CompanyImplementer;

        if sendTo = sendTo::Private then
            iAddressProvider := PrivateImplementer;
    end;

    var
        sendTo: enum SendTo;
}
```

See Also

Codeunit Object
Extensible Enums

XMLport Overview

2/17/2021 • 2 minutes to read • Edit Online

An XMLport is composed of the following items:

- An XMLport object
- An XMLport schema
- A Request page
- Properties, triggers, and code

XMLport object

You create an XMLport object in the AL Language development environment to define the schema of an XML document. You can export and import data between an external source and Dynamics 365 Business Central with XMLports. For more information, see XMLport Object.

XMLport schema

In order to define the underlying structure of the imported or exported document, you use the XMLport schema. An XMLport schema determines which data is exported from or imported to Dynamics 365 Business Central database tables and the format and structure of the files used. You build the XMLport schema by adding nodes. For more information, see Defining a XMLport Schema.

Request page

Request pages are dialog boxes that enable the user to set a filter on the data, sort the data, or choose whether to export or import the data. For more information, see Request Pages.

Unlike report request pages, XMLport request pages cannot be bookmarked by users from the user interface.

XMLport properties, triggers, and code

XMLport objects include triggers, methods, and properties that can be used to work with the object. For more information, see XMLport Data Type and XMLport Triggers.

When you design XMLports, you must set the value of the Format Property (XMLports) and the Direction Property. The Format Property (XMLports) indicates the type of file that you want to import or export and the Direction Property value indicates whether the XMLport will be used for import or export.

For more information about data consistency and validation against possible errors when using XMLports, see the blog post Importing and exporting valid data using XMLports in Dynamics 365 Business Central.

See Also

XMLport Object
Defining a XMLport Schema
Using Namespaces with XMLports
Request Pages
XMLport Data Type
XMLport Triggers

XMLport Object

2/17/2021 • 2 minutes to read • Edit Online

XMLports are used to export and import data between an external source and Dynamics 365 Business Central. Sharing data between different computer systems is seamless when it is shared in an XML format. Working with XML files can be tedious so the details of how the XML file is handled are encapsulated in XMLports.

To use an XMLport to import or export data, you first create an XMLport object. Once created, you can run the XMLport from a page or codeunit object.

You can design XMLports to include a request page, which is a dialog box that enables the user to set a filter on the data, sort the data, or choose whether to export or import the data. For more information about request pages, see Request Pages.

XMLport example

The following example shows a page extension of the **Permission Sets** page that adds an action to the specified page calling the XMLport **ExportPermissionSet**. The XMLport exports the permission set data to an XML file.

```
pageextension 50111 PermissionSetExporter extends "Permission Sets"
{
    actions
    {
        addafter(Permissions)
        {
            action(ExportPermissionSet)
            {
                Promoted = true;
                PromotedCategory = New;
                trigger OnAction();
                begin
                    Xmlport.Run(50112, false, false);
                end;
            }
        }
    }
}

xmlport 50112 ExportPermissionSet
{
    Format = xml;

    schema
    {
        textelement(PermissionSets)
        {
            tableElement(PSet; "Aggregate Permission Set")
            {
                SourceTableView = WHERE ("App Name" = FILTER (<> ''));
                XmlName = 'PermissionSet';
                fieldattribute(RoleID; pset."Role ID") { }
                fieldattribute(RoleName; pset.Name) { }
                tableelement(P; "Tenant Permission")
                {
                    XmlName = 'Permission';
                    LinkTable = pset;
                    LinkFields = "Role ID" = FIELD ("Role ID");
```

```
                    textelement(ObjectType)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Object Type";
                            ObjectType := format(int);
                        end;
                    }
                    textelement(ObjectID)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Object ID";
                            ObjectID := format(int);
                        end;
                    }
                    textelement(ReadPermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Read Permission";
                            ReadPermission := format(int);
                        end;
                    }
                    textelement(InsertPermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Insert Permission";
                            InsertPermission := format(int);
                        end;
                    }
                    textelement(ModifyPermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Modify Permission";
                            ModifyPermission := format(int);
                        end;
                    }
                    textelement(DeletePermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Delete Permission";
                            DeletePermission := format(int);
                        end;
                    }
                    textelement(ExecutePermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Execute Permission";
                            ExecutePermission := format(int);
                        end;
```

```
                    }
                    textelement(SecurityFilter)
                    {
                        trigger onbeforePassvariable();
                        begin
                            SecurityFilter := format(p."Security Filter");
                        end;
                    }
                }
            }
        }
    }
}
```

See Also

Developing Extensions
AL Development Environment
XMLport Overview
Using Namespaces with XMLports
Page Extension Object
Report Object

Defining an XMLport schema

2/17/2021 • 2 minutes to read • Edit Online

You use an XMLport object to export and import data between an external source and Dynamics 365 Business Central. The schema determines which and how data is extracted from or inserted into the Dynamics 365 Business Central database tables through an XMLport. For more information, see XMLport Object.

You build the XMLport schema from nodes. A node can be:

- A text element
- A text attribute
- A table element
- A field element
- A field attribute

You nest nodes inside other nodes in order to define the structure of the XMLport schema. Use the following keywords to define the structure.

KEYWORD	DESCRIPTION
`textelement`	Defines a new text element in the XMLport. It is used for XML elements that do not map to a database item or when the database does not need the information.
`textattribute`	Defines a new text attribute in the XMLport. It is used for XML attributes that do not map to a database item or when the database does not need the information.
`tableelement`	Defines a new table element in the XMLport. It is used for XML elements that map to a table in the database, which is specified in the SourceTable Property. When the XMLport is run, the code nested inside the table element is iterated for all records in the underlying table.
`fieldelement`	Defines a new field element in the XMLport. It is used for XMLport elements that map to a field in the database, which is specified in the SourceField Property. You must specify it inside the parent table element of the field.
`fieldattribute`	Defines a new field attribute in the XMLport. It is used for XMLport attributes that map to a field in the database, which is specified in the SourceField Property.

NOTE

There can only be one `<root>` node, which must be an element. If the Format Property is set to **Xml**, it must be a `textelement` node.

There can be several attributes for a single element and their order does not matter. Attribute nodes must be specified inside the element nodes they refer to and before other element nodes. They cannot have nested

element nodes.

Snippet support

Typing the shortcut `txmlport` will create the basic layout for an XMLport object when using the AL Language extension in Visual Studio Code.

Example

The following example adds the `Customer` table as a table element, the `Address` field as a field element and the `County` and `City` fields as field attributes.

```
schema
{
    textelement(Customers)
    {
        tableelement(Customer; Customer)
        {
            fieldelement(Address; Customer.Address)
            {
                fieldattribute(County; Customer.County){}
                fieldattribute(City; Customer.City){}
            }
        }
    }
}
```

For more information about designing XMLports, see XMLport Overview.

For more information about data consistency and validation against possible errors when using XMLports, see the blog post Importing and exporting valid data using XMLports in Dynamics 365 Business Central.

For information about the use of namespaces in XMLports see Using Namespaces with XMLports.

See Also

XMLport Object
XMLport Data Type
Using Namespaces with XMLports
XMLport Triggers
Request Pages
XMLport Overview

Using Namespaces with XMLports

2/17/2021 • 2 minutes to read • Edit Online

The external system that provides or consumes AL Language development environment data as XML might require that the XML documents include namespaces. Namespaces are used to avoid element name conflicts. In these cases, you must add namespaces on the XMLport to make it compatible with the XML schema that is used by the external system.

> **NOTE**
>
> Namespace-related properties are only available when the Format Property is set to **Xml**.

For example, the following code is a portion of a simple XML document for transferring sales order information. The XML includes namespaces for mapping fields from the `Sales Header` table.

```
<?xml version="1.0" encoding="UTF-16"?>
<Root xmlns="urn:bc:schema:all" xmlns:bcField="urn:bc:schema:field" xmlns:bcTable="urn:bc:schema:table">
  <bcTable:SalesHeader>
    <bcField:DocType>Order</bcField:DocType>
    <bcField:DocNo>101005</bcField:DocNo>
    <bcField:SellToCustNo>30000</bcField:SellToCustNo>
    <bcField:SellToCustName>John Haddock Insurance Co.</bcField:SellToCustName>
    <bcField:BillToCustNo>30000</bcField:BillToCustNo>
    <bcField:BillToCustName>John Haddock Insurance Co.</bcField:BillToCustName>
  </bcTable:SalesHeader>
...
</Root>
```

Each namespace has the syntax *xmlns:prefix="namespacename"*.

- The *namespacename* is a string of characters, often referred to as a Uniform Resource Identifier (URI), which uniquely identifies an Internet resource. This is typically a Uniform Resource Locator (URL) or Universal Resource Name (URN).
- The *prefix* is a short string of characters that acts as an alias for the namespace name. The prefix is applied to specific elements in the XML document. The example includes the prefixes `bcField` and `bcTable`.
- A namespace that does not include a prefix declares the default namespace. In the example, the default namespace is urn:bc:schema:all. The default prefix is applied to all the elements that do not include a prefix.

You declare the namespaces used in the XMLport using the Namespaces Property. For each namespace, you specify a prefix and a namespace name. You can declare a default namespace by defining an empty prefix `""`. In the XML documents exported or imported by the XMLport, the namespaces declarations are only supported in the `<root>` element.

You then apply the namespaces to XMLport elements by setting the NamespacePrefix Property of the element to one of the namespace prefixes declared in the Namespaces Property. This property only applies to `textelement`, `tableelement` and `fieldelement` nodes, otherwise it will be ignored.

You can also specify a default namespace using the DefaultNamespace Property and setting the UseDefaultNamespace Property to **true**. Note that there can only be one default namespace, so if you specify

the default namespace in the Namespaces Property, you must set the DefaultNamespace Property to **false**.

See Also

XMLport Object
Namespaces Property
NamespacePrefix Property
DefaultNamespace Property
UseDefaultNamespace Property

Request Pages

2/17/2021 • 4 minutes to read • Edit Online

A request page is a page that is run before the report or XMLport starts to execute. Request pages enable end users to specify options and filters for a report and an XMLport. Request pages are defined as part of designing a Report object and an XMLport object. The syntax is shown further down in this topic. You design the filters on request pages by using the following report and XMLport properties:

PROPERTY	DESCRIPTION
RequestFilterHeading Property	Sets a caption for the request page tab that is related to a report's data item or an XMLport's table element.
RequestFilterHeadingML Property	Sets the text used as a RequestFilterHeading Property for a request page tab.
RequestFilterFields Property	Specifies which fields are automatically included on the tab of the request page that is related to a report's data item or an XMLport's table element. The user can set filters on these fields.

NOTE

Request pages for XMLports are not supported by the Business Central Web client in versions prior to Dynamics 365 Business Central 2019 release wave 2. If you try to run an XMLport with a Request page from the web client in these versions, you receive an error that the XMLport page type is not supported. Alternatively, XMLport request pages do work in the Dynamics NAV Client connected to Business Central.

By default, a request page is displayed, unless the UseRequestPage is set to `false`; then the report or XMLport will start to print as soon as it is run. In this case, end users can't cancel the report or XMLport run. It is still possible to cancel the report or XMLport, but some pages may print.

By default, without having set anything else, a request page will always display the following buttons:

- Send to
- Print
- Preview
- Cancel

Additionally, you can add more options on the request page to allow the end user to filter the data displayed.

Filtering on request pages

The fields that you define as `RequestFilterFields` are shown on the request page and can be used for filtering the data before viewing or printing the report.

NOTE

Only on the Windows client, filtering is possible even if `RequestFilterFields` is not set.

Defining the `RequestFilterFields` property in the `dataitem()` part of the report code is done as illustrated in the following code example:

```
report 50103 "Customer List"
{
    CaptionML = ENU = 'Customer List';
    RDLCLayout = 'Customer List Report.rdl'; // if Word use WordLayout property
    dataset
    {
        dataitem(Customer; Customer)
        {
            RequestFilterFields = "No.", "Search Name", "Customer Posting Group";
...
```

> **NOTE**
>
> We recommend that you add fields that the end-users of the report will frequently set filters on.

For more information about the report object, see Report Object.

Defining the `RequestFilterFields` property in the `tableelement()` part of an XMLport is done in a similar way:

```
XMLport 50104 "Export Customer List"
{
    CaptionML = ENU = 'Export customer List';
    Direction = Export;
    schema
    {
        textelement(root)
        {
            XmlName = 'Root';
            tableelement(Customer; Customer)
            {
                RequestFilterFields = "No.", "Search Name", "Customer Posting Group";
...
```

For more information about the XMLport object, see XMLport Object.

By default, for every data item in the report and table element in a XMLport, a FastTab for defining filters and sorting is created on the request page. To remove a FastTab from a request page, do not define any `RequestFilterFields` for the data item or table element and set the DataItemTableView property in a report or the SourceTableView property in an XMLport to define sorting. The request page is displayed, but there is no tab for this data item or table element.

If a `DataItemTableView` or `SourceTableView` is not defined, then end users can select a sort field and sort order at runtime.

In a complex report or XMLport that uses data from several tables, the functionality may depend on a specific key and sort order. Design your reports and XMLports so that end users can't change the sort order in a way that affects their functionality.

For data items and table elements whose source table contains calculated fields, such as amounts and quantities, the **Filter totals by:** section is automatically included on the request page, which allows you to adjust various dimensions that influence calculations.

> **TIP**
> For information about how to enter filter criteria on the request page, see Filtering in the Business Central application help.

Defining a `requestpage` section

On reports, in addition to defining the filter options by setting the `RequestFilterFields` property, you can add a `requestpage` section. In this section, you can set the SaveValues property to `true` in order to save the values that the end user enters on the request page. When the report is run again, the end user will have the option to use previously defined filters. You can also add a `layout` to the request page, specifying an **Options** section to perform checks.

> **NOTE**
> With request pages that have `SaveValues = true`, users can preview the report multiple times in the client and the request page remains open. If `SaveValues = false` or omitted, a **Preview & Close** action appears on the request page. In this case, the request page closes once the report has been previewed.

```
...
requestpage
    {
        SaveValues = true;

        layout
        {
            area(content)
            {
                group(Options)
                {
                    Caption = 'Options';
                    field(PostingDate; PostingDateReq)
                    {
                        ApplicationArea = Basic, Suite;
                        Caption = 'Posting Date';
                        ToolTip = 'Specifies the posting date for the invoice(s) that the batch job creates.
This field must be filled in.';
                    }
                }
            }
        }

        trigger OnOpenPage()
        begin
            if PostingDateReq = 0D then
                PostingDateReq := WorkDate;
        end;

        var
            PostingDateReq: Date;
    }
...
```

See Also

Report Object
XMLport Object

Reports Overview
Report Design Overview
RunRequestPage Method
RequestFilterHeading Property
RequestFilterHeadingML Property
RequestFilterFields Property
DataItemTableView

XMLport Triggers

2/17/2021 • 2 minutes to read • Edit Online

The following triggers apply to XMLports.

XMLport triggers

XMLPORT TRIGGER	RUNS
OnAfterAssignField Trigger	Runs after a field has been assigned a value and before it is validated and imported. This trigger is only used to import data.
OnAfterAssignVariable Trigger	Runs after the value defined in the XML document is assigned to the text variable. This trigger is only used to import data.
OnAfterGetField Trigger	Runs after a field is passed to the XML document. This trigger is only used to export data.
OnAfterGetRecord (XMLports) Trigger	Runs after a record is retrieved from a table and before it is exported to the XML document. This trigger is only used to export data.
OnAfterInitRecord Trigger	Runs after a record is loaded. This trigger is only used to import data.
OnAfterInsertRecord Trigger	Runs after a record has been inserted into a database table. This trigger is only used to import data.
OnAfterModifyRecord Trigger	Runs after a record has been modified. The trigger is used to import data.
OnBeforeInsertRecord Trigger	Runs after a record has been loaded and before it is inserted into a database table. This trigger is only used to import data.
OnBeforeModifyRecord Trigger	Runs before a record is modified. This trigger is used to import data.
OnBeforePassField Trigger	Runs before a field is passed to the XML document. This trigger is only used to export data.

XMLPORT TRIGGER	RUNS
OnBeforePassVariable Trigger	Runs after the source expression has been formatted into a text variable and before the text variable is passed to the XML document. This trigger is only used to export data.
OnInitXMLport Trigger	Executes when the XMLport is loaded and before any table views and filters are set.
OnPreXMLport Trigger	Runs after the table views and filters are set and before the XMLport is run.
OnPostXMLport Trigger	Runs after the XMLport is run.
OnPreXMLItem Trigger	Runs after the table is initialized and before you start exporting data to an XML object. This trigger only applies to XMLport elements that have a source type of Table. This trigger is only used to export data.

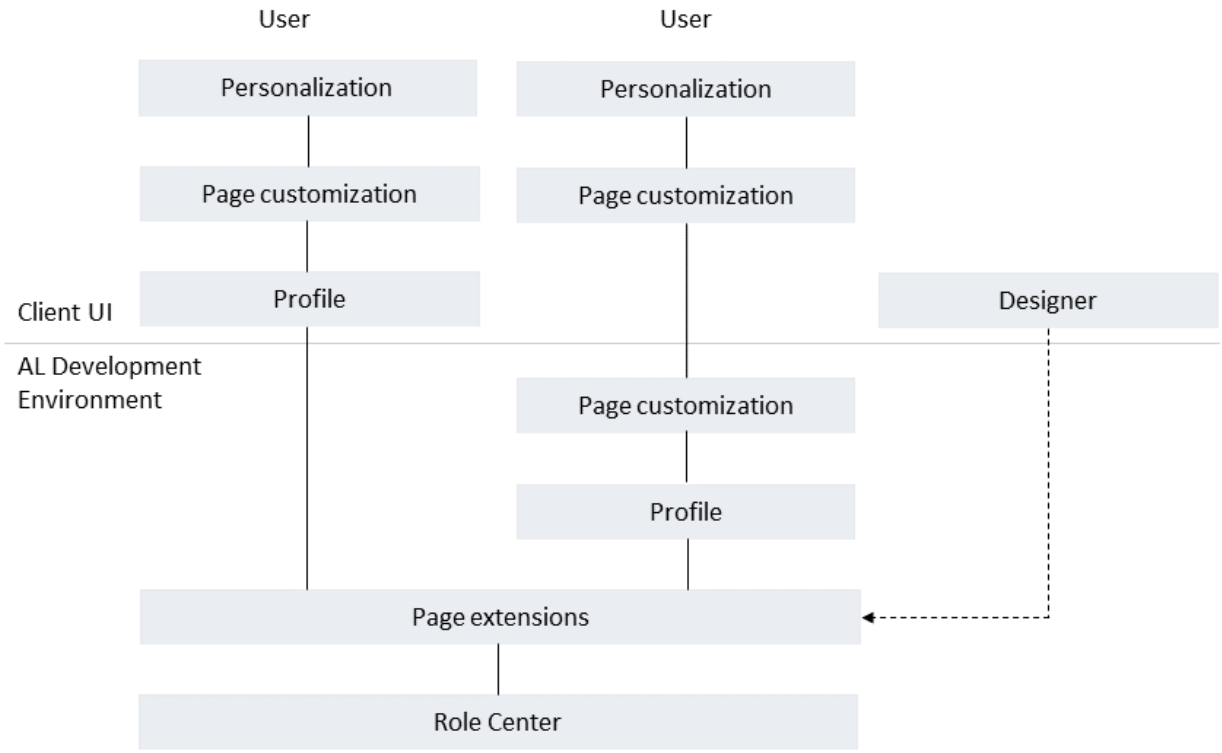
See Also

XMLPort Object
Triggers
XMLPort Properties

Customizing the User Interface for User Roles

2/17/2021 • 2 minutes to read • Edit Online

The strength of Business Central is its role-tailored experience that helps users focus on the work that is important to them. Business Central offers several features for developers, application administrators, and end-users, that can be used to customize the the pages that users work with in the client. These features customize the pages on different layers, as illustrated in the following figure. Some customization is done in AL extensions, while others can be done from the client.



Role Centers

The Role Center is first layer of customization. The Role Center is the user's entry point and home page for Business Central, displaying information that is pertinent to the user's role in the company and enabling them to easily navigate to relevant pages for viewing data and performing tasks. You can develop several different Role Centers, where each Role Center is customized to the profile of the intended users.

A Role Center is created in AL by the `rolecenter` page type.

For more information, see Designing Role Centers.

Page extensions and designer

A page extension object extends a page object by adding, moving or hiding UI elements that are defined in the page's source code. Page extensions can be created in Visual Studio Code or from the client using Designer. Changes made by page extensions affect the UI for all users, regardless of which profile they belong to.

Profiles and page customizations

A *profile* is the mechanism that makes a Role Center and its associated pages available to users in the client and enables you to build an individual experience for the specific user role. In the client, profiles are referred to as

Roles. Users sign in to the client under a specific role, which they can switch from the **My Settings** page. Different profiles can use the same Role Center. Profiles can be created as part of an extension by writing AL code or they can be created from the client by a user who has the proper permissions, typically an administrator or consultant.

On top of the profile are *page customizations*. Page customizations modify the layout of elements on specific pages. For example, you can move or hide actions, fields, columns on list, or entire parts; exactly the same modifications that can be done using personalization. The page customizations will be seen by all users of the profile. Like profiles, page customizations can be made in AL as part of an extension or from the client by modifying the profile. Customizations that are made from the client will take precedence over the customizations in AL.

For more information about profiles and page customization in AL, see Designing Profiles. For information about using the client, Customizing the Workspace for Profiles (Roles) in the Business Central Application Help.

Personalization

The last layer of customization is personalization. This is done strictly in the client by end-users for customizing their own workspaces. The changes that users make take precedent over page customizations made on the profile. The changes will only be seen by the user; not other users. For more information, see Personalizing Your Workspace in the Business Central Application Help.

See Also

Developing Extensions
AL Development Environment
Page Object
Page Extension Object
Page Extension Properties

Designing Profiles

2/17/2021 • 3 minutes to read • Edit Online

A profile is the mechanism that makes a Role Center page and its associated pages and reports available to users in the client. It enables you to build an individual experience for users based on their role in the company by customizing the pages that they use to perform the daily tasks. In the client, profiles are referred to as **Roles**. When users sign in to Business Central, they are doing so under a specific role. Users can switch the role from the **My Settings** page.

Creating profiles in AL involves two different object types: profile object and page customization object.

Profile objects

A profile object specifies an ID for the profile, a display name that appears in the client, a role center page, and the page customization objects that apply to the profile. Different profiles can use the same Role Center page.

Page customization object

A page customization object specifies layout and action modifications to a specific page object. For example, you can move actions, fields, columns on list, or entire parts. The modifications apply only to the profile that they are used with. The page customizations will be seen by all users of the profile.

To make the modifications, you use placement keywords in the `layout` and `actions` sections.

The same page customization objects can used in different profile objects. A profile object does not necessarily use any page customization objects, but a page customization has no effect without being associated with a profile object.

Example

The following example creates a profile object that uses the `Business Manager` role center page. It uses two page customization objects; one that modifies the **Business Manager Role Center** page and another that modifies the **Customer List** page.

```
profile TheBoss
{
    Description = 'This is the profile for the Boss';
    RoleCenter = "Business Manager";
    Customizations = MyCustomization1, MyCustomization1;
    Caption = 'Boss';
}

pagecustomization Configuration1 customizes "Business Manager Role Center"
{
  actions
  {
    // Hides the Customers action
    modify(Customers)
    {
      Visible = false;
    }
  }
}

pagecustomization MyCustomization customizes "Customer List"
{
    layout
    {
      {
        // Moves the Balance (LCY) column after the Phone No. column
        moveafter("Phone No.";"Balance (LCY)")
      }
    }
}
```

Using the client to create AL profiles and page customizations

Creating profiles and page customizations can also be done from the client. This will typically be done by administrators or consultants to create new profiles or fine-tune the page customizations that provided by extensions. However, as a developer, you can also leverage the client to make profiles and page customizations in AL extensions. For more information, see Using the Client to Create Profiles and Page Customizations.

Translating profiles

A profile's `Caption` and `ProfileDescription` properties appear in the client user-interface, so you will typically want to translate these into different languages. Like other objects, this is done by creating XLIFF files for the different languages from the source language. The generated XLIFF file contains a `<source>` tag for both the `Caption` and `ProfileDescription` to which you can add a `<target>` tag for the translated text. For more information, see Working with Translation Files.

For example, the following code is the content of an XLIFF file for translating the example profile mentioned above from its source language, en-US, to the target language da-DK.

```
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-transitional.xsd">
  <file datatype="xml" source-language="en-US" target-language="da-DK" original="profiles">
    <body>
      <group id="body">
        <trans-unit id="Profile 3523819904 - Property 4111922599" size-unit="char" translate="yes"
xml:space="preserve">
          <source>The Boss</source>
          <target>Chefen</target>
          <note from="Developer" annotates="general" priority="2"></note>
          <note from="Xliff Generator" annotates="general" priority="3">Profile TheBoss - Property
ProfileDescription</note>
        </trans-unit>
        <trans-unit id="Profile 3523819904 - Property 2879900210" size-unit="char" translate="yes"
xml:space="preserve">
          <source>Boss</source>
          <target>Chef</target>
          <note from="Developer" annotates="general" priority="2"></note>
          <note from="Xliff Generator" annotates="general" priority="3">Profile TheBoss - Property
Caption</note>
      </group>
    </body>
  </file>
</xliff>
```

See Also

Developing Extensions
AL Development Environment
Page Object
Page Extension Object
Page Extension Properties

Profile Object

2/17/2021 • 2 minutes to read • Edit Online

The profile object in Dynamics 365 Business Central allows you to build an individual experience for each user profile. The Profile object performs a validation to check whether the specified role center page exists, and page customization objects exists, when you define a new profile object. On a page customization you can add changes to the page layout, and actions; but you cannot add variables, procedures, or triggers.

> **NOTE**
>
> Page customizations only apply to the RoleCenter they are specified for. In order to see them, in Dynamics 365 Business Central under **My Settings**, **Role Center** change to the specific RoleCenter that a page customization is defined for.

> **NOTE**
>
> Extension objects can have a name with a maximum length of 30 characters.

> **NOTE**
>
> `showMyCode` does not apply to profiles. Profiles defined in an extension with `showMyCode` set to `false` can still be copied using Designer.

Snippet support

Typing the shortcut `tprofile` will create the basic layout for a profile object when using the AL Language extension in Visual Studio Code.

> **TIP**
>
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Profile example

The following profile object example creates a profile for the `MyRoleCenter` Role Center, which is available in the **Role Explorer** in the UI and available to end-users. The profile also depends on the customization `MyCustomization` and modifies the layout of the **Customer List** to make the `Name` field invisible using the `modify` method. For more information, see Profile Properties.

```
profile MyProfile
{
    Description = 'Some internal comment that only the Dev can see';
    Caption = 'My User-friendly Name';
    ProfileDescription = 'A detailed description of who is this profile for, why/how to use it (etc)';
    RoleCenter = MyRoleCenter;
    Enabled = true;
    Promoted = true;
    Customizations = MyCustomization;
}

pagecustomization MyCustomization customizes "Customer List"
{
    layout
    {
        modify(Name)
        {
            Visible = false;
        }
    }
}
```

See Also

AL Development Environment
Developing Extensions
Pages Overview
Page Customization Object

Page Customization Object

2/17/2021 • 2 minutes to read • Edit Online

The page customization object in Dynamics 365 Business Central allows you to add changes to the layout and actions on page that are accessible for a profile. See Using keywords to place actions and controls for how to place actions and controls on a page customization object.

The page customization object has more restrictions than the page extension object; when you define a new page customization object, you cannot add variables, procedures, or triggers.

> **NOTE**
> A single page customization can be used with multiple profiles within the same extension. Page customizations only apply to the RoleCenters they are specified for. In order to view or changes the RoleCenters in the client, go to **My Settings** > **Role Center**.

> **NOTE**
> Extension objects can have a name with a maximum length of 30 characters.

> **NOTE**
> Modifying actions in Cue groups on page extensions is not supported.

> **NOTE**
> `showMyCode` does not apply to page customizations. Page customizations defined in an extension with `showMyCode` set to `false` can still be copied using Designer.

Snippet support

Typing the shortcut `tpagecust` will create the basic layout for a page customization object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Views

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. For more information, see Views.

Page customization example

The following page customization example `MyCustomization` makes changes to **Customer List**. By using the `moveafter` method, `Blanket Orders` is moved after the `Orders` action item. And the `modify` method is used to hide the `NewSalesBlanketOrder` action item.

```
profile TheBoss
{
    Description = 'The Boss';
    RoleCenter = "Business Manager Role Center";
    Customizations = MyCustomization;
    Caption = 'Boss';
}

pagecustomization MyCustomization customizes "Customer List"
{
    actions
    {
        moveafter(Orders; "Blanket Orders")

        modify(NewSalesBlanketOrder)
        {
            Visible = false;
        }

    }
}
```

You can use the same page customization on another profile within the same extension package by referencing its name from the profile definition, for example:

```
profile TheSalesman
{
    ProfileDescription = 'The Boss';
    RoleCenter = "Sales Manager Role Center";
    Customizations = MyCustomization;
    Caption = 'Salesman';
}
```

See Also

Developing Extensions
AL Development Environment
Page Object
Page Extension Object
Views
Page, Page Fields, and Page Extension Properties

Using the Client to Create Profiles and Page Customizations

2/17/2021 • 5 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

Besides creating profiles and page customizations by writing AL code, you can use the client. The client is a useful alternative, because you work with the user interface just as the users would. This method is especially advantageous for consultants, application administrators, and less technical users.

Overview

For detailed instructions on how to use the client to create and modify profiles, see the following articles in the Business Central application help:

- Create Profiles
- Customize Pages for Profiles.

A consequence of using the client is that profile changes apply only to the tenant. Extension-based profiles and customizations, by contrast, are available for all tenants. However, the client lets you export user-created profiles and page customizations from a tenant, then import them on another tenant.

Profiles created in the client are marked as **(user-created)** profiles. The export and import functionality lets you:

- Backup profile and page customizations locally.
- Make further changes to profiles and page customizations away from the production environment.
- Replicate profiles and page customizations on other environments and tenants.
- Preview changes in sandbox environments before going into production.

> **IMPORTANT**
> You can't use the import functionality to migrate legacy profiles from Dynamics NAV or early versions of Business Central.

Exporting profiles

When you export profiles, the system exports all user-defined profiles and page customizations that have been made in the tenant. It doesn't export profiles and page customizations introduced by extensions.

To export profiles from the client, open the **Profiles (Roles)** page, and select the **Export Profiles** action. A zip file is downloaded to your computer. The file is referred to as a *profile package*.

Once you have the profile package, you can import it as-is to another tenant. Or, you extract the files and make more changes to profiles and page customizations. When you're done, you compress the files into a profile package again and import it on tenants.

> **NOTE**
>
> To export profiles, a user requires read permission to the **Profiles** page and table as a minimum. This capability allows normal users to hand over profiles to others for troubleshooting.

Importing profiles

Importing profiles lets you add new profiles and page customizations on a tenant or replace existing ones. To import profiles, do the following steps:

1. Get the profile package that contains the new or modified profiles.
2. Before you import a package, we recommend that you export the current profiles so you have a copy.
3. Open the **Profiles (Roles)** page, select the **Import Profiles** action, and follow the instructions to import the profile package.

   You don't have to import all profiles contained in the package. You can select specific profiles.

When you replace a profile, signed-in users of the profile may be interrupted briefly.

> **IMPORTANT**
>
> To import profiles, you must be assigned the **SUPER** or **D365 Profile Mgmt** permission set.

Working with the profile package and files

A profile package is a zip file that contains profile and page customizations in separate files. These files basically contain AL source code. The following table provides an overview of the file types in a profile package.

FILE TYPE	DESCRIPTION
app.json	A required configuration file that includes dependencies to the base application and system application. In most cases, you shouldn't modify this file after exporting it. The app.json must be included in the profile package that you import.
profile.json	A required configuration file that specifies information about the profiles in the package. In most cases, you shouldn't modify this file after exporting it. The profile.json must be included in the profile package that you import.
profile file	An AL file for a user-created profile. There's a separate file for each profile. A profile file has the format `Profile._<profile ID>_.al`.
profile extension file	An AL file for customizations made to an extension-based profile. These file types have the format `ProfileExtension._<profile ID>_.al`.
page customization file	An AL file that specifies all customizations made to a page for a specific profile. A page customization file has the format `PageCustomization._[page name]_.Configuration\ <number\>.al` .

The sections that follow explain a bit more about the file types and creating the profile package.

Profile files

Each user-created profile is exported to a separate AL file. This file contains the `profile` object that defines the profile's ID, name, and Role Center. It also includes references to the page customizations it uses. For example, let's say you created a profile with the ID **MyProfile** that uses the role center page **9022 Business Manager Role Center**. You then customized the Business Manager Role Center itself and the **Customer** list page. The exported profile package would contain a file called **PROFILE.MyProfile.al**. This file would include the following code:

```
profile MyProfile
{
  CaptionML = ENU='My Profile';
  Enabled = true;
  ProfileDescriptionML = ENU='This is my sample profiles';
  Promoted = true;
  RoleCenter = 9022;
  Customizations = Configuration1; Configuration2;
}
```

The `Customizations` property identifies the page customization objects used by the profile.

Profile extension files

Customizations made to extension-based profiles are exported to a profile extension file. This file includes a `profileextension` object that specifies two types of information, depending on the changes:

- Properties that specify metadata like, the CaptionML, ProfileDescriptionML, and RoleCenter.
- References to configuration files that define page customizations.

For example, let's say you changed the description and customized the **Customer** page for the **Business Manager** profile that is provided by the Base Application extension. The profile package would then contain the file **ProfileExtension._BUSINESS MANAGER.al**. This file will contain code similar to the following code:

```
profileextension BUSINESSMANAGER_1 extends "BUSINESS MANAGER"
{
  CaptionML = ENU='Business Manager';
  Enabled = true;
  ProfileDescriptionML = ENU='My changed functionality for managers in charge of keeping the business viable
by determining product and company direction.';
  Promoted = false;
  RoleCenter = 9022;
  Customizations = Configuration3;
}
```

`Configuration3` is a reference to a page customization file for the **Customer** page. For more information, see the next section.

Page customization files

Page customizations made to user-created profiles and extension-based profiles are exported to AL files that include a `pagecustomization` object. This object defines each modification to the page. Referring to the examples above, the zip file would include three files:

- PageCustomization.*Business Manager Role Center*.Configuration1.al
- PageCustomization.*Customer List*.Configuration2.al
- PageCustomization.*Customer List*.Configuration3.al

The files would include code similar to the following code:

PageCustomization.*Business Manager Role Center*Configuration1.al

```
pagecustomization Configuration1 customizes "Business Manager Role Center"
{
  layout
  {
    modify(Control16)
    {
    Visible = false;
    }
  }
  actions
  {
  }
}
```

PageCustomization.*Customer List*Configuration2.al

```
pagecustomization Configuration2 customizes "Customer List"
{
  layout
  {
    modify("Balance (LCY)")
    {
    Visible = false;
    }
  }
  actions
  {
  }
}
```

PageCustomization.*Customer List*Configuration3.al

```
pagecustomization Configuration3 customizes "Customer List"
{
  layout
  {
    modify("Balance Due (LCY)")
    {
    Visible = false;
    }
  }
  actions
  {
  }
}
```

Creating a profile package for import

After you make modifications to exported profiles and page customizations, you'll have to create the profile package before you can import the changes.

The profile package doesn't have to contain the same files that were originally exported. But it must contain the app.json and profile.json files. Otherwise, you can't import the package. To create the profile package, just compress the files into a zip folder.

See Also

Developing Extensions
AL Development Environment
Page Object

Web Client URL

2/17/2021 • 10 minutes to read • Edit Online

There are several parameters that you can add to the Web client URL to manipulate what is displayed in the client. For example, you can open a specific company, target a specific page, report, or table. The following URL displays page **9305 Sales Order List** for the CRONUS International Ltd. company:

```
https://businesscentral.dynamics.com/?company=CRONUS%20International%20Ltd.&page=9305
```

The following URL opens report **5 Receivables – Payables** for the same company:

```
https://businesscentral.dynamics.com/?company=CRONUS%20International%20Ltd.&report=5
```

This article describes how you can construct URLs. A well-constructed URL can be useful for including in other sources, such as emails or Word documents, or sending as hyperlinks to other users.

> **IMPORTANT**
> Certain data in the URL, such as filters, could be considered sensitive information. Use discretion if you distribute URLs that contain filters, or if it's possible, exclude this information from the address.

URL Syntax

The Web client URL has the following syntax:

```
https://<hostname>[/<aadtenantid>][/<environmentname>]/?[company=<companyname>]&[page|query|report|table=
<ID>]&[tenant=<tenantID>]&[mode=<View|Edit|Create>]&[profile=<profileID>]&[customize]&[bookmark=<bookmark>]&
[captionhelpdisabled=<0|1>]&[showribbon=<0|1>]&[shownavigation=<0|1>]&[showuiparts=<0|1>]&[showheader=
<0|1>]&[isembedded=1]&[pagesize=<number of lines>]&[redirect<0|1>]&[extension=<extensionID>]
```

The URL consists of two parts; the hostname part and the query string. The hostname part includes the protocol (https) and the hostname. The query string part includes everything after `<hostname>`. The query string determines what content to target.

Syntax Key

The following table describes the notation that is used to indicate the syntax.

NOTATION	DESCRIPTION
Text without brackets	Parameters that you must type as shown.
`<>`	A placeholder for values that you must supply. Don't include the brackets in the address.
`[]`	Optional parameters. Don't include the brackets in the address.

NOTATION	DESCRIPTION
`|`	A set of values from which to choose. Use one of the options and don't include `|` in the address.

Building the URL

Use the following guidelines to write URL syntax and create a URL:

- You can place parameters in any order after `/?`. For example, the following URLs will yield the same results.

```
https://businesscentral.dynamics.com/?company=CRONUS%20International%20Ltd.&page=9305&mode=View
```

```
https://businesscentral.dynamics.com/?page=9305&mode=View&company=CRONUS%20International%20Ltd.
```

- Separate parameters after `/?` with the ampersand symbol (`&`).
- Use `%20` for any spaces in values, or similar escape sequences for other characters that can't be used in URLs.
- Enclose values in single quotation marks (`''`) if they're unescaped.

URL Parameters

The following table describes the parameters of the URL for displaying a page.

PARAMETER	DESCRIPTION
`https`	Specifies the Internet protocol to use. Only `https` is supported.
`hostname`	Specifies the hostname for Dynamics 365, for example, `businesscentral.dynamics.com`.
`<aadtenantid>`	Specifies the unique identifier for an Azure Active Directory (AAD) tenant. The value can be formatted as a GUID or domain name. It's useful to people who work across multiple AAD organizations, such as delegated administrators, support personnel, or external accountants. It allows explicitly targeting an AAD tenant. If this parameter is omitted, you'll be directed to the primary AAD tenant or the same AAD tenant that you're currently signed in to.
`<environmentname>`	(online only) Specifies the display name of sandbox or production environment to target.
`company`	The name of the company in Dynamics 365 that you want to target. If you only have one company, then you can omit this parameter.
`page`	Opens a page object.

PARAMETER	DESCRIPTION
`query`	Opens a query object. **Note:** The TopNumberOfRows property is ignored when the query opens in the browser.
`report`	Opens a report object.
`table`	Opens a table object. Opening a table requires special permissions. For more information about opening a table, see Viewing Table Data.
`ID`	The ID of the page, report, or table to open.
`tenant`	(on-premises only) Specifies the ID of the tenant to connect to. Use this parameter when Web client is deployed in multitenant architecture. The tenant that you specify must be mounted on the Dynamics 365 Business Central service instance that the Web client connects to. For more information, see Multitenant Deployment Architecture.
`mode`	Specifies the mode in which to display the page when it opens. Once the page opens, you can switch modes as usual, unless restricted by the page itself. - `View` The page can only be viewed. The user can't change data on the page. **Note:** Worksheet page types only display in the edit mode, even if the value is set to `View`. - `Edit` The user can change data on the page. **Note:** To use the edit mode, the Editable Property of the page in Page Designer must be set to **Yes**. This mode isn't supported for pages of the type List, RoleCenter, and CardPart. If you set the value to `Edit`, pages of these types still display in the view mode. For List type pages, the user can modify the list by choosing **Edit List** on the page. - `Create` Opens a blank page that enables the user to create a new item. **Note:** The `Create` mode isn't supported for pages of the type CardPart, List, ListPart, RoleCenter, and Worksheet. For pages of the type CardPart, List, and ListPart, the page displays in the view mode. Don't use this mode for Worksheet pages; otherwise you'll get an error when you try to open the page.

PARAMETER	DESCRIPTION
`profile`	Specifies the ID of the profile to open, such as `accountant` or `order processor`. It's possible for two or more profiles have the same ID. Profiles can have a scope of either system or tenant. Also, tenant profiles can be either user-defined (added by using the **Profiles** page in the client) or extension-based (added by an extension). Among these different types, the IDs of some profiles might be the same. If there's more than one profile with the same ID as you provide, the profile is selected as follows: 1. If there's a matching system profile, it's used. 2. If there's a matching user-defined tenant profile, it's used. 3. If there's only one matching extension-based profile, it's used. 4. If there are two or more extensions-based profiles with the same ID, then the error message `More than one profile has the ID '<ID>' within the Tenant scope.` appears. In this case, you can't use the `profile` parameter for this profile.
`customize`	Opens the profile for customization, enabling you to change the layout of pages as seen by users of the profile. If you omit the `profile`, then the default profile opens. For more information, see Customizing the Workspace for Profiles (Roles) in the Business Central Application Help.
`bookmark`	Specifies a record in the underlying table of the page. The value of a bookmark is an alphanumeric string of characters, for example, `27%3bEgAAAAJ7CDAAMQA5ADAANQA4ADkAMw%3d%3`. For the page types Card, CardPart, and Document, the bookmark specifies the record that is shown in the page. For page types List, ListPart, and Worksheet, the bookmark specifies the record that is selected in the list on the page. **Important:** Bookmarks are generated automatically. You can only determine a value for the bookmark by displaying the page in the Web client and looking at its address. So, a bookmark is only relevant when the address you're working with was copied from another page instance.
`captionhelpdisabled`	Specifies that the ability to look up Help by selecting a field caption is disabled. If you want the Help look up from the field captions, either omit this parameter or set its value to `0`, such as `captionhelpdisabled=0`. If you don't want the Help lookup from field captions, set the value to `1`, such as `captionhelpdisabled=1`. **Note:** For this parameter to take effect, add it at the first request when the user signs in. Adding the parameter on an existing session has no effect.

PARAMETER	DESCRIPTION
`showribbon`	Specifies whether to show the action bar on pages when they open. To show the action bar, omit this parameter or use `showribbon=1`. To hide the action bar, use `showribbon=0`. When you hide the action bar, it will remain hidden as you move to different pages, until you refresh or reload the browser or use `showribbon=1` in the URL. Once you move to another page, the parameter is no longer shown in the URL, even though it's still in effect.
`shownavigation`	Specifies whether to show the navigation bar when the specified page opens. To show the navigation bar, omit this parameter or use `shownavigation=1`. To hide the navigation bar, use `shownavigation=0`. When you hide the navigation bar, it will remain hidden as you move to different pages, until you refresh or reload the browser or set `shownavigation=1` in the URL. Once you move to another page, the parameter is no longer shown in the URL, even though it's still in effect.
`showuiparts`	Specifies whether to show UI parts when the specified page opens. To show parts, omit this parameter or use `showuiparts=1`. To hide parts, use `showuiparts=0`. This parameter only affects parts that are shown in FactBoxes. Once you move to another page, the parameter is cleared.
`showheader`	Specifies whether to show the **Dynamics 365 Business Central** header and its functionality. To show the header, omit this parameter or use `showheader=1`. To hide the header, use `showheader=0`. The header is the bar at the top of pages. It gives access to the general functionality like **Tell Me**, **Notifications**, **My Settings**, and more. When the header is hidden, functionality that has a keyboard shortcut, like **Tell Me**, is still accessible by the shortcut. The parameter is preserved when you move to other pages and if you refresh or reload the browser. **Note:** Introduced in Business Central 2020 release wave 1, update 16.2.
`isembedded`	Specifies to open the Business Central Web client in the embedded mode. To set the embedded mode, use `isembedded=1`. This parameter is intended for use when Business Central Web client is embedded in another web application, like SharePoint. The embedded mode hides the **Dynamics 365 Business Central** header and adjusts how navigation works to suit the web application. For more information, see Embedding Business Central Web Client in Other Websites.

PARAMETER	DESCRIPTION
`pagesize`	Specifies the number of lines to display in a list. For example, `pagesize=10` specifies that 10 lines will be displayed. The default value, if the parameter isn't specified, is `20` The parameter applies only to pages that contain a `repeater` control. It's intended for use when Business Central Web client is embedded in another web application, like SharePoint. For more information, see Embedding Business Central Web Client in Other Websites.
`redirect`	Specifies whether users are presented with an option to download the Business Central Mobile App when they open the Web client in a mobile browser to improve the user experience. If you don't want to give users this option, set the value to `0`, such as `redirect=0`.
`extension`	Specifies the unique identifier (ID) of an extension that is deployed on the environment. This parameter is typically used during the development of the specified extension in a non-production environment. When this parameter is set, only the specified extension is available in the client, and the client opens in Designer. All other extensions are ignored and not visible. This parameter enables you to isolate and focus on the behavior of the specified extension only. An extension ID is a 32-digit GUID, such as `72CC5E27-BD97-4271-AF55-F77E4471E493`. You set this parameter using the format `extension={GUID}`, for example: `&extension={72CC5E27-BD97-4271-AF55-F77E4471E493}` You can determine an extension ID by opening the extension in Visual Studio Code and looking in the app.json file. Or, you can run the Get-NAVAppManifest cmdlet on the extension package.

Filtering Data on the Page

You can filter the data that is displayed in the page by using the filter parameter. The filter parameter let you display specific records from the underlying table of the page.

Example

The following address displays data in page 9305 only for the customer who has the **Sell-to Customer No.**=10000 and the **Location Code**=Blue.

```
https://businesscentral.dynamics.com/?company=CRONUS%20International%20Ltd.&page=9305&filter='Sell-to
Customer No.' IS '10000' AND 'Location Code' IS 'BLUE'
```

Filter Syntax

The filter has the following syntax.

```
&filter='<field>' IS '<value>'[ AND '<field>' IS '<value>']
```

Include a space or `%20` before and after the `IS` and `AND` operators. You can add the filter anywhere in the address after `/?`.

Filter Parameters

The following table describes the filter parameters.

PARAMETER	DESCRIPTION
`field`	The name of the table field on which to filter.
`IS`	Specifies the equal operator.
`value`	The value of the table field on which to filter.
`AND`	Use this parameter to specify more than one filter. It specifies an "and" operator for adding additional filters. Place `AND` between each additional filter. To be included in the page data, the table record must match values for all fields in the filter.

See Also

Viewing Table Data

Linking to the Dynamics 365 Business Central App

2/17/2021 • 4 minutes to read • Edit Online

The protocol handler for the Business Central Mobile App lets you construct a URL for starting the app on a device, such as a phone or tablet. You can then distribute this URL by e-mail or from a Web page to the users.

The Business Central Mobile App URL is based on the *ms-businesscentral* URI scheme, which is registered automatically when the app is installed. Invoking a URL based on this scheme will start the app with the provided parameters.

Constructing the URL

To construct a URL, start with *ms-businesscentral* scheme, and then add additional parameters as needed. Some parameters are required and others are optional.

The structure of a Business Central Mobile App link is very similar to links for the Web client, and has the following syntax:

```
ms-businesscentral://[<hostname>][/<aadtenantid>][/sandbox]/[?<parameter>=<value>[&<parameter>=<value>]]
```

`[]` indicates an optional parameter; all other parameters are required.

`<>` indicate values that you must supply. Do not include the brackets in the address.

Parameters

The following table describes the parameters for the main part of the URL, which are the parameters up to and including `[/sandbox]/`.

PARAMETER	DESCRIPTION	EXAMPLE

hostname	Domain name for the Dynamics 365 Business Central solution or IP address of the computer/server that hosts it. This is required for an ISV Embed solution. For standard Business Central, you use `businesscentral.dynamics.com` or you can omit this parameter.	`ms-businesscentral://businesscentral.dynamics.com/`
		`ms-businesscentral:///`
		`ms-businesscentral://businesscentral.mysolution.com/`

aadtenantid	The unique identifier for an Azure Active Directory (AAD) tenant. The value can be formatted as a GUID or domain name. This is useful to those who work across multiple AAD organizations, such as delegated administrators, support personnel or external accountants, because it allows explicitly targeting an AAD tenant. If this is omitted, you will be directed to the primary AAD tenant or the same AAD tenant that you are currently signed in to.	`ms-businesscentral://businesscentral.mysolution.com/mysolutionaadtena`
sandbox	Specifies that the URL should target the Dynamics 365 Business Central sandbox environment instead of a production environment.	`ms-businesscentral:/businesscentral.dynamics.com/sandbox/` `ms-businesscentral://businesscentral.mysolution.com/sandbox/`

The following table describes the optional parameters that are indicated by `[?<parameter>=<value>[&<parameter>=<value>]]` in the syntax. These parameters are referred to as the *query parameters*.

PARAMETER	DESCRIPTION	EXAMPLE
page	The ID of the page that you want to open directly.	`ms-businesscentral:///?page=21` `ms-businesscentral://businesscentral.mysolution.com/?page=21`

PARAMETER	DESCRIPTION	EXAMPLE

bookmark	The bookmark of the record you want to open. The value of a bookmark is an alphanumeric string of characters, for example, `19%3bGwAAAAJ7BDEAMAAwADA%3d`.
	
	For the page types Card, CardPart, and Document, the bookmark specifies the record that is shown in the page. For page types List, ListPart, and Worksheet, the bookmark specifies the record that is selected in the list on the page.
	
	Important: Bookmarks are generated automatically. You can only determine a value for the bookmark by displaying the page in the client and looking at its address. Therefore, a bookmark is only relevant when the address you are working with has been copied from another instance of the page.
		`ms-businesscentral:///?bookmark=19%3bGwAAAAJ7BDEAMAAwADA%3d`
		
		`ms-businesscentral://businesscentral.mysolution.com/?bookmark=19%3bGwAAAAJ7BDEAMAAwADA%3d`

filter	The filter you want to apply to the page.
	
	The filter parameter enables you to display only records from the underlying table of the page that have specific values for one or more fields. For more information about filters, see Filtering Data on the Page.
		`ms-businesscentral:///?page9305&filter='No.'%20IS%20'1001'`
		
		`ms-businesscentral:///?page9305&filter='Sell-to-Customer-No.'%20IS%20'10000'%20AND%20'Location-Code'%20IS%20'BLUE'`
		
		`ms-businesscentral://businesscentral.mysolution.com/?page9305&filter='No.'%20IS%20'1001'`
		
		`ms-businesscentral://businesscentral.mysolution.com/?page9305&filter='Sell-to-Customer-No.'%20IS%20'10000'%20AND%20'Location-Code'%20IS%20'BLUE'`

profile	Specifies the ID of the profile to open, such as `accountant` or `order processor`.
	
	Be aware that it is possible for two or more profiles have the same ID. Profiles can have a scope of either system or tenant. In addition, tenant profiles can be either user-defined (added by using the **Profiles** page in the client) or extension-based (added by an extension). Among these different types, the IDs of some profiles might be the same. When there is more than one profile with the same ID as the one you provide, the process for launching the profile is as follows:
	1. If there is a matching system profile, it is used.
	2. If there is a matching user-defined tenant profile, it is used.
	3. If there is only one matching extension-based profile, it is used.
	4. If there are two or more extensions-based profiles with the same ID, then the error message `More than one profile has the ID '<ID>' within the Tenant scope.` appears. In this case, you cannot use the `profile` parameter for this profile.
		`ms-businesscentral:///?profile=BUSINESS%20%MANAGER`
		
		`ms-businesscentral://businesscentral.mysolution.com/?profile=BUSINESS%20%MANAGER`

PARAMETER	DESCRIPTION	EXAMPLE
`customize`	Opens the profile for customization, enabling you to change the layout of pages as seen by users of the profile. If you omit the `profile`, then the default profile opens. For more information, see Customizing the Workspace for Profiles (Roles) in the Business Central Application Help.	
company	The company that you want to open in the client. If not provided, the default company is used. CRONUS%20International%20Ltd.	`ms-businesscentral:///?'company=CRONUS%20International%20Ltd.'` `ms-businesscentral://businesscentral.mysolution.com/?'company=CRONUS%`
mode	Whether the page opens in view, edit, or create mode. `view` only lets you see the data on the page, not modify data. `edit` lets you to modify data on the page. `create` lets you to modify data on the page and add new entities.	`ms-businesscentral:///?page=21&mode=create` `ms-businesscentral://businesscentral.mysolution.com/?page=21&mode=create`

The query parameters can be in any order. However, the first parameter must be preceded by the `?` symbol, and any additional parameters must be preceded by the `&` symbol.

See Also

Web Client URL
Introducing the Business Central Mobile App

Working with Translation Files

2/17/2021 • 4 minutes to read • Edit Online

Dynamics 365 Business Central is multilanguage enabled, which means that you can display the user interface (UI) in different languages. In Dynamics 365 Business Central this is done using XLIFF files, which is a standardized format used for computer-based translations.

Generating the XLIFF file

To add a new language to the extension that you have built, you must first enable the generation of XLIFF files. The XLIFF file extension is .xlf. The generated XLIFF file contains the strings that are specified in properties such as **Caption**, **CaptionML**, and **Tooltip**.

> **NOTE**
> To submit an app to AppSource, you must use XLIFF translation files.

In the app.json file of your extension, add the following line:

```
"features": [ "TranslationFile" ]
```

Now, when you run the build command (**Ctrl+Shift+B**) in Visual Studio Code, a `\Translations` folder will be generated and populated with the .xlf file that contains all the labels, label properties, and report labels that you are using in the extension. The generated .xlf file can now be translated.

> **IMPORTANT**
> Make sure to rename the translated file to avoid that the file is overwritten next time the extension is built.

By setting the `GenerateCaptions` flag in the app.json file, you specify that you want to generate captions based on the object name for pages, tables, reports, XMLports, request pages, and table fields. If the object already has a `Caption` or `CaptionML` property set, that value will be used, for table fields the `OptionCaption` is used. The syntax is the following:

```
"features": [ "TranslationFile", "GenerateCaptions" ]
```

GenerateLockedTranslations

APPLIES TO: Business Central 2020 release wave 2 and later

By setting the `GenerateLockedTranslations` flag in the app.json file, you specify that you want to generate `<trans-unit>` elements for locked labels in the XLIFF file. The default behavior is that these elements are not generated. For more information, see JSON Files.

```
"features": [ "GenerateLockedTranslations" ]
```

Label syntax

The label syntax is shown in the example below for the **Caption** property:

```
Caption = 'Developer translation for %1',  Comment = '%1 is extension name', locked = false, MaxLength=999;
```

> **NOTE**
>
> The `comment`, `locked`, and `maxLength` attributes are optional and the order is not enforced. For more information, see Label Data Type.

Use the same syntax for report labels:

```
labels
{
  LabelName='LabelText',Comment='Foo',MaxLength=999,Locked=true;
}
```

And the following is the syntax for **Label** data types:

```
var
a:Label'LabelText',Comment='Foo',MaxLength=999,Locked=true;
```

> **IMPORTANT**
>
> The **ML** versions of properties are **not** included in the .xlf file:
>
> - CaptionML
> - ConstValueML
> - InstructionalTextML
> - OptionCaptionML
> - PromotedActionCategoriesML
> - RequestFilterHeadingML
> - ToolTipML
>
> The TextConst Data Type is not included in the .xlf file either.

The XLIFF file

In the generated .xlf file, you can see a `<source>` element for each label. For the translation, you will now have to add the `target-language` and a `<target>` element per label. The `<trans-unit id>` attribute corresponds to the object ID in the extension. This is illustrated in the example below.

```
<?xml version="1.0" encoding="utf-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xliff:document:1.2 xliff-core-1.2-transitional.xsd">
  <file datatype="xml" source-language="en-US" target-language="da-DK" original="ALProject16">
    <body>
      <group id="body">
        <trans-unit id="PageExtension 50110" maxWidth="999" size-unit="char" translate="yes"
xml:space="preserve">
          <source>Developer translation for %1</source>
          <target>Udvikleroversættelse for %1</target>
          <note from="Developer" annotates="general" priority="2">%1 is extension name</note>
          <note from="Xliff Generator" annotates="general" priority="3">PageExtension - PageExtension</note>
        </trans-unit>
      </group>
    </body>
  </file>
</xliff>
```

NOTE

You can have only one .xlf file per language. If you translate your extension to multiple languages, you must have a translation file per language. There is no enforced naming on the file, but a suggested good practice is to name it `<extensionname>.<language>.xlf`.

When the extension is built and published, you change the language of Dynamics 365 Business Central to view the UI in the translated language.

Translating other extensions

To translate other extensions, for example, adding translations to the Base Application, you must reference the project to be translated using the `dependencies` section in the app.json file. For more information, see JSON Files. When you have the dependencies added, you can add xliff files in your current project that translates the object captions of the referenced extension. Create a directory named **Translations** in the root of the extension, and place the translated xliff file there. When your extension is then built and published, change the language of Dynamics 365 Business Central to view the UI in the translated language.

Translation and Localization apps

NOTE

The following section only applies to versions released before Business Central 2019 release wave 2.

The .xlf files approach cannot be used for translating the base application. If you are working on a translation or localization app (for example for a country/region localization), you must take the .txt file containing the base application translation, and place the file in the root folder of your extension. When the extension is compiled, the .txt file is then packaged with the extension.

We recommend that you use only one .txt file per language. There is no enforced naming on the .txt files, but a suggested good practice is to name it `<extensionname>.<language>.txt`.

For more information about importing and exporting .txt files, see How to: Add Translated Strings By Importing and Exporting Multilanguage Files in Dynamics NAV.

See Also

How to: Add Translated Strings By Importing and Exporting Multilanguage Files in Dynamics NAV
Working with labels
Working with multiple AL project folders within one workspace
JSON Files

Introducing the Dynamics 365 Business Central Mobile App

2/17/2021 • 3 minutes to read • Edit Online

The app displaying the Business Central tablet client and Business Central phone client is targeted at users in small and medium sized businesses that want to access data from a tablet or a phone. The main advantages of this offering are portability and flexibility, which allows end users to perform tasks when they are away from their desk. Having a Dynamics 365 Business Central solution that runs on a smaller device also brings it in the hands of many more users and your app is easy to distribute.

> **IMPORTANT**
>
> The Business Central tablet client and Business Central phone client do not replace the Business Central Web client. Instead, they offer a touch interface for a limited set of application scenarios compared to the Business Central Web client.

The Business Central Web client supports more complex business processes and heavier data entry than it is possible on the Business Central tablet client and Business Central phone client. Business Central is also designed for intensive use, and the user can have multiple windows open at the same time, but this is neither possible in Business Central phone client or Business Central tablet client. For more information, see Differences and Limitations When Developing Pages for the Business Central Mobile App.

The design for the Business Central tablet client is optimized for the touch experience and reduced use of the on-screen keyboard. On the other hand, the design for the Business Central phone client is about touch optimization, given its smaller screen size. The Business Central phone client layout is designed to support one-hand and both hands use, which allows the important data and buttons to be available within thumbs reach.

> **NOTE**
>
> In this documentation, you will see mentions of Business Central tablet client, Business Central phone client, and *Dynamics 365 Business Central Mobile App*. Business Central tablet client and Business Central phone client describe the interface tailored to the category of mobile device, which is one of the tools available for developers for designing mobile solutions, whereas *Dynamics 365 Business Central Mobile App* is the common name for the app across all devices; the end result made with these tools.

Considering the user scenarios

When you design your solution for the Business Central tablet client and the Business Central phone client, you must make sure that scenarios are simple enough to be meaningful and usable. The tablet and phone designs are meant for lighter tasks and are useful, for example, for traveling salespeople or service technicians who need a portable, online, easy-to-use app that provides an overview, for example, of daily tasks and items in stock.

Depending on the scenarios that your tablet and phone solution will support, it will either make sense to create a new Role Center for tablet and phone only, or share the same Role Center across all of the client types. In some cases, it can make sense to have a user sign in with two different profiles, one for a desktop client and one for mobile devices. In other cases, duplicating pages and designing specific duplicates to be device-oriented is the best solution.

If you have existing page objects that you want to make available on Business Central tablet client or Business Central phone client, we strongly recommend that you plan time to evaluate carefully which actions, sections,

and fields will be needed for the user scenarios you want to enable. Fields and actions that are not needed should not be visible to users of your app. The UI must be simplified significantly to work well on a small device. For more information, see Designing for Different Screen Sizes on Tablet and Phone.

Supported credential types

Business Central tablet client and Business Central phone client support the same credential types as Business Central Web client. For more information, see Authentication and Credential Types.

See Also

Getting Started Developing for the Dynamics 365 Business Central Mobile App
Differences and Limitations When Developing Pages for the Dynamics 365 Business Central Mobile App

Deciding on Your Tablet and Phone Strategy

2/17/2021 • 3 minutes to read • Edit Online

To offer users a great mobile experience, you must decide on a strategy for how to accomplish this based on an analysis of your users' needs. This topic explains the different options for developing for the Business Central tablet client and Business Central phone client, but the documentation you will find in this section is primarily focused on the first of these scenarios.

Mobile app development strategy

This section briefly describes some of the options that exist for Dynamics 365 Business Central.

DEVELOPMENT STRATEGY	WHAT ALSO TO CONSIDER	EXAMPLES	FOR MORE INFORMATION, SEE
Business Central platform Use the AL Language development environment to modify and extend the Business Central tablet client and Business Central phone client.This scenario resembles developing for Business Central Web client. The main advantages of this strategy are: - Extending and modifying Business Central tablet and phone clients is useful for a minimal learning curve and a reduction of development costs; scenarios automatically work on multiple operating systems without having to worry about the maintenance. - You can reuse existing investments in Dynamics 365 Business Central page objects, business logic, and javascript-based client add-ins. Ideal for user scenarios which can be achieved by using simple application pages.	Dynamics 365 Business Central only supports a specific number of page types, and this can be a limitation in some type of development projects.	- For salespeople tracking customers, looking up item details, and capturing orders. - For technicians on the road using and re-ordering spare parts. - For simple approval scenarios.	Introducing the Dynamics 365 Business Central Mobile App Getting Started Developing for the Dynamics 365 Business Central Mobile App

DEVELOPMENT STRATEGY	WHAT ALSO TO CONSIDER	EXAMPLES	FOR MORE INFORMATION, SEE
Power App Use the Power Apps platform connected to Business Central either using the Business Central connector in Power Apps or custom connector and access your modify data.	The standard Business Central connector for Power Apps only supports built in APIs, so you may need to use custom connector feature to access your custom APIs.	For field salesforce in need of user experience that is more customized or tightly connected to 3rd party software or hardware.	Connecting to Your Business Central Data to Build a Business App Using Power Apps Create a canvas app from a template in Power Apps
Connected mobile app Based on OData web services or SOAP web services technologies, write an app that interacts with Dynamics 365 Business Central. Visual Studio includes project templates for this kind of app. This strategy applies when you want to build a highly customized app with your own UI design that takes advantage of all the rich features which native apps provide.	Cost to learn development tools and languages outside AL, preparing new development environments. Cost of licensing any of these tools, and having to maintain code for different operating systems.	A simple touch interface for users to scan their access card for time registration.	OData Web Services Data Modification OData Web Services SOAP Web Services Web Services

Remarks

If you are developing using the AL language, use a browser for continuous development and test of the Business Central tablet client and the Business Central phone client solution that you are working on. Switching to running in a browser is an easy and efficient way to test what new and modified pages look like. Running the Business Central tablet client and Business Central phone client in a browser is only recommended for development scenarios. For more information, see Opening the Dynamics 365 Business Central Tablet or Phone Client from a Browser.

See Also

Getting Started Developing for the Dynamics 365 Business Central Mobile App
Introducing the Business Central Mobile App

Getting Started Developing for the Dynamics 365 Business Central Mobile App

2/17/2021 • 2 minutes to read • Edit Online

The Business Central tablet client and Business Central phone client are built on the same framework as the Business Central Web client, such that they are all based on the same Dynamics 365 Business Central pages. Developing for Business Central tablet client and Business Central phone client is not much different from developing pages for Business Central Web client either, since it is also done from the AL Language development environment.

Steps for Developing for the Business Central tablet client and Business Central phone client

The first step to take when you are developing for the Business Central tablet client and Business Central phone client is to consider the design and implementation of the solution:

- What is the business scenario that you want to support?
- Do you want to extend your existing data model?
- Should the solution work well on both a desktop computer, on the tablet and also on a phone?
- What design should the Role Center that will be the central dashboard for the solution have?

Understanding the business scenario is important to build the best solution. Design your solution for users who are most typically occasional users who need an overview of their daily work status and perform relatively simple or light data entry. Many of these preparations resemble those for developing for Business Central Web client, but in an even smaller scale.

Once the data model, users, permissions, and profiles are in place, you can start developing the Role Center. You can either reuse an existing Role Center or create a new one. This will depend on the business scenario. The Business Central tablet client and the Business Central phone client are designed to be smart about laying out the same Role Center depending on the display target.

When developing for the Business Central phone client you want to think even more about design and usability on a very small screen. On the phone the app will always by default start a page in view mode, meaning that the user actively must switch to edit mode.

There are some best practices and limitations to consider in particular caused by the smaller screen size and touch experience. For more information, see Differences and Limitations When Developing Pages for the Dynamics 365 Business Central Mobile App. On devices that run the Business Central Mobile App and have a camera and location capability you also have a couple of additional options. For more information, see Implementing the Camera in AL and Implementing Location in AL.

To complete designing your Business Central Mobile App solution, you should consider offering users Help, to guide them through pages or workflows. For more information about adding help to your solution, see Extend, Customize, and Collaborate on the Help for Dynamics 365 Business Central.

The next steps are to consider how to deploy your solution and how to distribute it to your customers. After you have completed your solution, you can send an e-mail to the users to let them know that they can download Dynamics 365 Business Central from the relevant store and include the organization URL and sign-in information. For more information, see Linking to the Dynamics 365 Business Central Mobile App.

See Also

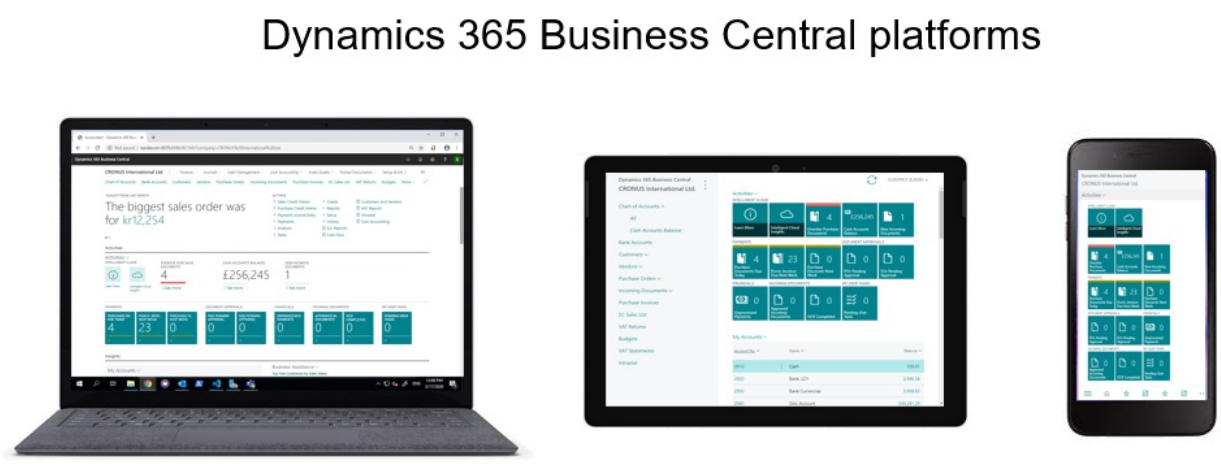
Deciding on Your Tablet and Phone Strategy
Differences and Limitations When Developing Pages for the Dynamics 365 Business Central Mobile App

Designing for Different Screen Sizes on Tablet and Phone

2/17/2021 • 2 minutes to read • Edit Online

When designing application pages for the Business Central tablet client and the Business Central phone client, it is best practice to consider the size of the tablets or phones that your end users have access to. It is an advantage if the solution works well on both small and large screen sizes, but we also recommend that you consider thoroughly the most frequently used screen sizes for your end user experience.

Designing for small screens can be more challenging, because pages will show fewer fields, columns, and tiles. Therefore, a good way to identify issues on how your application pages are displayed is to test on the smallest supported screen size.



Form factor considerations

Users can scroll both the navigation and the content area of the Business Central Mobile App on a tablet to access all data for a given page. However, it is recommended that the scrolling in the navigation bar is minimal. The navigation bar is the area on the left-hand side of Business Central Mobile App and it is designed to provide easy access to important information and tasks that the user should not lose sight of when scrolling on the content area.

On phones the Business Central Mobile App displays only one part at a time on the Role Center. On the Home page, the Activity tiles are always displayed first, and you navigate through the bottom menu to explore the content area.

Guidance for page element types on smallest tablet devices

The following table provides a list of elements that fit in the page content or the app bar without scrolling.

PAGE TYPE	DISPLAYS ON SMALLEST TABLET DEVICE
RoleCenter	4 tiles in 1 group, or 2 groups together with 2 tiles
List Pages	5 columns of type Text50 or 8 columns of type Text20

PAGE TYPE	DISPLAYS ON SMALLEST TABLET DEVICE
Card Pages	- CardPage Factbox with up to 15 fields - 2 CardPage Factboxes with up to 6 fields each - Activities Factboxes with 4 tiles in 1 group, or 2 groups together with 2 tiles
Document Pages	- CardPage Factbox with up to 15 fields - 2 CardPage Factboxes with up to 6 fields each - Activities Factboxes with 4 tiles in 1 group, or 2 groups together with 2 tiles

Testing using a browser

Using a browser you can test how your application pages will look on various device sizes. For more information, see Opening the Business Central Tablet or Phone Client from a Browser.

When running Business Central tablet client or Business Central phone client in a browser, you can use Microsoft Edge Developer Tools to emulate different screen sizes. For more information, see Microsoft Edge Developer Tools.

See Also

Deciding on Your Tablet and Phone Strategy
Differences and Limitations When Developing Pages for the Dynamics 365 Business Central Mobile App
Displaying Data as Tiles
Gesture Property

Differences and Limitations When Developing Pages for the Business Central Mobile App

2/17/2021 • 4 minutes to read • Edit Online

Developing for the Business Central tablet client and Business Central phone client is similar to developing for the Business Central Web client. However, there some natural limitations on tablets and phones, such as not having a physical keyboard and mouse, as well as a smaller screen.

Differences and limitations overview

The following table describes some of the most common differences and limitations that you might experience when developing for Business Central tablet client and Business Central phone client.

CONCEPT	ON TABLET	ON PHONE	EXAMPLE	RECOMMENDATION/ REMARKS
Activity groups	Only the **Home** activity group is shown.	Only the **Home** activity group is shown.	**Home** and **Posted Documents** on the `Sales Order Processor` Role Center.	Design pages to expose the workflows needed by the user. For example, configure the profile to show the important list pages under the **Home** activity group. Alternatively, consider designing a new Role Center if the activities for the activity group greatly vary from activities in other activity group.
Selecting multiple records in lists	Not available.	Not available.	`Ctrl+A` or `Ctrl+Click` on rows in a list using Business Central Web client.	Avoid scenarios requiring selecting multiple rows on a list. Also, try to minimize actions on lists.
Actions in the action bar	Only Promoted actions are shown.	Only Promoted actions are shown.	On the `Small Business` Role Center.	Use the development environment to promote actions. Alternatively, configure the profile and add actions to the Home tab.
FactBoxes	Not shown on List pages or Worksheet pages.	Not shown on List pages or Worksheet pages.	`Customer` list on the `Small Business` Role Center.	Make sure the same information is visible on the corresponding card page of the given record.

CONCEPT	ON TABLET	ON PHONE	EXAMPLE	RECOMMENDATION/ REMARKS
Advanced filters	No column-specific filtering is available.	No column-specific filtering is available.	On the `Customer` list page.	Send data to Excel and do the complex filtering there.
Tell Me	Not available yet.	Not available yet.	On Business Central Web client.	Design pages to expose the workflows needed by the user. For example via list places, tiles or actions.
Role Explorer	Not available yet.	Not available yet.	On Business Central Web client.	Design pages to expose the workflows needed by the user. For example via list places, tiles or actions.
Fields in FastTabs	Fields in FastTabs on list pages are not shown. Only the repeater control is shown in the content area of the page.	Not available.		Design List pages to avoid having important columns on the far right of the column list. Assume you have no control over how many columns are displayed and consider that only the first few columns will be made visible.
Select from full list	Not available on lookups. Users are not able to run actions on a lookup page, and they cannot access the full set of records.	Not available on lookups. Users are not able to run actions on a lookup page, and they cannot access the full set of records.	On the `Item Card` when selecting the **Base Units of Measure**.	Make sure the appropriate columns are visible on the lookup. The user is still able to filter, scroll, and search through the lookup.
Search across list columns	Partly supported. Search will not include FlowFields.	Partly supported. Search will not include FlowFields.	On the `Customer` list page.	
Lookups	Available.	Available, with the difference that advanced and simple lookups behave similarly on the phone. The lookup will not bring up the card, show FactBoxes, or any field groups.	See examples on the `Customer Card` page.	
Matrix controls	Not available.	Not available.	See example in `G/L Budget`.	

CONCEPT	ON TABLET	ON PHONE	EXAMPLE	RECOMMENDATION/ REMARKS
File download	Available. Cannot download multiple files at the same time.	Available. Cannot download multiple files at the same time.	`Trial Balance` report in the **Print to Excel** check box.	
Worksheet pages	Available.	Not available; an error message is displayed.	`Sales Price` Worksheet or `Cash Flow` Worksheet.	Run this type of page from the Business Central Web client, or Business Central tablet client.
Lists	Available.	Available, with the difference that these are displayed in a brick layout with a number of differences and limitations. For an overview, see Displaying Data as Tiles.	Customers or Sales Orders pages.	
Indentation in repeater controls	Available.	Not available. The repeater control will be rendered as a regular flat brick layout.	Chart of Accounts and Contacts List pages.	
Scope of actions	Available.	Available, but there are some behavioral differences regarding the Scope Property. Also, see Defining Action Scope for Business Central Pages.		
Automatic input focus on first editable field of a page	Not available.	Not available.	`Customer Card` page. In the Web client, focus will automatically be on the first editable field (such as the `Name` field), enabling you to change the value right away. In the Tablet or Phone client, this field will not be in focus; instead, you will have to manually select the field first in order to make changes.	The reason for this behavior is to prevent the in-app keyboard from initially displaying and occupying screen space.

See Also

Displaying Data as Tiles
Implementing the Camera in AL
Implementing the Location in AL
Role Center Behaviors
Defining Action Scope for Business Central Pages

Opening the Business Central Tablet or Phone Client from a Browser

2/17/2021 • 2 minutes to read • Edit Online

You can open the Business Central tablet client or the Business Central phone client by using a browser from a device that has a network connection. This can make it easier to test your solution during the design phase.

> **IMPORTANT**
>
> The steps in this article illustrates how you can open the Business Central tablet client in a browser. The syntax and options for opening Business Central phone client in a browser are the same; just replace *tablet* with *phone* in the examples later in this section.

To open Business Central tablet client in a browser

1. Open the web browser.
2. In the address box of the browser, type one of the following URLs.

TO OPEN	URL	EXAMPLE
The Role Center for the default company	*https://ComputerName:Port/WebServerInstance/tablet* Or (for multitenant deployments) *https://ComputerName:Port/WebServerInstance/tablet?tenant=TenantID*	https://MyBCWeb:8080/BC160/tablet
The Role Center for a specific company	*https://ComputerName:Port/WebServerInstance/tablet?company=CompanyName* Or *https://ComputerName:Port/WebServerInstance/tablet?tenant=TenantID&company=CompanyName*	https://MyBCWeb:8080/BC160/tablet?company=CRONUS%20International%20Ltd.
A specific page	*https://ComputerName:Port/WebServerInstance/tablet?page=ID* Or *https://ComputerName:Port/WebServerInstance/tablet?tenant=TenantID&page=ID*	https://MyBCWeb:8080/BC160/tablet?page=22

TO OPEN	URL	EXAMPLE
A specific report	*https://ComputerName:Port/WebServerInstance/tablet?report=ID* Or *https://ComputerName:Port/WebServerInstance/tablet?tenant=TenantID&report=ID*	https://MyBCWeb:8080/BC160/tablet?report=8
A specific profile	*https://ComputerName:Port/WebServerInstance/tablet?profile=ProfileID* Or *https://ComputerName:Port/WebServerInstance/tablet?tenant=TenantID&profile=ProfileID*	https://MyBCWeb:8080/BC160/tablet?profile=Small-Business

Substitute the following parameters:

- **ComputerName** with the name of the computer that is running the Business Central Web Server components.
- **Port** with the port number that you configured for the Business Central Web Server components during installation.
- **WebServerInstance** with the virtual directory alias under which the Business Central tablet client or the Business Central phone client exists on the web server. For more information, see Installing Business Central Using Setup.
- **TenantID** with the name of the tenant that you want to connect to. This parameter is only required when Business Central is deployed in a multitenant architecture. The tenant that you specify must be mounted on the Business Central Server instance that the Business Central Web client connects to. For more information, see Multitenant Deployment Architecture.
- **CompanyName** with the name of the company in Business Central. This parameter is optional and is only needed if you want to open a different company than the one specified in **My Settings**.
- **ID** with the ID that is assigned to the page or report in Business Central.
- **ProfileID** with the ID that is assigned to the profile in Business Central.

See Also

Introducing the Dynamics 365 Business Central Mobile App

Develop a Sales Rep Role Center for the Tablet Client

2/17/2021 • 3 minutes to read • Edit Online

In this example, you will learn how to create a new Role Center for the Business Central tablet client. Developing for the Business Central tablet client occurs in the AL Language development environment and is not much different from developing for one of the other Business Central clients. This example will concentrate on how to build a Role Center for a sales representative, which links to already existing page objects, but combined in a way so that it works well on the tablet.

About this example

This example illustrates the following tasks:

- Creating a Role Center page
- Adding existing pages to the Role Center
- Adding actions to pages from the Role Center
- Testing the Role Center page

Prerequisites

To complete this example, you will need:

- Business Central installed with a developer license
- Business Central Web Server components
- CRONUS International Ltd. demonstration database
- A supported browser. For more information, see System Requirements for Dynamics 365 Business Central 2020 Release Wave 1

Story

Simon is a partner developer working for CRONUS International Ltd. Nancy is a Sales Representative at Contoso Consulting. Simon has to build a new Role Center to support Nancy in her job. When at work, Nancy spends part of her time on the road with only her tablet available on customer visits. Nancy needs access to KPIs on the front page. She needs easy access to filter for the customers who she will visit. When at the customer site, she creates sales quotes. Simon wants to build a Role Center that can be used on a tablet and he wants to reuse as much code and as many page objects as possible.

The following code illustrates how Simon implements the Role Center.

```
page 50106 "Sales Rep Role Center"
{
    PageType = RoleCenter;

    layout
    {
        // Add already existing pages to help Nancy access activities, customers, and charts to the Role
Center.
        area(RoleCenter)
        {
            part("O365 Activities"; "O365 Activities")
            {
            }

            part("My Customers"; "My Customers")
            {
            }

            part("Generic Chart"; "Generic Chart")
            {
            }

            part("Trial Balance"; "Trial Balance")
            {
            }
        }
    }

    // Add actions that link to other pages, which Nancy uses in her daily work on the tablet.
    actions
    {
        area(Creation)
        {
            action("Sales Quote")
            {
                Caption = 'New';
                RunObject = Page "Sales Quote";
                Image = Quote;
                Promoted = true;
            }

            action("Customer List")
            {
                Caption = 'Customers';
                RunObject = Page "Customer List";
                Promoted = true;
                //  Sales Quote as an action item available from the action pane in the New group
                PromotedCategory = New;
            }

            action("Item List")
            {
                Caption = 'Items';
                RunObject = Page "Item List";
                Promoted = true;
                PromotedCategory = New;
            }
        }
    }
}
```

For more information about the specifics of Role Center structure and design, see Designing Role Centers.

Simon now wants to test the Sales Rep Role Center that he created, and for testing purposes he uses a browser window. He enters a URL that specifically opens the page 50006 from tablet.aspx. His URL now resembles this: *https://MyBCWeb:8080/BC160/tablet.aspx?page=50006*. For more information, see Opening the Business

Central Tablet or Phone Client from a Browser.

Next steps

Nancy now has a Role Center that gives her access to most of the information that she needs when she is on the road. The next step for Simon is to refine the Sales Rep Role Center by adding more functionality, for example, the ability to retrieve more lists or making sure that Nancy can smoothly continue to work when she is back at the office on her desktop computer.

See Also

Designing for Different Screen Sizes on Tablet and Phone
Differences and Limitations When Developing Pages for the Business Central Mobile App
Designing Role Centers
Role Center Behaviors

Instrumenting an Application for Telemetry

2/17/2021 • 2 minutes to read • Edit Online

This article describes how you can implement custom telemetry signals in your application for emitting telemetry data.. This data can then be collected and visualized for analyzing the application against the desired business goals, troubleshooting, and more.

Telemetry overview

One aspect of event logging is collecting data about how the application and your deployment infrastructure is working in order to diagnose conditions and troubleshoot problems that affect operation and performance. For example, this type of event logging includes Business Central Server events and trace events like SQL and AL method (function) traces.

Another aspect of logging is *telemetry*, which is collecting data about how your application functions and how it is being used in production. Telemetry can tell you about specific activities that users perform within the application in the production environment. Telemetry is also a useful tool for troubleshooting, especially instances where you are not able to reproduce the conditions experienced by the user or have no access to the user's environment. Telemetry can be divided into different levels or categories, like: telemetry for engineering, telemetry about the business, telemetry for customers.

Creating custom telemetry signal

There are two different resources where telemetry trace signals can be sent for monitoring and analyzing: Event Log and Microsoft Azure Application Insights. By default, the Business Central application is instrumented to emit several system telemetry trace signals to these destinations. Custom telemetry trace signals enable you to send telemetry data from anywhere in the application code to either of these destinations.

The procedure for creating custom telemetry signals is different for each resource. Your choice might also depend on whether you are developing for Business Central online or on-premises.

REOURCE	DESCRIPTION	ONLINE	ON-PREMISES	MORE INFORMATION

REOURCE	DESCRIPTION	ONLINE	ON-PREMISES	MORE INFORMATION
Application Insights	Create custom telemetry signals that are sent to an Application Insights resource in Azure. Application Insights is a service hosted within Azure that gathers telemetry data for analysis and presentation. Extension developers can specify whether the signal is only sent to the extension publisher or also to the VAR partner telemetry resource. You create these custom trace signals by using the LOGMESSAGE method in code.	✓	✓	See...
Event Log	Create custom telemetry trace signals that are sent to the Event Log of the Business Central Server machine. You create these custom trace signals by using the SENDTRACETAG method in code.		✓	See...

NOTE

Using Application Insights is recommended.

See Also

Monitoring and Analyzing Telemetry
Monitoring Business Central Server Events

Creating Custom Telemetry Traces for Application Insights Monitoring

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

This article explains how to develop extensions to send custom telemetry trace signals to Azure Application Insights for viewing and analyzing.

You can add AL code in extensions to emit messages about activities or operations that users do within the application. At runtime, the messages can be picked up by an Application Insights resource, which you set up in beforehand. In Application Insights, the custom telemetry events are stored in the *traces* table.

Set up Application Insights

An Application Insights resource can be configured in two places:

- In the app.json file of the extension.

  For more information, see Enabling Application Insight on an Extension.

- In the tenant on the Business Central Server service/server.

  For more information, see Enabling Application Insights.

When you create a custom telemetry trace signal, you can specify a telemetry scope. The telemetry scope enables you to send an signal only to the Application Insights resource specified in the extension's app.json. Or to all available resources.

> **NOTE**
> Having Application Insights resources configured is not required to create custom signals in AL code.

Create a custom telemetry trace signal

To create a custom telemetry trace signal, use a LOGMESSAGE method in AL code where you want to trigger the signal. The LOGMESSAGE method defines the information that is sent to Application Insights for a specific operation or activity.

There are two variations of the LOGMESSAGE method. The difference is that one method uses a dictionary object to define custom dimensions for the trace signal. The other method includes two overloads so you don't have to construct a dictionary. You can use these methods in any object, trigger, or method. The methods have the following signatures:

Using a dictionary

The LOGMESSAGE method for using a dictionary for dimensions has the following signature:

```
Session.LogMessage(EventId: String, Message: String, Verbosity: Verbosity, DataClassification:
DataClassification, TelemetryScope: TelemetryScope, CustomDimensions: Dictionary of [Text, Text])
```

Using dimension overloads

The LOGMESSAGE method for using dimension overloads has the following signature:

```
Session.LogMessage(EventId: String, Message: String, Verbosity: Verbosity, DataClassification:
DataClassification, TelemetryScope: TelemetryScope, Dimension1: String, Value1: String [, Dimension2:
String] [, Value2: String])
```

Setting the parameters

Use the parameters to build the dimensions, or columns, that will show for the trace in Application Insights. `Message` and `Verbosity` will appear as general dimensions. All other parameters appear as custom dimensions.

PARAMETER	DESCRIPTION	DIMENSION
EventID	A text string that assigns an identifier to the telemetry trace signal. The tag can consist of letters, numbers, and special characters. Try to make your tags unique. For example, use at least 8 characters or a prefix, like Cronus-0001 and Cronus-0002.	eventId
Message	A text string that specifies the descriptive message for the telemetry trace signal.	message

Verbosity*	An enumeration that specifies the severity level of the telemetry trace signal. The value can be `Critical`, `Error`, `Warning`, `Normal`, or `Verbose`.	severityLevel
		
		`4` = `Critical`
		`3` = `Error`
		`2` = `Warning`
		`1` = `Normal`
		`0` = `Verbose`

DataClassification*	A DataClassification data type that assigns a classification to the telemetry trace signal. For more information, see Data Classifications.	dataClassification
TelemetryScope	Scope of emitting the telemetry. • `extensionpublisher` sends the custom signal only to the Application Insight resource specified in the extension's app.json file • `all` sends the custom signal to Application Insight resource specified in the extension's app.json file and on the tenant.	telemetryScope
CustomDimensions	A dictionary of text that defines the custom dimensions for the trace signal in Application Insights.	
Dimension1	A text string that specifies the name of the custom dimension.	
Value1	A text string that specifies the value of Dimension1.	

PARAMETER	DESCRIPTION	DIMENSION
Dimension2	A text string that specifies the name of the custom dimension.	
Value2	A text string that specifies the value of Dimension2.	

Examples

The following code snippets create simple telemetry trace signals. They create a critical-level telemetry signal that is scoped to the signal publisher. For a simple test of this code, add it to the `OnRun` trigger of a codeunit, and then run the codeunit.

Using a dictionary:

```
trigger OnRun();
var
    CustDimension: Dictionary of [Text, Text];
begin
    CustDimension.Add('result', 'failed');
    CustDimension.Add('reason', 'critical error in code');
    LogMessage('MyExt-0001', 'This is a critical error message', Verbosity::Normal,
DATACLASSIFICATION::SystemMetadata, TelemetryScope::ExtensionPublisher, CustDimension);
end;
```

Using an overload:

```
trigger OnRun();
begin
    LogMessage('MyExt-0001', 'This is a critical error message', Verbosity::Critical,
DATACLASSIFICATION::SystemMetadata, TelemetryScope::ExtensionPublisher, 'result', 'failed', 'reason',
'critical error in code');
end;
```

Design considerations

- For Business Central on-premises, the **Diagnostic Trace Level** setting on the Business Central Server instance controls which signals are sent, based on their severity level.

  If the **Diagnostic Trace Level** is set to **Warning** for example, then **Normal** and **Verbose** signals won't be sent to Application Insights. For more information, see Configuring Business Central Server - General.

- For privacy reasons, events that have a `DataClassification` other than `SystemMetadata` aren't sent to Application Insight resources set up on the tenant. During development of your extension, it's good practice to have a privacy review of the use of LOGMESSAGE calls to ensure that customer data isn't mistakenly leaked into Application Insights resources.

See Also

Instrumenting an Application for Telemetry
LOGMESSAGE method
LOGMESSAGE method
Monitoring and Analyzing Telemetry

Sending Extension Telemetry to Azure Application Insights

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

This article describes how to develop an extension to send telemetry data to Azure Application Insights for monitoring and analyzing. Business Central emits telemetry data for several operations that occur when extension code is run. You can configure an extension to send this telemetry data to a specific Application Insights resource on Microsoft Azure. For an overview about the telemetry with Application Insights, see Monitoring and Analyzing Telemetry.

This feature targets publishers of per-tenant extensions to give them insight into issues in their extensions before partners and customers report them.

Get an Application Insights resource in Azure

The first thing to do is to create an Application Insights resource in Azure if you don't have one. For more information, see Create an Application Insights resource.

The Application Insights resource is assigned an instrumentation key, which you can see on the **Overview** page for the resource in Azure. Copy this key because you'll need it to enable Application Insights in the extension.

Add the Application Insights Key to the extension's app.json

The next step is to add the `"applicationInsightsKey"` setting the extension's app.json as shown:

```
"applicationInsightsKey":"<instrumenation key>"
```

Replace `<instrumenation key>` with your key.

When done, build the extension package, then publish and install it as usual. When the extension is run from Business Central, Application Insights gathers the telemetry data for viewing and analyzing.

See Also

Getting Started with AL
Publishing and Installing Extensions
JSON Files
Viewing telemetry data in Application Insights
LogMessage Method

Creating Custom Telemetry Events for Event Log Monitoring

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central On-Premises.

This article explains how to create custom telemetry trace events in AL code that will be sent to the Event Log of the Business Central Server machine.

> **NOTE**
> The SENDTRACETAG method is marked as obsolete in Business Central 2020 release wave 2 (v17). You can still use it, but we recommend that you send traces to Application Insights using the LOGMESSAGE method instead. For more information, see Creating Custom Telemetry Traces for Application Insights Monitoring.

Create custom telemetry events

To create a custom telemetry event, you use the SENDTRACETAG method in code. You can use the SENDTRACETAG method in any object, trigger, or method. The SENDTRACETAG method has the following syntax:

```
SENDTRACETAG(Tag, Category, Verbosity, Message[, DataClassification])
```

You use the parameters to define the information about the telemetry trace event. This information is can be consumed by event logging tools, and presented in different ways.

PARAMETER	DESCRIPTION
Tag	A text string that assigns an identifier to the telemetry trace event. The tag can consist of letters, numbers, and special characters. Business Central system telemetry events use an auto-generated, auto-incremented, 7-character tag that includes numbers and letters, such as 000002Q. and 000013P. Try to make your tags unique from these telemetry event tags by, for example, using at least 8 characters or a prefix, like Cronus-0001 and Cronus-0002.
Category	A text string that assigns the telemetry trace event to a category that you define. For example, you could have a category for upgrading, user activity, or reporting.
Verbosity	An enumeration that specifies the severity level of the telemetry trace event. The value can be Critical, Error, Warning, Normal, or Verbose. This severity level can be used by Business Central Server to filter out lower-level telemetry trace events from being emitted. See Viewing and collecting telemetry data.
Message	A text string that specifies the descriptive message for the telemetry trace event.

PARAMETER	DESCRIPTION
DataClassification	A DataClassification data type that assigns a classification to the telemetry trace event. For more information, see Data Classifications.

For example, the following code creates simple telemetry trace events for the five different severity levels.

```
SENDTRACETAG('Cronus-0001', 'Action', VERBOSITY::Critical, 'This is a critical message.',
DATACLASSIFICATION::CustomerContent);
SENDTRACETAG('Cronus-0002', 'Action', VERBOSITY::Error, 'This is an error message.',
DATACLASSIFICATION::EndUserIdentifiableInformation);
SENDTRACETAG('Cronus-0003', 'Action', VERBOSITY::Warning, 'This is a warning message.',
DATACLASSIFICATION::AccountData);
SENDTRACETAG('Cronus-0004', 'Action', VERBOSITY::Normal, 'This is an informational message.',
DATACLASSIFICATION::OrganizationIdentifiableInformation);
SENDTRACETAG('Cronus-0005', 'Action', VERBOSITY::Verbose, 'This is a verbose message.',
DATACLASSIFICATION::SystemMetadata);
```

For a simple test of this code, add it to the `OnRun` trigger of a codeunit, and then run the codeunit. Of course, you can also call the code from other objects, triggers or functions as well.

View and collect telemetry data

Viewing and collecting telemetry data is done the same way as with other trace events emitted by Business Central, for example, by using tools like Event Viewer, Performance Monitor, PerfView, or logman.

- In Event Viewer, telemetry trace events can be viewed from **Applications and Services Logs**, in the **Microsoft** > **DynamicsNAV** > **Common** folder. The custom telemetry trace events are recorded in the **Admin** folder. You should be aware that only events with severity level of Warning, Error, and Critical will appear.

  For more information, see Monitoring Business Central Server Events Using Event Viewer.

- With other tools like Performance Monitor, PerfView, and logman, you can collect telemetry data by using **Microsoft-DynamicsNAV-Common** as the event trace provider.

  For more information, see Get Started Monitoring Events.

> **IMPORTANT**
> The Business Central Server instance includes a configuration setting called **Diagnostic Trace Level** (`TraceLevel` in the customsettings.config file) that enables you to specify the lowest severity level of telemetry events to be recorded in the event log, or even turn off telemetry event logging altogether. If you do not see the expected events, then verify the Business Central Server instance configuration with an administrator. For information, see Configuring Business Central Server.

See Also

Instrumenting an Application for Telemetry
Monitoring and Analyzing Telemetry
Monitoring Business Central Server Events

Exporting Permission Sets

2/17/2021 • 2 minutes to read • Edit Online

Permission sets that exist in Dynamics 365 Business Central can be exported and packaged for your extension directly from the client, instead of defining XML by hand.

To export permission sets from Dynamics 365 Business Central

1. In Dynamics 365 Business Central, search for **Permission Sets**, and then choose the relevant link.
2. On the **Permission Sets** page, choose the permissions that you want to export, and then choose **Export Selected Permissions**.
3. In the **Export Permission Sets** dialog, choose to export permission sets only for the application, only for the tenant, or for both.
4. Save the file to your extension folder.
5. Delete the permission sets from Dynamics 365 Business Central.

You can generate a permission set file which contains permissions to all the files in your extension. This will make it easier to start setting up permissions for your app. You can do this by simply creating an extension with some objects as described below.

To export permission sets using Visual Studio Code

1. In Visual Studio Code, open your extension with objects; pages, reports, tables, queries, codeunits, and/or XMLports.
2. Open the command palette using the `Ctrl+Shift+P` keys and select the **AL: Generate permission set containing current extension objects** command.

   > **NOTE**
   > If you do this repeatedly, Visual Studio Code will probe for overwriting the file, there is no support for merging manual corrections into newly generated content.

3. Publish the app.

Now, you have the XML file with default permissions to all your objects.

Example from Visual Studio Code

The following example illustrates the generated .xml file from **MyProject** which contains a table with objectID 50106 and two object types are generated; `<ObjectType>0</ObjectType>` is `TableData` and `<ObjectType>1</ObjectType>` is `Table`.

> **NOTE**
> The maximum length of the RoleID attribute is 20 characters. If the RoleID exceeds this length, then the XML schema validation fails and the corresponding permission set file is not included in the resulting app package. You will get a warning in the Visual Studio output panel.

```
<?xml version="1.0" encoding="utf-8"?>
<PermissionSets>
  <PermissionSet RoleID="MyProject" RoleName="MyProject">
    <Permission>
      <ObjectID>50106</ObjectID>
      <ObjectType>0</ObjectType>
      <ReadPermission>1</ReadPermission>
      <InsertPermission>1</InsertPermission>
      <ModifyPermission>1</ModifyPermission>
      <DeletePermission>1</DeletePermission>
      <ExecutePermission>0</ExecutePermission>
      <SecurityFilter />
    </Permission>
    <Permission>
      <ObjectID>50106</ObjectID>
      <ObjectType>1</ObjectType>
      <ReadPermission>0</ReadPermission>
      <InsertPermission>0</InsertPermission>
      <ModifyPermission>0</ModifyPermission>
      <DeletePermission>0</DeletePermission>
      <ExecutePermission>1</ExecutePermission>
      <SecurityFilter />
    </Permission>
 </PermissionSet>
 </PermissionSets>
```

Object type mapping

The mapping of object types in the XML such as `<ObjectType>0</ObjectType>` generated from Visual Studio Code is the following:

OBJECT TYPE	NUMBER
TableData	0
Table	1
Report	3
Codeunit	5
XMLPort	6
Page	8
Query	9
FieldNumber	11
PageExtension	14
TableExtension	15
Enum	16
EnumExtension	17

OBJECT TYPE	NUMBER
Profile	18
ProfileExtension	19

See Also

Permissions on Database Objects
Permissions Property
TestPermissions Property

Getting started with Microsoft .NET Interoperability from AL

2/17/2021 • 2 minutes to read • Edit Online

You can call .NET type members, including methods, properties, and constructors, from AL code. In this article we will guide you through the process of creating an extension that uses .NET types.

> **IMPORTANT**
>
> .NET Interoperability is only available on-premise. If you want to use this functionality, you must set the `"target": "OnPrem"` in the `app.json` file. For more information, see JSON Files.

Enabling .NET Interoperability

.NET interoperability can only be used by applications that target on-premise deployments. See JSON Files for more information on how to set the correct compilation target and Compilation Scope Overview.

Declaring a .NET package

Using a .NET type in AL is a two-step process. First, you must declare the type in a **dotnet** package, and then reference it from code using the **DotNet** type.

You start by declaring an empty **dotnet** package in your extension. See the example snippet below.

```
dotnet
{

}
```

It is recommended to have only one package per extension that contains all the .NET types which you will be using.

You continue by adding a declaration of the assembly that you will be referencing. For this example, we will use the `mscorlib` assembly that contains the core .NET types. A **dotnet** package can contain an unlimited number of assembly declarations. The name of the assembly must be the one defined in the assembly's manifest. See the following example snippet.

```
dotnet
{
    assembly(mscorlib)
    {
    }
}
```

By default, the compiler only knows about the location of the `mscorlib` assembly. You can reference any compatible assembly by providing the compiler with a path to the assembly's containing folder. This can be achieved by adding the path to assembly's containing folder to the `"al.assemblyProbingPaths"` setting. Open the Command Palette **Ctrl+Shift+P** and choose either **User Settings** or **Workspace Settings** and specify the `al.assemblyProbingPaths` setting. For example:

```
"al.assemblyProbingPaths": [
        "./.netpackages",
        "C:/Program Files/Assemblies"
    ]
```

> **NOTE**
> Any update to an assembly's code is not automatically detected by the compiler. If an assembly has changed, then you must restart your development environment.

You continue by adding a reference to a type from the referenced assembly. In this example, we will use `System.DateTime` from `mscorlib` and we will give it the alias `MyDateTime`. The type must be referenced using its fully-qualified name. The alias is used for referencing the .NET type from code. If an alias is not provided, the compiler will use the .NET type name. A .NET assembly declaration can contain any number of type declarations. See the example below.

```
dotnet
{
    assembly(mscorlib)
    {
        type(System.DateTime; MyDateTime){}
    }
}
```

Using a .NET type from AL code

From this point on, we can reference the .NET type from AL code using its given alias, as shown in the example below.

```
dotnet
{
    assembly(mscorlib)
    {
        type(System.DateTime; MyDateTime){}
    }
}

pageextension 50100 CustomerListExt extends "Customer List"
{
    trigger OnOpenPage();
    var
        now: DotNet MyDateTime;
    begin
        now := now.UtcNow();
        Message('Hello, world! It is: ' + now.ToString());
    end;
}
```

Publishing your extension

The AL Language extension, including the AL compiler, and the server to which you publish your code are completely decoupled. When publishing, the server re-compiles your code and tries to resolve all the references to external assemblies. In order for the compilation to succeed, the server must be able to locate and load all the referenced assemblies and types.

The server will search the global assembly cache (GAC), the **Add-ins** folder, and the **Add-in** table. You must

manually install any custom assembly in one of these locations.

See Also

Getting Started with AL
.NET Control Add-Ins
Subscribing to Events in a .NET Framework Type
Serializing .NET Framework Types
AL Language Extension Configuration

.NET Control Add-Ins

2/17/2021 • 3 minutes to read • Edit Online

In Dynamics 365 Business Central on-premises you can use existing .NET and Javascript control add-ins from the AL Language through .NET interoperability. It is recommended that you convert your existing .NET and Javascript add-ins to native AL control add-ins that are supported both on-premises and in the cloud. For more information about native AL control add-ins, see Control Add-In Object.

To declare the usage of a .NET or Javascript add-in in AL, you need three critical pieces of information about the .NET type that represent the interface of the add-in. These are the name of the assembly containing the add-in, the name of the control add-in, and the name of the class that implements the control add-in. We will show how to retrieve this information for the `Microsoft.Dynamics.Nav.Client.PingPong` control add-in that ships with Business Central.

The name of the assembly can be retrieved from the AssemblyName element in the `.csproj` file associated with the .NET project that represents the control add-in. In this case the name of the assembly is `Microsoft.Dynamics.Nav.Client.PingPong`.

> **NOTE**
> If you do not have access to the `.csproj file`, you can determine the name of the assembly by following the instructions in How to: Determine an Assembly's Fully Qualified Name.

The following code sample contains the stub definition of the `Microsoft.Dynamics.Nav.Client.PingPong` .NET add-in.

```
namespace Microsoft.Dynamics.Nav.Client.PingPong
{

/// <summary>
/// Add-in for pinging the server from the client. The client will respond with a pong.
/// </summary>
[ControlAddInExport("Microsoft.Dynamics.Nav.Client.PingPong")]
public class PingPongAddIn : WinFormsControlAddInBase
{

        /// <summary>
        /// Event will be fired when the AddIn is ready for communication through its API
        /// </summary>
        [ApplicationVisible]
        public event MethodInvoker AddInReady;

        /// <summary>
        /// Event will be fired when the specified time by the ping has elapsed.
        /// </summary>
        [ApplicationVisible]
        public event MethodInvoker Pong;

        /// <summary>
        /// Starts the ping process.
        /// </summary>
        /// <param name="milliseconds">Number of milliseconds before ponging.</param>
        /// <remarks>If a milliseconds are less than the minimum then the MinimumValue is used.</remarks>
        [ApplicationVisible]
        public void Ping(int milliseconds)
        {
            ...
        }
}

}
```

The next needed piece of information is the namespace-qualified name of the type annotated with the `ControlAddInExport` attribute. This is the type that provides the implementation of the control add-in and which exposes members annotated with the `ApplicationVisible` attribute to the AL runtime. In this example this is `Microsoft.Dynamics.Nav.Client.PingPong.PingPongAddIn`.

The `ControlAddInExport` attribute's constructor takes as an argument the name of the control add-in, as represented in the runtime, and in existing C/AL code. In this example, the name of the control add-in is `Microsoft.Dynamics.Nav.Client.PingPong`. This was the last component needed to construct a declaration for this .NET control add-in in AL. The name of the assembly is used in creating the `assembly` construct, the namespace-qualified name of the type is used as the first element in the `type` declaration, and the name of the control add-in is used as the alias of the type. You complete the declaration by setting the `IsControlAddIn` property to true. This property is used to tell the AL compiler to treat the given type declaration as a .NET control add-in declaration.

Remember to add the setting **"AL: Assembly Probing Paths"** in the **User Settings** or **Workspace Settings** specifying the path of the folder containing the assembly so that the compiler can access it. For more information, see Getting started with Microsoft .NET Interoperability from AL.

```
dotnet
{
    assembly("Microsoft.Dynamics.Nav.Client.PingPong")
    {
        type("Microsoft.Dynamics.Nav.Client.PingPong.PingPongAddIn"; PingPongAddIn)
        {
            IsControlAddIn = true;
        }
    }
}
```

You can now use the `Microsoft.Dynamics.Nav.Client.PingPong` from AL, just as you use a native control add-in.

```
page 50100 MyPage
{
    layout
    {
        area(Content)
        {
            usercontrol(PingPongControl; PingPongAddIn)
            {
                trigger Pong()
                begin
                    Message('Pong received.');
                end;

                trigger AddInReady()
                begin
                    Message('Ready');
                end;
            }
        }
    }
}
```

Remarks

Only members of the .NET type implementing the control add-in that are annotated with the `ApplicationVisibleAttribute` will be accessible from AL. Usages of .NET control add-ins in C/AL are automatically converted to AL by the Txt2Al conversion tool, but the code will only compile, if you manually insert the declaration of the control add-in, as outlined above.

If within the same project you have a native AL control add-in and a .NET add-in with the same name, the .NET add-in will be the one used.

See Also

Getting Started with AL
Control Add-In Object
Getting started with Microsoft .NET Interoperability from AL
Subscribing to Events in a .NET Framework Type
Serializing .NET Framework Types
How to: Determine an Assembly's Fully Qualified Name
AL Language Extension Configuration

Subscribing to Events in a .NET Framework Type

2/17/2021 • 2 minutes to read • Edit Online

With .NET Framework interoperability in Dynamics 365 Business Central objects, you can configure a DotNet variable to subscribe to events that are published by a .NET Framework type. Events are handled by triggers in the AL code of the Business Central object.

You start by declaring in AL the usage of two .NET types from the `System` assembly. The first type is `System.Timers.Timer` and it will be used to generate .NET events. The second one is called `System.Timers.ElapsedEventArgs` and it is required for creating a subscriber to the `Elapsed` event emitted by the `Timer` type.

```
dotnet
{
    assembly(System)
    {
        type(System.Timers.Timer; MyTimer) {}
        type(System.Timers.ElapsedEventArgs; MyElapsedEventArgs) {}
    }
}
```

You can only subscribe to events that are emitted by global variables of the .NET type marked with the `WithEvents` attribute. For all the global variables that are marked with this attribute, the compiler will expose the events available on the type as triggers on the variable. The syntax for declaring these triggers is `{VariableName}::{EventName}(...ParameterList)`, but IntelliSense will offer suggestions for the event name and autocomplete the parameter list.

```
pageextension 50101 CustomerListExt extends "Customer List"
{
    var
        [WithEvents]
        timer: DotNet MyTimer;

    trigger OnOpenPage()
    begin
        SetupTimer();
    end;

    procedure SetupTimer()
    begin
        timer := timer.Timer(2000);
        timer.AutoReset := true;
        timer.Enabled := true;
        timer.Start();
    end;

    trigger timer::Elapsed(sender: Variant; e: DotNet MyElapsedEventArgs)
    begin
     // Print a message when this event is published
        Message('%1', e.SignalTime());
        timer.Stop();
    end;
}
```

See Also

Getting started with Microsoft .NET Interoperability from AL
.NET Control Add-Ins
Serializing .NET Framework Types
Method Attributes
AL Language Extension Configuration

Serializing .NET Framework Types

2/17/2021 • 6 minutes to read • Edit Online

In Microsoft .NET Framework, serialization is the process of converting an object into a format that can be transmitted across a network connection. Microsoft .NET Framework interoperability uses serialization for communication between client-side .NET Framework objects and server-side .NET Framework objects. When you configure DotNet variables in a Dynamics 365 Business Central object, you can specify .NET Framework objects to target either the Business Central Windows client or Business Central Server. In some cases, a client-side object and a server-side object must communicate and share data, such as return values and parameters. The serialization occurs when the following conditions are true:

- When a server-side object is assigned to a client-side object, and vice-versa.
- When a server-side object is passed as a parameter in a method call from the server to a client-side object, and vice-versa.

Serialization requires that the .NET Framework types that are used by the DotNet variables are serializable. Many types in the Microsoft .NET Framework class library are already serializable. If you are using a .NET Framework type that cannot be serialized, then you must modify the type to make it serializable.

> **IMPORTANT**
> For the Business Central Web client, you cannot implement Microsoft .NET Framework interoperability objects that target the client.

Making a Type Serializable

There are two ways that you can make a .NET Framework type serializable. You can implement basic serialization by applying the System.SerializableAttribute attribute to the type or you can implement custom serialization by using System.Runtime.Serialization.ISerializable interface.

Basic Serialization Using SerializableAttribute

Basic serialization uses the .NET Framework to automatically serialize an object. To implement basic serialization on a type, you decorate the type with the SerializableAttribute class as shown in the following example.

```
[Serializable]
public class MyObject
{
    code
}
```

This method requires that you have access to the source code of the .NET Framework assembly.

Basic Serialization of Date Fields

You can use basic serialization only if all data fields in the type are serializable. Fields that are calculated at runtime cannot be serialized. If a field cannot be serialized, then at runtime, the serialization process will throw an exception and the AL code execution will fail.

You can exclude fields from the serialization process by decorating the field with the System.NonSerializedAttribute class.

Custom Serialization Using ISerializable Interface

With custom serialization, you can create an object that controls the serialization of types in another object. This method is useful when you do not have access to the source code of the assembly that contains the .NET Framework types that you are implementing with .NET Framework interoperability. The custom object specifies which types will be serialized and how serialization will be done.

To implement custom serialization, you create a class that implements the ISerializable interface, and decorate the class with SerializableAttribute. In most cases, you must also implement the System.Runtime.Serialization.ISerializable.GetObjectData method and a special constructor that is used when the source object is deserialized. You use the **GetObjectData** method to populate the SerializationInfo the data that is required to serialize the source object at runtime.

The common language runtime calls the constructor during deserialization to construct a replica of the source object. The constructor takes two parameters, a **SerializationInfo** type and a System.Runtime.Serialization.StreamingContext type. The **StreamingContext** parameter describes the source and destination of a given serialized stream.

Custom Serialization Example

The following code example demonstrates a custom serialization object that implements the basic functionality that is required for compliance with the **ISerializable** interface. In the first procedure of this example, you create a .NET Framework assembly that includes a serializable type. In the second procedure, in the Business Central development environment, you create a codeunit that includes two DotNet variables for the serializable type. You set one variable to target the Business Central Windows client and the other to target the Business Central Server. In AL code, you add code that transfers the value for the DotNet variable on the Business Central Server to the Business Central Windows client. You will also add code that verifies that the data transfer is successful.

To create the custom serialization object

1. In Microsoft Visual Studio, create a C# Class Library project called *SerializationSample*.
2. Add the following code.

```
using System;
using System.Runtime.Serialization;

[Serializable]
public class SerializeWithInterface : ISerializable
{
     // Defines a field that will not be serialized.
     [NonSerialized]
     private string notSerializedField;
     // Defines two fields that will be serialized.
     private int serializedIntField;
     private string serializedStringField;
     // Specifies literal field names.
     private const string serializedIntFieldName = “serializedIntField”;
     private const string serializedStringFieldName = “serializeStringField”;

     // Defines a default constructor that initializes the object with default values.
     public SerializeWithInterface()
     {
          this.serializedIntField = 1;
          this.notSerializedField = string.Empty;
          this.serializedStringField = string.Empty;
     }

     // Defines the protected constructor that is required by the ISerializable interface.
     // Data is stored in the SerializationInfo argument and is extracted using the GetValue method.
     protected SerializeWithInterface(SerializationInfo si, StreamingContext context)
     {
          this.serializedStringField = (string)si.GetValue(serializedStringFieldName, typeof(string));
          this.serializedIntField = (int)si.GetValue(serializedIntFieldName, typeof(int));
     }

     // Fills the SerializationInfo object with data that must to be sent to the replicated object.
     // Data is stored in the dictionary using the AddValue method.
     // The SerializationInfo object is a key/value dictionary.
     // The name you use to store the value must match the name used in the constructor.
     // For this reason, a string constant is used.
     public void GetObjectData(SerializationInfo info, StreamingContext context)
     {
          info.AddValue(serializedStringFieldName, this.serializedStringField);
          info.AddValue(serializedIntFieldName, this.serializedIntField);
     }

     // Remaining class implementation is irrelevant for the serialization process.
     // The SerializedStringField property is used by AL code to verify that the contained data is
transferred between the server and client.
     // SerializedStringField gets or sets the internal serialized string field.
     public string SerializedStringField
     {
          get { return this.serializedStringField; }
          set { this.serializedStringField = value; }
     }
 }
```

3. Build the project.
4. Copy the SerializationSample.dll to the **Add-ins** folder of the Business Central Windows client and Business Central Server installation folders.

By default, the path of the Business Central Windows client installation folder is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\RoleTailored Client\Add-ins.

By default, the path of the Business Central Server installation folder is C:\Program Files\Microsoft Dynamics 365 Business Central\130\Service\Add-ins.

To test the serialization object

1. In the AL Development Environment, add the following code.

```
dotnet
{
     assembly(SerializationSample)
     {
          type(SerializationSample.SerializeWithInterface; SerializeWithInterface){}
     }
}

codeunit 50101 SerializationSample
{
    procedure Testing()
    var
        ServerObject: DotNet SerializeWithInterface;
        ClientObject: DotNet SerializeWithInterface;

    begin
        // Constructor that instantiates the ServerObject object on the server.
        ServerObject := ServerObject.SerializeWithInterface();
        // Constructor that instantiates the ClientObject object on the server.
        ClientObject := ClientObject.SerializeWithInterface();
        // Assign unique values to the data members in the two objects.
        ServerObject.SerializedStringField := 'ServerSide';
        ClientObject.SerializedStringField := 'ClientSide';
        // Transfer the server object to the client object using serialization.
        ClientObject := ServerObject;
        // Test if the objects contain the same data.
        If ClientObject.SerializedStringField <> ServerObject.SerializedStringField then
            Error('Client object does not match the server object.');
            Message('Server data has been serialized to the client object.');
    end;
}
```

The line that contains assignment of the **ServerObject** to the **ClientObject** causes the serialization process to run. When completed, the message **Server data has been serialized to the client object** appears, which verifies that the server object has been transferred to the client object.

See Also

Getting Started with AL
Getting started with Microsoft .NET Interoperability from AL
.NET Control Add-Ins
Subscribing to Events in a .NET Framework Type
Using Designer
AL Language Extension Configuration

Implementing the Camera in AL

2/17/2021 • 3 minutes to read • Edit Online

You can access the camera of a device from the Business Central Web client in the browser and from the Business Central Mobile App. This allows the user to take pictures and handle them directly from the same device, and in that way, improve accuracy of capturing data closest to the source, and reduce end-to-end time to perform tasks.

You can also add access to the camera to a specific page from the AL Language development environment. For a Dynamics 365 Business Central existing implementation of this, see the `Picture` factbox on the `Item Card`, which lets you take a picture of a specific item and store it together with the item.

> **IMPORTANT**
>
> The camera access is only available on devices that have a camera.

Example

This example illustrates how to implement the camera capability on a page in AL. The example implements three actions to take a picture: **Take Picture**, **Take Picture High Quality**, and **Take Picture Low Quality**. However, it does not include code that saves the picture to the database.

The example also shows how to specify options for the camera functionality such as picture quality or source type. For more information about the different options that can be set for the camera, see CameraOptions Overview.

> **NOTE**
>
> To enable the camera functionality, it is required that you add the path of the folder containing the `"Microsoft.Dynamics.Nav.ClientExtensions"` assembly on the **AI: Assembly Probing Paths** setting on the **User Settings** or **Workspace Settings** so the compiler can access it. For more information, see Getting started with Microsoft .NET Interoperability from AL.

The following code will create two variables; the `CameraAvailable` variable is a **Boolean** that checks whether the current device has a camera. The `Camera` variable is a **DotNet** type that gets instantiated by adding code to the `OnOpenPage` trigger. Then, it will add the actions to the page that lets the user start the camera. Finally, the trigger `Camera::PictureAvailable` is defined to handle the incoming picture.

```
page 50101 "Card with Camera Capability"
{

    Caption = 'Card Page';
    PageType = Card;
    RefreshOnActivate = true;
    SourceTable = "Test Table";

    layout
    {
        area(content)
        {
            //...
        }
```

```
    }

    actions
    {

        area(Processing)
        {
            action(TakePicture)
            {
                Visible = CameraAvailable;
                Promoted = true;
                PromotedCategory = Process;
                PromotedIsBig = true;
                Image = Camera;

                trigger OnAction()
                begin
                    Camera.RequestPictureAsync();
                end;
            }

            action(TakePictureHigh)
            {
                Visible = CameraAvailable;
                Promoted = true;
                PromotedCategory = Process;
                PromotedIsBig = true;
                Image = Camera;

                trigger OnAction()
                begin
                    CameraOptions := CameraOptions.CameraOptions();
                    CameraOptions.Quality := 100;
                    Camera.RequestPictureAsync(CameraOptions);
                end;
            }

            action(TakePictureLow)
            {
                Visible = CameraAvailable;
                Promoted = true;
                PromotedCategory = Process;
                PromotedIsBig = true;
                Image = Camera;

                trigger OnAction()
                begin
                    CameraOptions := CameraOptions.CameraOptions();
                    CameraOptions.Quality := 10;
                    Camera.RequestPictureAsync(CameraOptions);
                end;
            }
        }

    }

    trigger OnOpenPage()
    begin
        // The IsAvailable() enables the camera functionality based on its presence.
        if Camera.IsAvailable() then begin
            Camera := Camera.Create();
            CameraAvailable := True;
        end;
    end;

    // The PictureName contains the name of the file including its extension on the device.
    // The naming scheme depends on the device platform.
    // The PictureFilePath contains the path to the picture in a temporary folder on the server for the
current user.
```

```
    // The PictureAvailable trigger handles the picture for when the camera has captured it and it has been
uploaded.
    trigger Camera::PictureAvailable(PictureName: Text; PictureFilePath: Text)
    begin
        IncomingFile.Open(PictureFilePath);
        Message('Picture size: %1', IncomingFile.Len());
        IncomingFile.Close();
        // It is important to clean up by using the File.Erase command to avoid accumulating image files.
        File.Erase(PictureFilePath);
    end;

    var
        [RunOnClient]
        [WithEvents]
        Camera: DotNet UT_CameraProvider;
        CameraOptions: DotNet UT_CameraOptions;
        // Checks whether the current device has a camera.
        CameraAvailable: Boolean;
        IncomingFile: File;
}

dotnet
{
    assembly("Microsoft.Dynamics.Nav.ClientExtensions")
    {

        type("Microsoft.Dynamics.Nav.Client.Capabilities.CameraProvider"; "UT_CameraProvider")
        {

        }

        type("Microsoft.Dynamics.Nav.Client.Capabilities.CameraOptions"; "UT_CameraOptions")
        {

        }
    }
}
```

For information about troubleshooting access to camera, see Troubleshooting: Camera and Location.

See Also

Getting started with Microsoft .NET Interoperability from AL
Implementing Location in AL
CameraOptions Overview
RunOnClient property
WithEvents property

Implementing Location in AL

2/17/2021 • 2 minutes to read • Edit Online

You can access the location information of a device from the Business Central Web client in the browser and from the Business Central Mobile App. This functionality could be useful in scenarios such as calculating routes from the current location or planning the next round of customer visits based on their addresses.

You can also add access to location information to a specific page from the AL Language development environment. For a Dynamics 365 Business Central existing implementation of this, see the `Show On Map` link on the `Customer Card`, which displays a map that shows where your customer is located based on the GPS coordinates and gives directions to reach its location.

> **IMPORTANT**
> Location information is only available on devices that are able to obtain location coordinates, such as using GPS capabilities.

Example

This example illustrates how to implement the location capability on a page in AL. The example implements a **GetLocation** action on a page that returns the coordinates of the current customer's address, but does not save this information to the database.

The example also shows how to specify options for the location functionality such as setting a timeout or enabling high accuracy. For more information about the different options that can be set for location, see LocationOptions Overview.

> **NOTE**
> To enable the location functionality, it is required that you add the path of the folder containing the `"Microsoft.Dynamics.Nav.ClientExtensions"` assembly on the **AL: Assembly Probing Paths** setting on the **User Settings** or **Workspace Settings** so the compiler can access it. For more information, see Getting started with Microsoft .NET Interoperability from AL.

The following code will create two variables; the `LocationAvailable` variable is a **Boolean** that checks whether the current device has location capabilities. The `Location` variable is a **DotNet** type that gets instantiated by adding code to the `OnOpenPage` trigger. Then, it will add an action to the page that lets the user retrieve the location information. Finally, the trigger `Location::LocationChanged` is defined to handle the incoming location information.

```
page 50101 "Card with Location Capability"
{

    Caption = 'Card Page';
    PageType = Card;
    RefreshOnActivate = true;
    SourceTable = "Test Table";

    layout
    {
        area(content)
        {
```

```
            //...
        }
    }

    actions
    {
        area(Processing)
        {
            action(GetLocation)
            {
                Visible = LocationAvailable;
                Promoted = true;
                PromotedCategory = Process;
                PromotedIsBig = true;
                Image = Map;

                trigger OnAction()
                begin
                    LocationOptions := LocationOptions.LocationOptions;
                    LocationOptions.EnableHighAccuracy();
                    Location.RequestLocationAsync();
                end;
            }
        }

    }

    trigger OnOpenPage()
    begin
        if Location.IsAvailable() then begin
            Location := Location.Create();
            LocationAvailable := true;
        end;
    end;

    trigger Location::LocationChanged(Location: DotNet Location)
    begin
        // Location.Status can be:
        //      0 = Available
        //      1 = NoData (no data could be obtained)
        //      2 = TimedOut (location information not obtained in due time)
        //      3 = NotAvailable (for example user denied app access to location)
        if Location.Status = 0 then
            Message('Your position: %1 %2', Location.Coordinate.Latitude, Location.Coordinate.Longitude)
        else
            Message('Position not available');
    end;

    var
        [RunOnClient]
        [WithEvents]
        Location: DotNet LocationProvider;
        LocationAvailable: Boolean;
        LocationOptions: DotNet UT_LocationOptions;
}

dotnet
{
    assembly("Microsoft.Dynamics.Nav.ClientExtensions")
    {

        type("Microsoft.Dynamics.Nav.Client.Capabilities.LocationProvider"; LocationProvider)
        {

        }

        type("Microsoft.Dynamics.Nav.Client.Capabilities.Location"; Location)
        {
```

```
        }

        type("Microsoft.Dynamics.Nav.Client.Capabilities.LocationOptions"; UT_LocationOptions)
        {

        }
    }
}
```

For information about troubleshooting access to location information, see Troubleshooting: Camera and Location.

See Also

Getting started with Microsoft .NET Interoperability from AL
LocationOptions Overview
Troubleshooting: Camera and Location
Implementing the Camera in AL
RunOnClient property
WithEvents property

Testing the Application Overview

2/17/2021 • 7 minutes to read • Edit Online

Before you release your Business Central application, you should test its functionality to ensure it works as expected. Testing is an iterative process. It's important to create repeatable tests, and helpful to create tests that can be automated. This article describes the features in Business Central that help you test the business logic in your application, and it provides some best practices for testing.

For a walkthrough concerning advanced extension testing, see Testing the Advanced Extension Sample.

Business Central includes the below features to help you test your application.

Environment testing support and limitations

The extent to which you can run automated tests will depend on your Business Central solution type and environment. The following table gives an overview.

BUSINESS CENTRAL SOLUTION	ENVIRONMENT	TESTING ALLOWED	MORE DETAILS
Online	Production		Running tests isn't allowed because it might have an adverse effect on your business. Testing can incidentally invoke external systems, like CDS, PayPal, and web hook subscriptions. Invoking these systems may slow down the solution for other users or cause data corruption.
	Sandbox	✔	You can use a sandbox environment to run tests manually to verify functionality on an environment. Running a large number of tests or tests that take a long time (more than 15 minutes per test method) isn't allowed. It's recommended that you don't run tests more that one or two hours a day.

BUSINESS CENTRAL SOLUTION	ENVIRONMENT	TESTING ALLOWED	MORE DETAILS
On-premises	Production	✅	For Business Central on-premises, running automated tests is only possible with a Partner license or a license that includes the Application Builder module. You can disable the ability to run tests by turning off **Enable Test Automation** (TestAutomationEnabled) on the Business Central Server instance. For more information, see Configuring Business Central Server - General Settings.
	Container-based development environment	✅	This setup should be the default environment for running large number of tests or setting up CI/CD gates. For more information, see Running a Container-Based Development Environment or Running Tests In Containers.

Test Codeunits and Test Methods

You write tests as AL code in methods of codeunits that are configured to be test codeunits. Test codeunits have the SubType Property set to **Test**. There are three types of methods that you can add in a test codeunit: test, handler, and normal. Each method type is used for a specific purpose and behaves differently. When a test codeunit runs, it executes the **OnRun** trigger, and then executes each test method in the codeunit. The outcome of a test method is either SUCCESS or FAILURE.

This pattern doesn't apply to test isolation and isn't recommended as a method for running tests.

For more information about test codeunits and test methods, see Test Codeunits and Test Methods.

Test Runner Codeunits

You use test runner codeunits to manage the execution of test codeunits and to integrate with other test management, execution, and reporting frameworks. By integrating with a test management framework, you can automate your tests and enable them to run unattended.

Test runner codeunits are codeunits that have the SubType Property set to **TestRunner**.

Test runner codeunits include the following triggers:

- OnRun Trigger
- OnBeforeTestRun Trigger
- OnAfterTestRun Trigger

In the **OnRun** trigger you enter the code to run the codeunits. It runs when you execute the codeunit and before the test methods run. You can use the **OnBeforeTestRun** and the **OnAfterTestRun** triggers to do preprocessing and postprocessing, such as initialization or logging test results.

For more information about test runner codeunits, see Test Runner Codeunits.

> **TIP**
> You can reuse test runners from Test Runner module in the Microsoft/ALAppExtensions GitHub repo. You can also use the repo to request for the new functionality.

Test Pages

Test pages mimic actual pages but don't present any UI on a client computer. Test pages let you test the code on a page by using AL to simulate user interaction with the page.

There are two types of test pages:

- TestPage, which is a regular page and can be any kind of page. It includes page parts and subpages as well.
- TestRequestPage, which represents the request page on a report.

You access the page's fields and properties or a field by using the dot notation. You open and close test pages, do actions on the test page, and navigate around the test page by using AL methods. For more information, see Testing Pages.

UI Handlers

To create tests that can be automated, you must handle cases when user interaction is requested by code that is being tested. UI handlers run instead of the requested UI. UI handlers provide the same exit state as the UI. For example, a method that has the ConfirmHandler Attribute set handles Confirm Method calls. If code that is being tested calls the Confirm Method, then the **ConfirmHandler** method is called instead of the Confirm Method. You write code in the **ConfirmHandler** method to verify that the expected question is displayed by the Confirm Method. You write AL code to return the relevant reply.

You create a specific handler for each page that you want to handle and a specific report handler for each report that you want to handle.

If you run a test codeunit from a test runner codeunit, then any unhandled UI in the test methods of the test codeunit causes a failure of the test. If you do not run the test codeunit from a test runner codeunit, then any unhandled UI is displayed as it typically would.

For more information, see Creating Handler Methods.

ASSERTERROR Keyword

You use `AssertError` statements in test methods to test how your application behaves under failing conditions. These statements are called positive and negative tests. The `AssertError` keyword specifies that an error is expected at run time in the statement that follows the `AssertError` keyword.

If a simple or compound statement that follows the `AssertError` keyword causes an error, then execution successfully continues to the next statement in the test method. You can get the error text of the statement by using the GETLASTERRORTEXT Method.

If a statement that follows the `AssertError` keyword doesn't cause an error, then the `AssertError` statement

causes the following error and the test method that is running produces a FAILURE result.

> **IMPORTANT**
> Use ASSERTERROR in test code only. It isn't allowed or supported in production code.

Example

To create a test method to test the result of a failure of a CheckDate method that you've defined, you can use the following code. This example requires that you create a method called CheckDate. This method checks whether the date is valid for the customized application. You also create the following text constant, *Date* variable InvalidDate, and *Text* variable InvalidDateErrorMessage.

```
InvalidDate := 010184D;
InvalidDateErrorMessage := 'The date is outside the valid date range.';
ASSERTERROR CheckDate(InvalidDate);
if GETLASTERRORTEXT <> InvalidDateErrorMessage then
  ERROR('Unexpected error: %1', GETLASTERRORTEXT);
```

Testing Best Practices

We recommend the following best practices for designing your application tests:

- Test code should be kept separate from the code that is being tested. That way, you can release the tested code to a production environment without releasing the test code.
- Test code should test that the code works as intended both under successful and failing conditions. These tests are called positive and negative tests. The positive tests validate that the code being tested works as intended under successful conditions. The negative tests validate that the code being tested work as intended under failing conditions.
    1. In positive tests, the test method should validate the results of application calls, such as return values, state changes, or database transactions.
    2. In negative tests, the test method should validate that the intended errors occur, error messages are presented, and data has the expected values.
- Automated tests shouldn't require user intervention.
- Tests should leave the system in the same well-known state as when the test started. This way, you can rerun the test or run other tests in any order and always start from the same state.
- Test execution and reporting should be fast and able to integrate with the test management system. This way, the tests can be used as check-in tests or other build verification tests. These other tests typically run on unattended servers.
- Create test methods that follow the same pattern:
    1. Initialize and set up the conditions for the test.
    2. Invoke the business logic that you want to test.
    3. Validate that the business logic worked as expected.
- Only use hardcoded values in tests when you really need it. For all other data, consider using random data.

    For example, you want to test the `Ext. Doc. No. Mandatory` field in the `Purchases & Payables Setup` table. To do this, you need to create and post typical purchase invoice. The typical purchase invoice line specifies

an amount. For most tests, it doesn't matter exactly what amount. For inspiration, see the use of the **GenerateRandomCode** method in the tests that are included in the **TestToolkit** folder on the Business Central product media. For more information, see Random Test Data.

> **TIP**
> Use the Any module in the Microsoft/ALAppExtensions GitHub repo to generate pseudo-random values during test set-up. This module generates the same set of numbers, allowing you to reproduce test failures.

- Tests should be readable and fast to execute. We recommend that test codeunits run under 2 minutes, and that you don't add more than 100 test methods to the codeunit.

See Also

Testing Pages

Creating Handler Methods
Test Codeunits and Test Methods
Application Testing Example: Testing Purchase Invoice Discounts
Random Test Data
Testing the Advanced Extension Sample

Creating Test Codeunits and Test Methods

2/17/2021 • 2 minutes to read • Edit Online

In Dynamics 365 Business Central, you can create test codeunits and create test methods in the test codeunits.

Test codeunits are codeunits that have the SubType Property set to **Test**. You write tests as Al code in the methods inside of the test codeunits. There are three types of methods that you can add in a test codeunit: test, handler, and normal. Each method type is used for a specific purpose and behaves differently. When a test codeunit runs, it runs the **OnRun** trigger, and then runs each test method in the codeunit.

By default, each test method runs in a separate database transaction, but you can use the TransactionModel Property on test methods and the TestIsolation Property on test runner codeunits to control the transactional behavior.

The results of a test codeunit and of the individual test methods are displayed in a message window, but you can use the OnAfterTestRun Trigger on a test runner codeunit to capture the results. The outcome of a test method is either SUCCESS or FAILURE. If any error is raised by either the code that is being tested or the test code, then the global outcome of the test codeunit is FAILURE and the error is included in the results log file.

The difference between a normal codeunit and a test codeunit is their execution at runtime. When a normal codeunit is run, if one of its methods fails, then the codeunit is terminated. When a test codeunit is run, even if the outcome of one test method is FAILURE, the next test methods are still run.

The methods in a test codeunit can be one of the following types:

TYPE	DESCRIPTION	
Test method	You use test methods that include AL code that tests the business logic in the application, where each method covers a transaction. You declare the Test Attribute on the method.	
Handler method	You use handler methods to automate tests by handling instances when user interaction is required by the code that is being tested by the test method. In these instances, the handler method is run instead of the requested user interface. The handler method should simulate the user interaction for the test case, such as validating messages, making selections, or entering values. You declare a handler type attribute on the method. For more information, see Creating Handler Methods	
Normal method	You use normal methods to structure the test code by using the same design practices and principles as methods in other codeunits of the application. You declare the Normal Attribute on the method.	

See Also

Testing the Application

Creating Test Runner Codeunits

2/17/2021 • 2 minutes to read • Edit Online

You can create test runner codeunits to manage the execution of test codeunits and to integrate with test management or test reporting frameworks. By integrating with a test management framework, you can automate your tests and enable them to run unattended.

To create a test runner codeunit, create a codeunit and set the SubType Property to **TestRunner**.

To specify what changes in the database you want to roll back after the tests in the test runner codeunit execute, set the TestIsolation Property.

Test runner codeunits include the following triggers:

- OnRun Trigger
- OnBeforeTestRun Trigger
- OnAfterTestRun Trigger

In the **OnRun** trigger you enter the code to run the codeunits. It runs when you execute the codeunit and before the test methods run. You can use the **OnBeforeTestRun** and the **OnAfterTestRun** triggers to perform preprocessing and postprocessing, such as initialization or logging test results. If you implement the **OnBeforeTestRun** trigger, then it executes before each test method executes. If you implement the **OnAfterTestRun** trigger, then it executes after each test method executes and also suppresses the automatic display of the results message.

> **WARNING**
> The **OnBeforeTestRun** and **OnAfterTestRun** triggers always run in their own transactions, regardless of the value of the TestIsolation Property, the value of the TransactionModel Property, or the outcome of a test method.

Example

This sample codeunit runs three test codeunits in the automated application test libraries.

```
codeunit 50101 TestRunnerCodeunit
{
    Subtype = TestRunner;

    trigger OnRun()
    begin
        Codeunit.RUN(Codeunit::"ERM Vendor Statistics");
        Codeunit.RUN(Codeunit::"ERM Sales Quotes");
        Codeunit.RUN(Codeunit::"ERM Dimension");

    end;
}
```

You may want to define your test suite in a table and then write code in the test runner codeunit to iterate through items in the table and run each test codeunit. In that case, you can make use of the following example.

```
codeunit 50102 TestRunnerCodeunit
{
    Subtype = TestRunner;

    trigger OnRun()
    var
        EnabledTestCodeunit: Record "CAL Test Enabled Codeunit";
        Object: Record "Object";
    begin
        if EnabledTestCodeunit.FINDSET then
            repeat
                if Object.GET(ObjectType::Codeunit, '', EnabledTestCodeunit."Test Codeunit ID") then
                    CODEUNIT.RUN(EnabledTestCodeunit."Test Codeunit ID");
            until EnabledTestCodeunit.NEXT = 0

    end;
}
```

See Also

Testing the Application

Testing Pages

2/17/2021 • 3 minutes to read • Edit Online

You use test page objects to simulate user interactions with the application. You can:

- View or change the value of a field on a test page.
- View the data on page parts.
- View or change the value of a field on a subpage.
- Filter the data on a test page.
- Perform any actions that are available on the page.
- Navigate to different records.

You can create and open a test page in the following ways:

- Declare a test page variable and then write AL code to open the test page by using one of the following methods:
  - OpenNew Method (TestPage)
  - OpenEdit Method (TestPage)
  - OpenView Method (TestPage)
- Create a **PageHandler** or **ModalPageHandler** method that has a test page parameter.
- Write AL code to trap a call to open a test page by using the Trap Method (TestPage).

> **NOTE**
> You must consider how you set the TransactionModel Property to simulate the scenario that you want to test and to return the database to its initial state after the test.

> **NOTE**
> Test methods and code on test pages run on the Dynamics 365 Business Central Server instance, even though they simulate client interactions.

For more information about the AL methods that you use on a test page, see TestPage Data Type.

Accessing Fields on Test Pages

You access the fields on a test page by using the dot notation. For example, if you have a test page variable named `CustomerCard` that represents the `Customer Card` page, then to access the `Name` field on the test page, you write `CustomerCard.Name` in your code.

These fields are instances of the TestField Data Type, so you can use the corresponding AL methods to work with them. For example, if you have a test page variable named `CustomerCard` that represents the `Customer Card` page, then to assign the value of the `No.` field to a variable named `CustNo`, you write `CustNo := CustomerCard."No.".Value` in your code. To write a value in the `Address` field of a `Customer Card` page,

you write `CustomerCard.Address.Value := '<address>'` in your code.

Accessing Page Parts and Subpages

You access page parts and subpages on a test page by using the dot notation.

These are instances of the TestPart Data Type, so you can use the corresponding AL methods to work with them.

For example, to compare the value of the `No.` field on a page to the value of the `No` . field on a FactBox on the page, you can write the following code.

```
if CustomerCard."No.".Value <> CustomerCard."Sales Hist. Sell-to FactBox"."No.".Value then
  error("Page part data is not updated.");
```

Filtering Data on Test Pages

To filter the data that can be accessed on a test page, you use AL methods correspoding to the TestFilter Data Type instances. For example, to filter the customers on the `Customer List` page based on a range of values in the `No.` field, you can write the following code.

```
CustomerList.Filter.SetFilter("No.", '20000..30000');
```

Invoking Actions on Test Pages

Any action that is available on a page is also available on the test page that mimics that page. You access page actions by using the dot notation and the Invoke Method.

These are instances of the TestAction Data Type, so you can use the corresponding AL methods to work with them. To access built-in actions, such as Yes, No, OK, or Cancel you can also call the Yes Method, No Method, Ok Method and Cancel Method respectively, directly in the test page.

Navigating Among Records

To simulate moving to different items on a list page or moving to different records on a card page, you use one of the following navigation methods:

- Next Method (TestPage)
- Previous Method (TestPage)
- First Method (TestPage)
- Last Method (TestPage)
- GoToRecord Method (TestPage))
- GoToKey Method (TestPage)
- FindFirstField Method (TestPage)
- FindNextField Method (TestPage)
- FindPreviousField Method (TestPage)

See Also

Creating Handler Methods

2/17/2021 • 3 minutes to read • Edit Online

You can create test codeunits, test methods, and test pages to test your application. We recommend that you create tests that can be automated. To create automated tests, you must write code to handle all UI interactions so that the tests do not require user interaction when they are running. To do this, you create special handler methods.

You can use the following handler methods:

METHOD TYPE	PURPOSE	SIGNATURE	
MessageHandler	Handles **Message** statements.	< *Function name*> (< *Message*> : Text[1024])	
ConfirmHandler	Handles **Confirm** statements.	< *Function name*> (< *Question*> : Text[1024]; var < *Reply*> : Boolean)	
StrMenuHandler	Handles **StrMenu** statements.	< *Function name*> (< *Options* : Text[1024]; var < *Choice*> : Integer; < *Instruction*> : Text[1024])	
PageHandler	Handles specific pages that are not run modally.	< *Function name*> (var < *Page*> : Page < *page id*>) < *Function name*> (var < *Page*> : TestPage < *testpage id*>)	
ModalPageHandler	Handles specific pages that are run modally.	< *Function name*> (var < *Page*> : Page < *page id*>; var < *Response*> : Action) < *Function name*> (var < *Page*> : Page < *testpage id*>)	
ReportHandler	Handles specific reports. If you create a **ReportHandler** method, then that method replaces all code for running the report, including the request page, and a **RequestPageHandler** is not called. Only create a **RequestPageHandler** if you are not using a ReportHandler.	< *Function name*> (var < *Report*> : Report < *report id*>)	

METHOD TYPE	PURPOSE	SIGNATURE	
FilterPageHandler	Handles a specific filter page. The FilterPageHandler tests the UI that is generated by a FilterPageBuilder Data Type.	< *Function name* > (var < *Record1* > : RecordRef)[, var < *Record2* > : RecordRef] [, ...]): Boolean	
RequestPageHandler	Handles the request page of a specific report.	< *Function name* > (var < *RequestPage* > : TestRequestPage)	
HyperlinkHandler	Handles specific hyperlinks.	< *Function name* > (< *Hyperlink* > : Text[1024])	
SendNotificationHandler	Handles **Send** statements.	< *Function name* > (< *TheNotification* >: Notification): Boolean	
RecallNotificationHandler	Handles **Recall**	< *Function name* > (< *TheNotification* >: Notification): Boolean	
SessionSettingsHandler	Handles **RequestSessionUpdate** statements.	< *Function name* > (var < *SessionSettings* >: SessionSettings): Boolean	

How to create a handler method

To create a handler method, you set one of the handler attributes on a method. You must use the method signature specified for the handler attribute that you are using, as illustrated in this code example.

```
[MessageHandler]
procedure MessageHandler(Message: Text[1024])
begin
    Assert.IsTrue(StrPos(Message, MSG_HAS_BEEN_CREATED) > 0, Message);
end;
```

The parameters of the methods that are being handled are passed as parameters to the handler methods. For example, when **Message** is called in a test method, the parameter of the **Message** method is passed as the parameter of the **MessageHandler** method. For page and report handlers, the page, report, or request page is passed as the parameter of the **PageHandler**, **ModalPageHandler**, **ReportHandler**, or **RequestPageHandler**.

You can call handler methods from methods that have the Test Attribute and then specify the handler methods that it will use in the HandlerFunctions Attribute. The code inside the test method should simulate that the UI was actually raised and some values entered or some actions were taken. You can specify more than one handler method by separating the handler method names with a comma.

> **NOTE**
> Every handler method that you enter in the HandlerFunctions Attribute of a test method must be called at least one time in the test method. If you run a test method that has a handler method listed that is not called, then the test fails.

The following example shows a test method that uses the HandlerFunctions Attribute to call the

MessageHandler method.

```
[Test]
[HandlerFunctions('MessageHandler')]
procedure ApproveRequestForPurchCreditMemo()
var
    PurchHeader: Record "Purchase Header";
begin
    ApproveRequestForPurchDocument(PurchHeader."Document Type"::"Credit Memo");
end;
```

See Also

Testing the Application
AL Methods

Application Testing Example: Testing Purchase Invoice Discounts

2/17/2021 • 4 minutes to read • Edit Online

Before you release a customized Dynamics 365 Business Central application to a production environment, you must test the application. This walkthrough demonstrates how to use the test codeunits and test libraries to test an application.

About this example

You have modified codeunit 70, Purch-Calc.Discount, which is a codeunit in the CRONUS International Ltd. database. You want to test the functionality of the customized codeunit before you offer the customized application for sale. You create a new test codeunit with new test methods to test the Purch-Calc.Discount codeunit. During development, You use the application test libraries to help write a test with fewer lines of code.

Prerequisites

To complete this example, you will need:

- Dynamics 365 Business Central with a developer license.
- The CRONUS International Ltd. demo data company.
- To import the Test Toolkit.

Task 1: Create the test codeunit and method

1. Create a new codeunit and specifies that it is a test codeunit.
2. Define the scenario that you want to verify, and add a test method to test the Purch-Calc.Discount functionality.

   > **TIP**
   > By default, methods in test codeunits are test methods unless you specify otherwise.

Guidelines

- In this example, the name of the test method consists of the tested functionality, Purchase Invoice Discount Calculation, and relevant parameters that affect the test result. We recommend that you follow this naming pattern for your test methods also. In our example, the following parameters are introduced:
- `PInv` for the document that is tested, purchase invoices. You can apply the same test to purchase orders or purchase credit memos. Also, we recommend that you have mirrored sets of tests for the sales area.
- `Above`. It is a good practice to have tests for positive and negative scenarios. In this test, Isaac wants to check that discount come into effect when the document amount is above a minimum amount. But the amount in the document can be also less than or equal to the minimum amount.
- Isaac first defines the test scenario [SCENARIO], then details it with the GIVEN-THEN-WHEN notation. Finally, he adds the AL code.The code in this test method prepares the test data by setting a random discount percent, a minimum amount, and a document amount. Then, it creates a purchase document

with a line and runs the Purch-Calc.Discount codeunit, which contains the code that is being tested. Finally, it verifies the results of running the Purch-Calc.Discount codeunit and raises an error if the results are not as expected.

- You can create additional test methods in this test codeunit to test other aspects of vendor discounts. These test methods should include negative tests, which validate that the code being tested works as intended under failing conditions.

Task 2: Create a helper method

The next task is to create a helper method that generates data for the test. The helper method can be reused if you decide to extend test coverage.

Guidelines

- The code in the helper method prepares data for the test by creating a new vendor, setting up the invoice discount, and creating a purchase document with an item. Because this helper method is not specific to the test itself, you can reuse it for similar tests. For example, you can call it with other parameters and create a purchase credit memo, or set up 0% discount, or create a document where the total amount is less than the minimum amount that is specified in **Vendor Invoice Disc.** table.

> **NOTE**
> This test code does not guarantee that the state of the database after you run the test is the same as the state of the database before you run the test.

Code

```
codeunit 50111 "ERM Vendor Discount"
{
    // Specifies the codeunit to be a test codeunit
    Subtype = Test;

    trigger OnRun()
    begin

    end;

    // Makes the method a test method
    [Test]

    // Adds the test logic to the test method
    procedure PurchInvDiscCalculationPInvAbove()

    var
        PurchLine: Record "Purchase Line";
        MinAmount: Decimal;
        DocAmount: Decimal;
        DiscountPct: Decimal;
        PurchCalcDisc: Codeunit "Purch.-Calc.Discount";

    begin
        // [SCENARIO] "Inv. Discount Amount" should be calculated on Purchase Invoice (in LCY), where
Invoice amount is
        // above the minimal amount required for invoice discount calculation.
        // [GIVEN] Vendor with invoice discount percentage "D" for minimal amount "A" in LCY
        // [GIVEN] Create purchase invoice with one line and amount >"A"
        DiscountPct := RandomNumberGenerator.RandDec(100, 5);
        MinAmount := RandomNumberGenerator.RandDec(1000, 2);
        DocAmount := MinAmount + RandomNumberGenerator.RandDec(100, 2);
        CreatePurchDocument(PurchLine, PurchLine."Document Type"::Invoice, DocAmount, MinAmount,
```

```
DiscountPct);
        // [WHEN] Calculate invoice discount for purchase document (line)
        PurchCalcDisc.RUN(PurchLine);
        // [THEN] "Inv. Discount Amount" = Amount "A" * discount "D" / 100
        PurchLine.Find;
        Assert.AreEqual(Round(PurchLine."Line Amount" * DiscountPct / 100), PurchLine."Inv. Discount
Amount", PurchInvDiscErr);
    end;

    // Creates the test helper method
    local procedure CreatePurchDocument(var PurchLine: Record "Purchase Line"; DocumentType: Option;
DocAmount: Decimal; MinAmount: Decimal; DiscountPct: Decimal)

    var
        VendorInvoiceDisc: Record "Vendor Invoice Disc.";
        PurchaseHeader: Record "Purchase Header";
        VendorNo: Code[30];

    begin
        // Create vendor
        VendorNo := LibraryPurchase.CreateVendorNo;
        // Create vendor invoice discount
        VendorInvoiceDisc.Init;
        VendorInvoiceDisc.Code := VendorNo;
        VendorInvoiceDisc.Validate("Currency Code", '');
        VendorInvoiceDisc.Validate("Minimum Amount", MinAmount);
        VendorInvoiceDisc.Validate("Discount %", DiscountPct);
        VendorInvoiceDisc.Insert(TRUE);
        // Create purchase line
        LibraryPurchase.CreatePurchaseDocumentWithItem(PurchaseHeader, Purchline, DocumentType, VendorNo,
'', 1, '', 0D);
        PurchLine.Validate("Direct Unit Cost", DocAmount);
        PurchLine.Modify(TRUE);
    end;

    var
        RandomNumberGenerator: Codeunit "Library - Random";
        LibraryPurchase: Codeunit "Library - Purchase";
        Assert: Codeunit Assert;
        myInt: Integer;
        PurchInvDiscErr: Label 'The Purchase Invoice Discount Amount was not calculated correctly.';

}
```

See Also

Testing the Application

The Performance Toolkit Extension

2/17/2021 • 10 minutes to read • Edit Online

This extension is built for independent solution vendors (ISVs) and value added resellers (VARs) who develop vertical solutions and customize Business Central for their customers. In this type of collaboration, things often change between released versions on both sides, so it's important that ISVs and VARs can ensure that new versions of their solutions don't introduce performance regressions as the volume of users grows. To help, the Performance Toolkit lets developers simulate workloads in realistic scenarios to compare performance between builds of their solutions.

In short, the Performance Toolkit helps answer questions such as, "Does my solution for Business Central support X number of users doing this, that, and the other thing at the same time?" It doesn't answer questions such as, "How many orders can Business Central process per hour?"

> **IMPORTANT**
> You can use the toolkit only in sandbox environments and Docker images. You cannot use it in a production tenant.

Components of the Performance Toolkit

The Performance Toolkit is two extensions, the **Performance Toolkit**, which is available for free on AppSource, and **BCPT-SampleTests**, which you can download from the ALAppExtensions repository on GitHub. To get the full benefit of the toolkit, we recommend that you download and install both extensions. Combined, the extensions provide the following:

- A framework for defining a set of tests or scenarios to run in parallel. The framework also logs results and lets you import and export suite definitions.
- Predefined test suites that cover basic scenarios, which can also serve as inspiration for other suites that suit your customer environments.
- A command line tool that must be installed on a client computer. To simulate multiple users signing in and using pages, you must start those scenarios from outside Business Central. The command line tool will run the number of concurrent client sessions that is specified in the suit. For more information, see Starting the Run from PowerShell.

Single and Multiple Sessions

Typically, you'll want to run the suite for multiple sessions at the same time. After you configure the suite, you can do that by using the **Start** action. However, if you want to do light-weight testing, for example, early in the development phase, you can choose the **Start in Single Run mode** action to run your suite just once, and as fast as possible. Single Run mode lets you monitor the number of SQL statements between runs and define baselines, and gives you quick feedback that can help identify regressions early on.

You can run up to 125 sessions at the same time for a test suite. The **Total No. of Sessions** field shows how many sessions will be created when you run the suite.

Running in the Background and Foreground

On the suite lines, the **Run in Foreground** check box lets you run tests in sequence, rather than in parallel. You can also use it in combination with background tasks, but that will run each session individually. You can only run one session in the foreground at a time.

> **NOTE**
> If you use Single Run mode and run in the foreground, you don't need to go to PowerShell to run the test.

Configuring a Suite

The settings to configure a suite depend on the environment that you want to simulate. The following procedure provides an example for testing multiple sessions, but the steps also apply to a single run.

To configure a suite

1. Search for **BCPT Suites**, and then choose the related link.
2. Choose **New** to open the BCPT Suite page.
3. In the **Code**, **Description**, and **Tag** fields, provide an identifier, some information about the test, and a tag that you can use to find the results of the suite on the Log Entries page.
4. Define timing for the run.
   a. The **1 Work Day Corresponds to** field works with the **Duration (minutes)** field to update the **Work Date Starts at** field, and you use it to test processes that have deadlines, such as payments. The duration can be up to 240 minutes.
   b. The **Default Min. User Delay** and **Default Max. User Delay** fields in the header let you simulate pauses between actions. You must specify a delay. Between each iteration, for example when creating sales orders, you can define a **Delay** (Delay between iterations) before the test starts on the next sales order. The delay can be **Fixed** or **Random**. Delays are not included the results for run times.
5. Specify the base version to compare.

   > **TIP**
   > To change the value in the **Base Version** field, you might need to turn on Edit mode.

6. Configure lines for the suite.

   > **TIP**
   > The lines will contain some of the settings from the header. Updating the values on the lines will also update the header.

   a. On the **BCPT Suite Lines** FastTab, choose the codeunits to run.
   b. In the **Parameters** field, enter a parameter to define iterations such as, for example, creating lines on documents. For example, a parameter **Lines=10** will create 10 lines on a document.
   c. In the **No. of Sessions** field, enter the number of concurrent users to simulate.
   d. Optional: If you want to run in Single Run mode, or you want to run one of the sessions without applying settings such as minimum and maximum delays, choose the **Run in Foreground** check box. For more information, see Running in the Background and Foreground.

Starting the Run from PowerShell

After you have installed the binaries and scripts and configured your suite, you can create the credential object and run the tests from PowerShell by using the following commands

To create the credential object, run the following command:

```
$Credential = New-Object PSCredential -ArgumentList <user email>,(ConvertTo-SecureString -String <password>
-AsPlainText -Force)
```

To start tests in a Business Central online sandbox, run the following command:

```
RunBCPTTests.ps1 -Environment PROD -AuthorizationType AAD -Credential $Credential -SandboxName <sandbox
name> -TestRunnerPage 149002 -SuiteCode "TRADE-50U"
```

> **NOTE**
>
> When you start tests from PowerShell, there is a two second delay between new sessions.

Analyzing Results

When a run has completed, you can view the results on the lines on the **BCPT Suite Lines** FastTab. For more information, see Analyzing the Results.

Troubleshooting log entries

After running the suite, you can choose **Log Entries** to see what happened. Use the **Show Errors** and **Show sessions running at the same time as this** actions to apply filters to the results. For example, filtering can help troubleshoot errors by showing what a user was doing when an error occurred. By default, the page is filtered to show the latest version, but you can change or remove the filter if you want to compare runs. You can use the **Open in Excel** action to build dashboards that can help visualize performance results.

Log entries are listed in the order that they were created, which means that the different scenarios are mixed. Each run is identified by the value in the **Version No.** fields.

The **Operation** column shows the individual measurements, where the term *Scenario* is used for running the codeunit without the user wait time. The **No. of SQL Statements** column includes the SQL statements that were issued by the scenario and system activities, such as retrieving metadata. The counts exclude the log entries themselves. To drill down to a single session, filter on the **Session No.** field or choose **Open in Excel** to create a pivot table and pivot chart for deeper analysis.

Example: Evaluate SQL calls and timing in Single Run mode

This example shows how to use Single Run mode for performance regression testing (PRT) between changes to code, to evaluate SQL calls and timing. Often, when developing a new extension, you start out with limited code and may want to wait to do a larger benchmark test with simulated concurrent users until you're closer to having a full, end-to-end scenario. You can use the **Start in Single Run Mode** action to perform a limited test, for example, on a new extension. Single Run mode will still provide things like a baseline, the ability to run the test in the background, and give you instant feedback.

The data that the runs generate is persisted in the database. If the database is maintained, you can set previous runs as baseline.

To run a test in Single Run mode

The following steps provide an example of how to run a PRT in Single Run mode.

1. On the **BCPT Suites** page, choose **New**.

2. In the **Code** field, enter a name for the test suite. In this example, we'll use **PreTest**.
3. In the **Tag** field, enter something that will make it easy to identify the suite later when you analyze the results.
4. The fields for duration and delays are not used for Single Run mode, so we'll leave their default values.
5. On the BCPT Suit Lines FastTab, choose the test suite for your component.
6. In this example, we're calculating the total weight of the items on a sales order, so we'll use the **Sales Order** page test with a parameter that creates four lines. The parameter looks as follows: Lines=4.
7. Clear the **Run in Foreground** check box. This will run the test suite in a background thread.
8. Choose **Start in Single Run mode**.
9. To set your baseline after the run completes, in the **Base Version** field, enter **1**.

> **TIP**
> If you run the test again you'll have delta values in the test case line. For more information, see Analyzing the Results. You can reset the baseline to any version, and run as many test as needed.

Analyzing the results

The results of the PRT are shown on the **BCPT Suite Lines** FastTab. The following tables describe the fields that show results of the current run, the results of the baseline, and the delta between the two.

FIELDS FOR CURRENT RUN	DESCRIPTION
No. of Iterations	How many times it ran. this will remain 1 as we use the Start In single Run Mode.
Total Duration in (ms)	How long it took to complete one iteration.
Average duration	Equal to **Total Duration in (ms)** field for the first run, but after you run test multiple times it's a result of the sum of runs since your baseline.
No. of SQL statements	The number of SQL statements generated for one run.
Avg. No. of SQL Statements	Similar to the duration fields, for the first run this will be the same as the **No. of SQL statements** field, but will become averaged after subsequent runs.

The following table describes the fields related to the baseline.

FIELDS FOR BASELINE	DESCRIPTION
No. of Iterations Base	Not relevant for PRTs in Single Run mode.
Total Duration Base (ms)	The total duration of the baseline run.
Avg. Duration Base (ms)	Not relevant for PRTs in Single Run mode.
Avg. No. of SQL Statements Base	The number of SQL statements in your base run.

The following table describes the fields that show the delta between the run and the baseline.

FIELDS FOR THE DELTA	DESCRIPTION
Change in No. of SQL statements (%)	The difference, as a percent, between a baseline and the latest run.
Changes in Duration (%)	The change in measured time between a baseline and the latest run.

> **NOTE**
> The first iteration of any scenario running will show higher number of SQL Statements because nothing has been cached.

Writing Test Cases (codeunits)

A test case is a codeunit of either a **Normal** or **Test** subtype. If the subtype is Normal, the test scenario should be defined in the OnRun trigger because the Performance Toolkit uses the codeunit to run the testcase.

To interact with pages and make the tests more realistic, define a codeunit of the subtype **Test**, as shown in the example for codeunit BCPT Open Item List. Codeunit BCPT Test Context is an interface for running tests. Tests can use the StartScenario amd EndScenario functions on the BCPT Test Context codeunit to log when a scenario that is being measured started and stopped. To simulate user delays, the UserWait() function should be called while moving between fields to make the tests more realistic. The BCPT Test Context codeunit also exposes the parameters that are set on the test codeunit to the test suite. When using Business Central, an implicit Commit() is called for every interaction, and that should be simulated in the tests by calling an explicit Commit().

See Also

Testing the Application Overview
Development in AL
Best Practices for AL

Rules and Guidelines for AL Code

2/17/2021 • 4 minutes to read • Edit Online

This page defines the rules and guidelines to follow when writing AL code in an extension package for Dynamics 365 Business Central. The rules and guidelines are grouped according to two importance levels: critical errors that must be resolved, and important errors that should be resolved. Errors that are not resolved must include an explanation and justification for the error.

Critical errors

- Code uses encryption key functions such as IMPORTENCRYPTIONKEY, EXPORTENCRYPTIONKEY, CREATEENCRYPTIONKEY, and DELETEENCRYPTIONKEY. (It is fine to use the ENCRYPT and DECRYPT methods.)
- Code uses ASSERTERROR.
- External data connections do not properly handle sensitive data.
- It does not encrypt sensitive table data. (i.e. credit card info, passwords, etc.).

Important errors

- Temporary files are not cleaned up after use.
- Code uses codeunits that require printers to be selected.
- Code uses a specific time zone or locale.

Common pitfalls

To help you save time, we're sharing a list of the top 15 common pitfalls that regularly lead to app validation failures.

1. Prefix/Suffix missing

   One of the app requirements is for you to reserve a prefix/suffix for your app. This is needed to ensure a healthy app ecosystem by avoiding collision amongst apps. This common failure occurs due to not setting your prefix/suffix in some or all required places. For more information, see Benefits and Guidelines for using a Prefix or Suffix.

2. DataClassification missing or set incorrectly

   Due to GDPR requirements, fields of field class *Normal* must use the DataClassification property, and its value must be different from *ToBeClassified*. This applies to fields in tables and table extensions. Use the AppSourceCop tool for detecting this.

3. Required translation files missing

   There are many countries today that where Dynamics 365 Business Central is available, and that you can support as well with your app. For specifying additional languages, we no longer support Caption ML. You must use xliff translation files instead. For more information, see Working with Translation Files. Microsoft provides a free translation tool that you can access from https://lcs.dynamics.com. To support a specific country, you must include a translation file per for each language code. For example, to support Switzerland, you must provide fr-CH, de-CH, and it-CH.

4. Missing permission sets

Your app must provide one or more permission sets so that users can use your app's functionality. Your app must never require the SUPER permission set.

5. Permission errors

   For your app to be a good citizen in Dynamics 365 Business Central, permission errors must not appear unless it is a necessary reason for showing the error.
   It is acceptable to throw an error to a user that does not have your permission set marked and tries to access your page object. It is not acceptable to throw an error to that same user trying to access Business Central pages in the base application, or to throw an error to a user who is not trying to access your app's functionality.

6. Missing application area tagging

   Tag in which part your app participates. Pages, controls, actions, and fields will not appear in Dynamics 365 Business Central if the Application Area property has not been set.

7. Usage Category not set

   You enable a page or report to be available through search in Dynamics 365 Business Central by using the UsageCategory property. For more information, see Using Tell Me to Find Features and Information.

8. OnCompany procedure

   Due to their performance impact, *OnBeforeCompanyOpen* and *OnAfterCompanyOpen* cannot be used. For more information, see Replacing OnBeforeCompanyOpen and OnAfterCompanyOpen.

9. Upgrade procedures

   Make sure that your app can be upgraded properly. For more information, see Upgrading extensions.

10. Profiles

    Do not insert into the Profile table. Use the Profile object instead.

11. App file not properly code signed

    Your app file must be digitally signed with a certificate from a third-party certification authority trusted by Windows.

12. You tested your app on an obsolete Dynamics 365 Business Central version (or never even tested it)

    Make sure that your app is properly tested on the correct version. For more information, see Current Build - Developing for Dynamics 365 Business Central on the Collaborate site.

13. You tested using SUPER permissions

    You tested your app, but your user had SUPER permissions. This can hide critical errors. You must test with a user that doesn't have the SUPER permissions. The user must have the ESSENTIAL license. For more information, see Testing your Extension.

14. User scenario document unclear

    Our validation team is testing your app functionality manually, so we need to be able to understand the core functionality of your app. If your user scenario document is missing important details that are needed for us to properly walk through your app's setup and usage scenarios, we cannot validate your app successfully. For more information, see User Scenario Documentation.

15. The .json file is incorrect

    There are many values in the app.json file that may not be mandatory to compile your app, but are mandatory for your app to be in AppSource. For example, your app cannot be published to a production

tenant if the **target** value is set to *OnPrem*. It must be set to *Cloud*. For information, see JSON Files.

See Also

Best Practices for AL Code
Checklist for Submitting Your App

Best Practices for AL

2/17/2021 • 4 minutes to read • Edit Online

This page defines some of the best practices to follow when writing AL code for Dynamics 365 Business Central. These best practices are additional to rules and guidelines that are caught during compilation of AL code. We recommend following these best practices when developing extensions in AL to ensure consistency and discoverability on file, object, and method naming, as well as better readability of written code.

> **NOTE**
> If a best practice is not mentioned here, the PreCal rules listed here apply.

Extension structure

An extension is fully contained in a single folder. This folder often contains multiple files, such as `app.json` and `launch.json` files, perhaps an image file representing the extension's logo, various folders for source; "\src", other resources; "\res", and a test folder; "\test" folder. The extension does not need to follow a flat structure, which means that, depending on the amount of application files, additional folders can be used in the "src" or "test" folders to group objects based on their functionality, which can help make maintaining a large .al project easier.

File naming

Each file name has object names with only characters [A-Za-z0-9], object type, and dot al, for file type. In your extension, the name of each new application object (table, page, codeunit), can contain a prefix or suffix.

The CodeCop analyzer suggests that the object name is part of the file name, which is encouraged as a best practice. Adding any affixes to the file names is voluntary.

> **NOTE**
> If you are submitting an app to AppSource, you must follow the guidance in the Technical Validation Checklist.

File naming notation

Follow the syntax for file naming as shown below:

FULL OBJECTS	EXTENSIONS
`<ObjectNameSuffix>.<FullTypeName>.al`	`<ObjectNameSuffix>.<FullTypeName>Ext.al`
`<PrefixObjectName>.<FullTypeName>.al`	`<PrefixObjectName>.<FullTypeName>Ext.al`
`<ObjectNameExcludingAffix>.<FullTypeName>.al`	`<ObjectNameExcludingAffix>.<FullTypeName>Ext.al`

Type map

Use the listed abbreviations for each type of object in the file naming:

OBJECT	ABBREVIATION
Page	Page
Page Extension	PageExt
Page Customization	PageCust
Codeunit	Codeunit
Table	Table
Table Extension	TableExt
XML Port	Xmlport
Report	Report
Request Page	RequestPage
Query	Query
Enum	Enum
Enum Extension	EnumExt
Control Add-ins	ControlAddin
Dotnet	Dotnet
Profile	Profile
Interface	Interface

Examples of object naming

Table

```
table 70000000 MyPrefixSalesperson
```

Page

```
page 70000000 MyPrefixSalesperson
```

Page extension

```
actions
{
    addafter(ApprovalEntries)
    {
        action(MyPrefixVacation)
```

Codeunit

```
codeunit 70000000 MyPrefixSalesperson
```

File naming examples

For the listed objects above, these are examples of the file naming.

OBJECT NAME	FILE NAME
codeunit 70000000 MyPrefixSalesperson	`MyPrefixSalesperson.Codeunit.al`
page 70000000 MyPrefixSalesperson	`MyPrefixSalesperson.Page.al`
page 70000000 MyPrefixSalesperson extends "Customer Card"	`MyPrefixSalesperson.PageExt.al`

Formatting

We recommend keeping your AL code properly formatted as follows:

- Use all lowercase letters for reserved language keywords. Built-in methods and types are not included in this rule because they are written using Pascal case.
- Use four spaces for indentation.
- Curly brackets are always on a new line. If there is one property, put it on a single line.

The following example illustrates these formatting rules.

```
page 123 PageName
{
    actions
    {
        area(Processing)
        {
            action(ActionName)
            {
                trigger OnAction()
                begin
                end;
            }
        }
    }

    var
        TempCustomer: Record Customer temporary;

    [EventSubscriber(ObjectType::Page, Page::"Item Card", 'OnAfterGetCurrRecordEvent', '', false, false)]
    local procedure OnOpenItemCard(var rec: Record Item)
    var
        OnRecord: Option " ", Item, Contact;
    begin
        EnablePictureAnalyzerNotification(rec."No.", OnRecord::Item);
    end;
}
```

The AL Language extension offers users the option to automatically format their source code. For more information on how to use it, see AL Formatter.

Line length

In general, there is no restriction on line length, but lengthy lines can make the code unreadable. We recommend that you keep your code easily scannable and readable.

Object naming

Object names are prefixed. They start with the feature/group name, followed by the logical name as in these two examples:

- `Intrastat extension validation codeunit for Denmark`
- `codeunit 123 "IntrastatDK Validation"`

> **NOTE**
> The "MS - " prefix is not required.

File structure

Inside an .al code file, the structure for all objects must follow the sequence below:

1. Properties
2. Object-specific constructs such as
   - Table fields
   - Page layout
   - Actions
   - Triggers
3. Global variables
   - Labels
   - Global variables
4. Methods

Referencing

In AL, objects are referenced by their object name, not by their ID.

Example

```
Page.RunModal(Page::"Customer Card", ...)

var
    Customer: Record Customer;
```

Variable and field naming

For variables they must:

- Be named using PascalCase
- Have the `Temp` prefix if they are temporary variables
- Include the object name in the name (for objects)

Furthermore:

- Field and variable names should not include wildcard symbols, such as `%` and `&`. This might break features such as export using Excel or RapidStart.

- Name fields using aA-zZ and 0-9 and use Caption and xliff files to display the field appropriately. For more information, see Working with Translation Files.
- Using English as the language for naming improves the ability to troubleshoot issues that may arise.

Example

```
TempCustomer: Record Customer temporary;
Vendor: Record Vendor;
```

Method declaration

To declare a method, follow the guidelines below:

- Include a space after a semicolon when declaring multiple arguments.
- Semicolons can be used at the end of the signature/method header. If you use a snippet, the semicolons are not automatically added.
- Methods are named as variables using Pascal case. However, this is not a mandatory rule.
- There must be a blank line between method declarations. If you format your code using the AL Formatter tool, the auto-formatter sets the blank line between procedures.

Example

```
local procedure MyProcedure(Customer: Record Customer; Int: Integer)
begin
end;

// space

local procedure MyProcedure2(Customer: Record Customer; Int: Integer)
begin
end;
```

Calling methods

When calling a method, include one space after each command if you are passing multiple parameters. Parentheses must be specified when you are making a method call or system call such as: `Init()`, `Modify()`, `Insert()` etc.

Example

```
MyProcedure();
MyProcedure(1);
MyProcedure(1, 2);
```

Type definition (colon)

When declaring a variable or a parameter, the name of that variable or parameter must be immediately followed by a colon, then a single space, and then the type of the variable/parameter as illustrated in the example below.

```
var
    Number: Integer;

local procedure MyProcedure(a: Integer; b: Integer): Integer
```

See Also

Checklist for Submitting Your App
Rules and Guidelines for AL Code
Using the Code Analysis Tool

Best Practices for Deprecation of Code in the Base App

2/17/2021 • 2 minutes to read • Edit Online

This topic provides guidelines that describe how code in the Base App is obsoleted. The topic describes some best practices that Microsoft is using for obsoleting code, and is meant as a non-enforced guidance and best practice. You can use this topic as an inspiration on how to set up a best practice for your own code. For obsoleting code, preprocessor statements in AL can be used. For more information, see Directives in AL.

Obsoleting Code

When we obsolete code, we:

- Add the preprocessor statements `#if`, `#else`, and `#endif` surrounding the code to be obsoleted.
- Use one of the following preprocessor symbols, where the pattern is `CLEAN<Version>`, such as `CLEAN15`, `CLEAN16`, `CLEAN17`, and `CLEAN18`.

  > **NOTE**
  > These symbols are not shipped with the product.

- The version to use in the symbol matches the `<major>` of the `ObsoleteTag`. For example:
  - If a method is to be removed, then we are using `#if not`

```
#if not CLEAN18
    [Obsolete('Replaced by SetParameters().', '18.0')]
    procedure SetParams(NewAnalysisArea: Option Sales,Purchase,Inventory; NewReportName:
Code[10]; NewLineTemplateName: Code[10]; NewColumnTemplateName: Code[10])
    begin
        SetParameters("Analysis Area Type".FromInteger(NewAnalysisArea), NewReportName,
NewLineTemplateName, NewColumnTemplateName);
    end;
#endif
```

  - If an action is to be removed, then we are also using `#if not`

```
#if not CLEAN17
    action("Social Listening Setup")
    {
        ApplicationArea = All;
        Caption = 'Social Engagement Setup';
        RunObject = page "Social Listening Setup";
        Visible = false;
        ObsoleteState = Pending;
        ObsoleteReason = 'Microsoft Social Engagement has been discontinued.';
        ObsoleteTag = '17.0';
    }
#endif
```

  - If a table is to be removed, then we'll use `#if #else #endif`

```
table 1808 "Aggregated Assisted Setup"
{
    Access = Internal;
    Caption = 'Aggregated Assisted Setup';
#if CLEAN16
    ObsoleteState = Removed;
#else
    ObsoleteState = Pending;
#endif
    ObsoleteReason = 'Data available in Assisted Setup already- extensions also register in
the same table.';
    ObsoleteTag = '16.0';
```

- If a table is to be marked as `Temporary`, then we'll use `#if #else #endif`

```
table 5503 "Acc. Schedule Line Entity"
{
    Caption = 'Acc. Schedule Line Entity';
    // TableType = Temporary;
#if CLEAN17
    TableType = Temporary;
#else
    ObsoleteState = Pending;
    ObsoleteReason = 'Table will be marked as TableType=Temporary. Make sure you are not using
this table to store records';
    ObsoleteTag = '17.0';
#endif
```

In order to have the compiler take the new 'clean' code path, the symbols must be defined. This is done in the `app.json` file with the following setting. For more information, see JSON Files.

```
"preprocessorSymbols": [ "CLEAN15", "CLEAN16", "CLEAN17", "CLEAN18" ]
```

> **IMPORTANT**
> A best practice is to change this locally to make sure everything compiles, run tests locally, and submit test jobs.

Fixing code when objects are removed

If an action or other code element points to a now removed object, then the guidance is to:

- Ensure that the action is obsoleted.
- Add preprocessor statements to fix the issue.
  - If code points to an obsoleted method, then use directives to put in the fixed code.
  - If code points to an obsoleted table/field, then use directives to put in the fixed code.

See Also

AL Development Environment
Directives in AL
ObsoleteTag Property
ObsoleteState Property
ObsoleteReason Property

Benefits and Guidelines for using a Prefix or Suffix

2/17/2021 • 3 minutes to read • Edit Online

In your extension, the name of each new application object (table, page, codeunit), must contain a prefix or suffix. This rule applies to all objects. You can use the Caption values for what you decide to display to the user. When you modify a core Dynamics 365 object using a table extension or a page extension, the prefix/suffix must be defined at the control/field/action/group level.

Examples

Declare your objects with a prefix as shown in the following examples.

Table

```
table 70000000 MyPrefixSalesperson
```

```
table 70000001 SalespersonMySuffix
```

Page

```
page 70000000 MyPrefixSalesperson
```

```
page 70000001 SalespersonMySuffix
```

Page extension

```
actions
{
    addafter(ApprovalEntries)
    {
        action(MyPrefixVacation)
```

```
actions
{
    addafter(ApprovalEntries)
    {
        action(VacationMySuffix)
```

Codeunit

```
codeunit 70000000 MyPrefixSalesperson
```

```
codeunit 70000001 SalespersonMySuffix
```

Benefits

Apps in AppSource are required to register and use a prefix or a suffix. Only in vary rare cases will apps run into naming conflicts.

Per-tenant extensions are not required to use a prefix or suffix, but we strongly recommend that you do so. You can use *pte* as prefix or suffix to avoid conflicts with AppSource apps or base objects.

> **NOTE**
> If your per-tenant extension causes a conflict with a new object in the base application or an updated AppSource app, then the per-tenant extension will be required to make the change.

General rules

- The prefix/suffix must be at least 3 characters
- The object/field name must start or end with the prefix/suffix
- If a conflict arises, the one who registered the prefix/suffix always wins
- For your own pages/tables/codeunits, you must set the prefix/suffix at the top object level
- For pages/tables in the base application or other apps that you extend, you must set the prefix/suffix at the top object level and also at the control/field/action level
- Use the AppSourceCop tool to find all missing prefixes and/or suffixes. Configuration options for this tool can be found here. The Rules section explains the different checks that the analyzer will do. For prefix/suffix detection, refer to the Configuration section. It explains how to set your prefix in the AppSourceCop.json file.

Examples – per-tenant extension

Let's say that you're creating a per-tenant extension, **myext** and you want to future-proof the naming by applying the prefix or suffix *pte*, which you are not required to register.

Let's look at some examples:

PREFIXES	SUFFIXES
pte-myext-salespersoncode	salespersoncode-myext-pte
pte_myext_salespersoncode	salespersoncode_myext_pte
pte myext salesperson code	salesperson code myext pte
pteMyExtSalesPersonCode	SalesPersonCodeMyExtPte

Examples - AppSource app

Alternatively, let's say your company is Fabrikam, and you're building an app called *Rentals*. First thing, you email d365val@microsoft.com and register *fab* as your company affix.

> **TIP**
> It is always a good idea to supply a few suggestions in priority order to avoid back and forth communication.

> **NOTE**
> For AppSource validation file names are not enforced; only object names.

A registered affix must be 3 letters, no more, no less, and you must provide the publisher name, which you will be using in app.json when you apply for an affix.

Once you get confirmation from Microsoft, you are free to use object and field names that start with *fab* or end with *fab*. Here are some examples:

PREFIXES	SUFFIXES
fab-salespersoncode	salespersoncode-fab
fab_salespersoncode	salespersoncode_fab
fab salespersoncode	salespersoncode fab
FabSalesPersonCode	SalesPersonCodefab

At Fabrikam, another team is building another app, so you request a special affix for your app so that the two Fabrikam apps can be kept apart. In this scenario, you do not have to register anything with Microsoft, as long as you do this with your company affix. Here are some examples:

PREFIXES	SUFFIXES
fab-rentals-salespersoncode	salespersoncode-rentals-fab
fab_rentals_salespersoncode	salespersoncode_rentals_fab
fab rentals salesperson code	salesperson code rentals fab
FabRentalsSalesPersonCode	SalesPersonCodeRentalsfab

In this scenario, your appSourceCop.json configuration will specify something like `fab-rentals` and `rentals-fab` as values for `mandatoryaffixes`, even though only *fab* was registered with Microsoft.

> **IMPORTANT**
> You are not required to change any already registered affixes; you can continue using these affixes. The guidelines above only apply to new registrations.

> **TIP**
> Contact us at d365val@microsoft.com to reserve the prefix/suffix of your choosing.

See Also

Checklist for Submitting Your App
Rules and Guidelines for AL Code

Instrumenting an Application for Telemetry

2/17/2021 • 2 minutes to read • Edit Online

This article describes how you can implement custom telemetry signals in your application for emitting telemetry data.. This data can then be collected and visualized for analyzing the application against the desired business goals, troubleshooting, and more.

Telemetry overview

One aspect of event logging is collecting data about how the application and your deployment infrastructure is working in order to diagnose conditions and troubleshoot problems that affect operation and performance. For example, this type of event logging includes Business Central Server events and trace events like SQL and AL method (function) traces.

Another aspect of logging is *telemetry*, which is collecting data about how your application functions and how it is being used in production. Telemetry can tell you about specific activities that users perform within the application in the production environment. Telemetry is also a useful tool for troubleshooting, especially instances where you are not able to reproduce the conditions experienced by the user or have no access to the user's environment. Telemetry can be divided into different levels or categories, like: telemetry for engineering, telemetry about the business, telemetry for customers.

Creating custom telemetry signal

There are two different resources where telemetry trace signals can be sent for monitoring and analyzing: Event Log and Microsoft Azure Application Insights. By default, the Business Central application is instrumented to emit several system telemetry trace signals to these destinations. Custom telemetry trace signals enable you to send telemetry data from anywhere in the application code to either of these destinations.

The procedure for creating custom telemetry signals is different for each resource. Your choice might also depend on whether you are developing for Business Central online or on-premises.

REOURCE	DESCRIPTION	ONLINE	ON-PREMISES	MORE INFORMATION

REOURCE	DESCRIPTION	ONLINE	ON-PREMISES	MORE INFORMATION
Application Insights	Create custom telemetry signals that are sent to an Application Insights resource in Azure. Application Insights is a service hosted within Azure that gathers telemetry data for analysis and presentation. Extension developers can specify whether the signal is only sent to the extension publisher or also to the VAR partner telemetry resource. You create these custom trace signals by using the LOGMESSAGE method in code.	✓	✓	See...
Event Log	Create custom telemetry trace signals that are sent to the Event Log of the Business Central Server machine. You create these custom trace signals by using the SENDTRACETAG method in code.		✓	See...

NOTE

Using Application Insights is recommended.

See Also

Monitoring and Analyzing Telemetry
Monitoring Business Central Server Events

Testing your Extension

2/17/2021 • 4 minutes to read • Edit Online

Several key things lead to your Dynamics 365 Business Central extension passing the Microsoft validation process. However, one of the most important checks you can do is to take the time and test your extension before submitting it for validation. This allows you to catch some of the basic errors that could lead to validation failures. The following list calls out key points, and the sections below provide more context.

- Always test in a Dynamics 365 Business Central online environment. If you test in an on-premises deployment, you might miss errors that would be seen online.
- Ensure that your extension can be published without code signing errors. You **must not** use the `-skipverification` flag.
- The extension should be able to be installed without errors.
- If you are using the **Assisted Setup**, ensure that you can use your wizard to completion without errors.
- Walk through the setup and usage of your extension to verify it works as expected (**remember to test as a user that does not have SUPER permissions**).
- Check that you can uninstall and unpublish your extension without any errors.
- Make sure you can republish and reinstall your extension without any errors.

Use the correct Dynamics 365 Business Central version

Use Docker for your development and testing. At least, run your full test in Dynamics 365 Business Central online at least once before submitting for validation. We use Docker, and this ensures that you will be testing on the same as what we validate your app on.

If you test in an on-premises deployment, you might miss errors that would be seen online.

Make sure that you use the correct artifacts for Docker to set up the correct Dynamics 365 Business Central version number. Use the function `Get-BCArtifactUrl` in the BcContainerHelper module to retrieve the artifactUrl of the current version. If you test on a build that is older or much newer than the current version, your app will most likely fail validation.

Also make sure that you refresh the Docker instance each time you want to submit. If you haven't run your Docker script to refresh for months, then you are on a much older build.

Use the correct data when you test your app

When we validate apps, we use the base CRONUS demo data. This of course is there for some countries and you don't have to do anything additional to receive that demo data. However, for countries that are empty and don't include demo data, you must import the CRONUS evaluation demo data. Not the International CRONUS data, the evaluation demo data. We do not use custom data and we do not use any other data. We always use the base evaluation demo data. To get this same data (if you don't have it by default), you follow these instructions:

1. Search for **Configuration Packages**, and then choose the link.
2. Choose **Process > Import Predefined Package**.
3. Click the link for **GB.ENU.EVALUATION**.

   That will start processing. You will see the process bar.
4. If there are popups at all, just choose **Yes** or **OK**.

5. Once the process is complete, choose the **Apply Package** button, and then choose **Yes**.
6. Again, if any popups or anything just click through them.
7. Once it completes, sign out of Business Central and then sign back in again.
8. You now have data.

Use the right user for your testing

Do not do your testing with a user that has SUPER permissions marked. The SUPER user can do all without issue and you won't catch your true app bugs. No live customer will have several users marked with this permission set. Therefore, we cannot test with it. You need to setup a user in your test environment that only has the BUS FULL ACCESS permission set, LOCAL, and any of your own permission sets. For information on how to set up this user, see this blog Enabling Premium Experience in Business Central Sandbox Containers.

Testing your app

Now it is time to test your app. The following are all things you must do as part of your testing effort.

- Test your app in its entirety. We expect you to test 100% of the functionality of your app. Testing just a few areas of your app will not suffice. Test everything.
- We are not going to test 100% of the functionality of your app. We expect you to be doing that.
- If the testing works for you, it will most likely work for us.
- Ensure that no permission errors are thrown for any of your app's functionality.
- With the ESSENTIAL user (before you assign your permission sets to it), make sure that the user can still use the core Business Central without facing permission errors. You must allow that user to do things such as accessing the Customer card, posting sales order, and so on.

Maintaining your app

Although we do regular testing of your app when we prepare a new version of Dynamics 365 Business Central, we expect you to do the same on your end. You have access to the same builds that we do through the Collaborate program. You can do more thorough testing than we can because you know your app the best. By doing this testing, you can catch future Dynamics 365 Business Central changes that may impact or break your app. Catching these changes in advance leaves less risk for customers to run into them.

You should be doing regular testing against our next minor (monthly) and next major (bi-annual) release branches that ports into our monthly service updates. To test against these builds, sign into aka.ms/collaborate, navigate to packages, and locate the package named **Working With Business Central Insider Builds**. The package contains a Shared Access Signature token with which you can download artifacts of future releases for use with Docker.

See Also

Checklist for Submitting Your App
Rules and Guidelines for AL Code

User Scenario Documentation

2/17/2021 • 3 minutes to read • Edit Online

One of the keys to a successful extension validation is a document that guides the tester through the setup and usage of the extension. You must include a document that helps Microsoft test some of the key scenarios of your extension. We want to ensure that we are validating the functionality in the correct manner. Following are some key points to keep in mind when writing the user scenario document.

- Be as detailed as possible. No detail is too small. If a field needs a specific value, include that value in your document.
- Keep the inexperienced user in mind. You know your app well, but other users Microsoft does not.
- Use screenshots as much as possible. They provide a good picture of what you want the user to accomplish.
- Provide all prerequisite and setup steps required for successful test scenario completion.
    - If your app requires setup of its own, include those details.
    - If any setup is extensive, consider using the import of Rapid Start packages.
    - If your app has a dependency on non-standard settings in the core default version of Business Central, include those details. The Microsoft-provided demo data might not have everything that your app needs to work properly.
- Include the most important functionality scenarios of your extension. We are not looking to test your entire extension, but we do want to ensure we are validating the most used aspects of your app.
    - Do not give a summary as to what these scenarios do. List step by step details instead. Again, the tester doing the validation might not have the same product knowledge as you do.
    - It should be possible for the inexperienced tester to **complete the user scenario test in less than 20 minutes** (after installation and startup).

> **NOTE**
> This is not the same as the requirement to include Help for your functionality. For more information about getting started with extending and customizing the Business Central user assistance, see User assistance model.

Use the correct Business Central version

We recommend that you use Docker as a guide for writing your user scenario document. This way, you can take screenshots and other visuals that really help the tester walk through your validation. Keep these things in mind:

- Do not base your user document on an on-premises environment. Business Central on-premises deployments can have different windows, data, and so on. As a result, your document can lead to confusion and differences in our results.
- Use the correct Business Central version. If you are basing your document on a build that is months old, many things could now be different in the latest production environment. This also can lead to much confusion. For more information, see Current Build - Developing for Dynamics 365 Business Central on the Collaborate site.
- Use the correct data for your document. Do not submit a document based on custom data that our testers will not have access to. You should always base your documents on the base demo CRONUS data and then

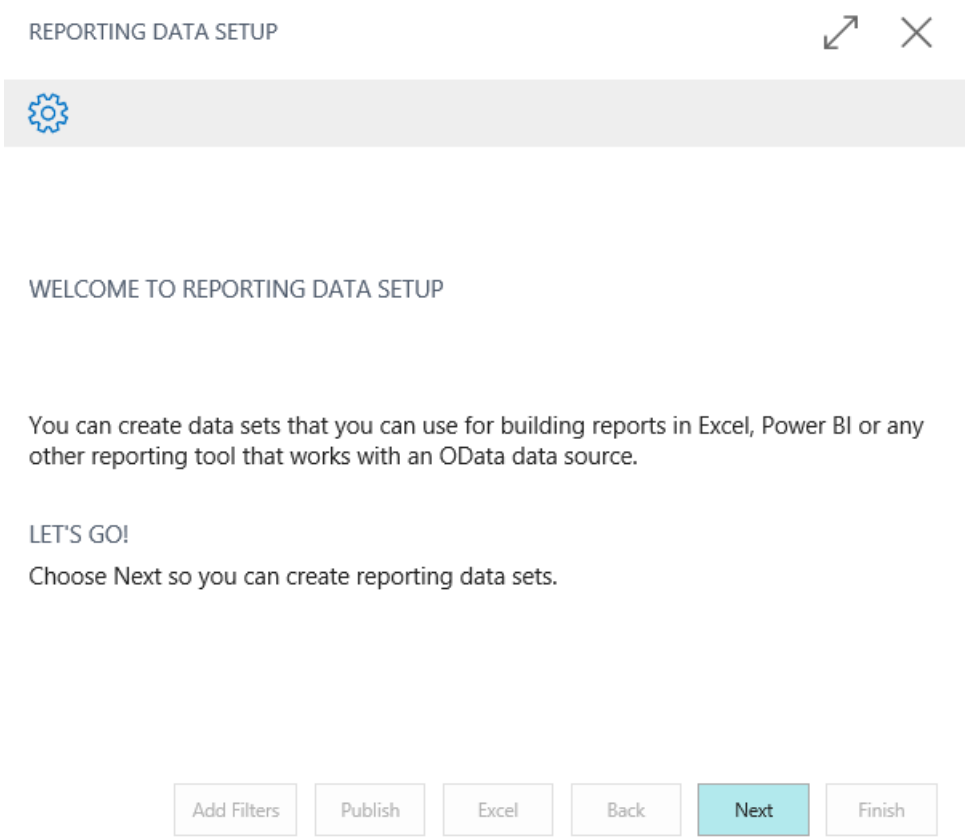
include Rapid Start packages with any additional data.

- If your app's functionality is different per country, provide that important information. Some of the steps might be different (for your app) between US and DK. If that is the case, mention that in the document.

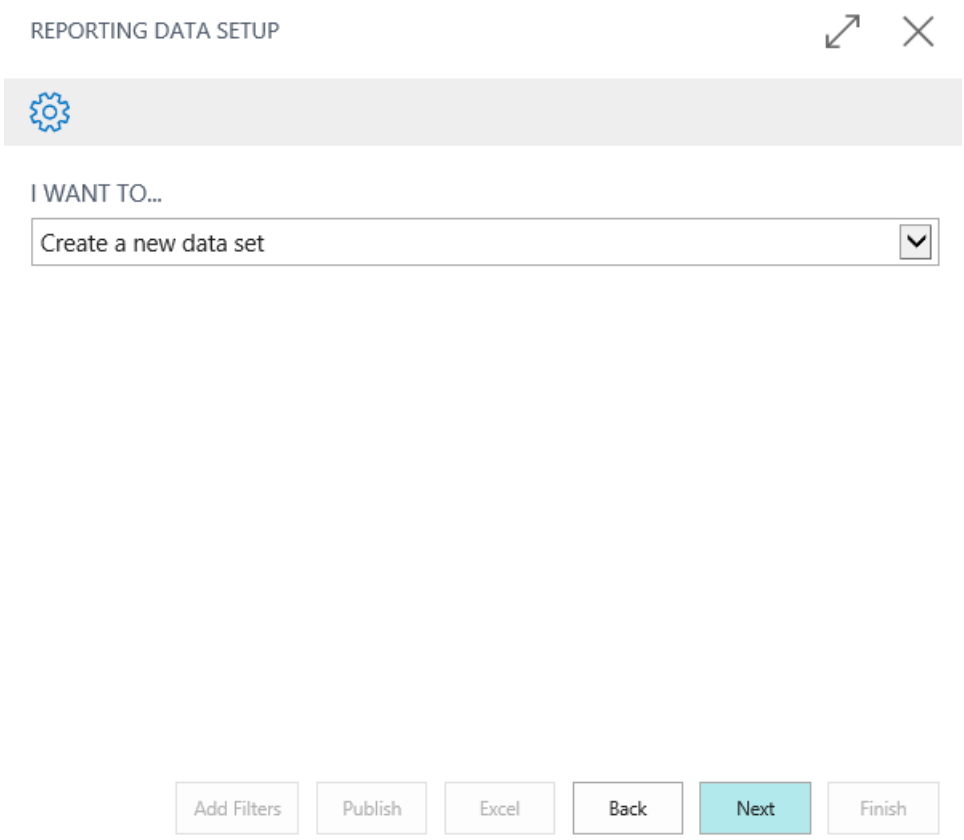
Example

Here is an example of the level of detail we are looking for, based on running the Assisted Setup wizard:

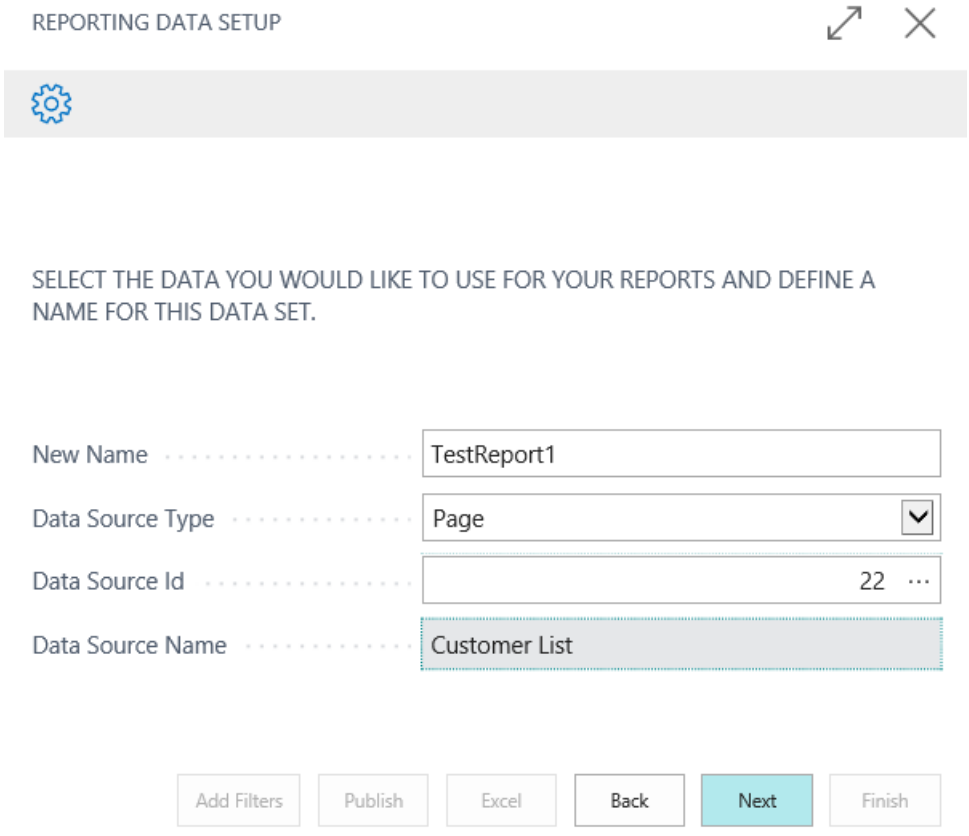
1. On the Home Page, click the **Setup & Extensions** dropdown
2. Click **Assisted Setup**
3. Click the **Set up reporting data** link
4. This will launch the wizard for this process



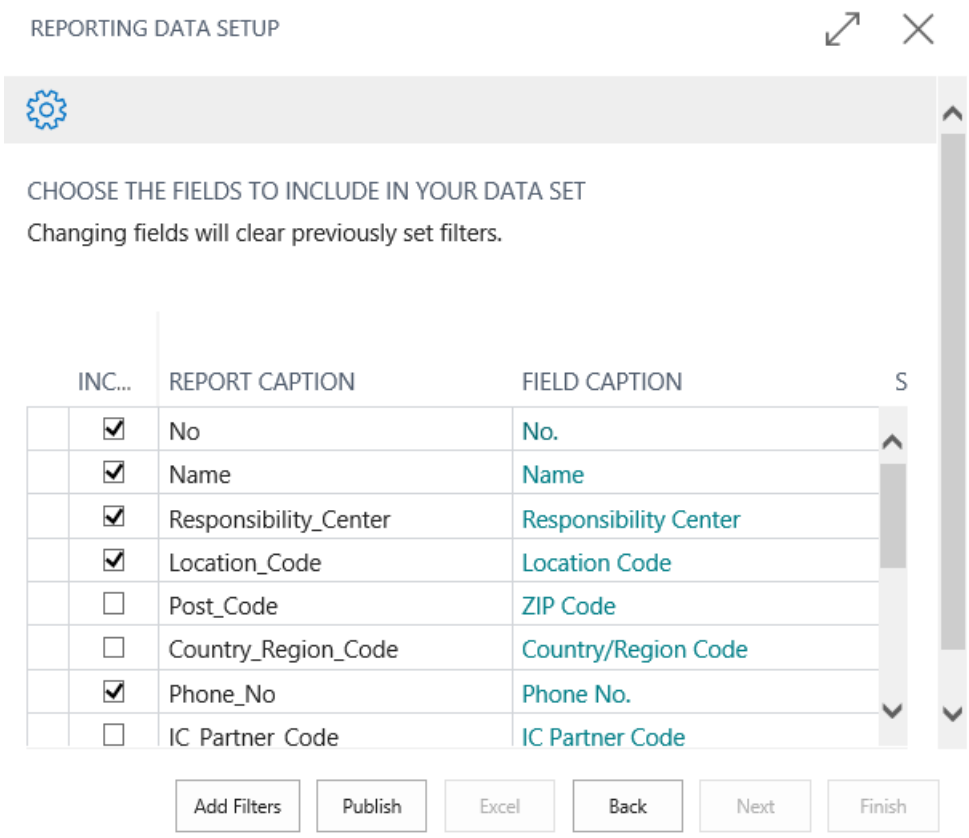
5. Click **Next**



6. Click **Next**
7. New Name = TestReport1
8. Data Source Type = Page
9. Data Source Id = 22
10. Data Source Name = Customer List



11. Click **Next**



12. Click **Publish**

See Also

Checklist for Submitting Your App
Rules and Guidelines for AL Code
User assistance model

Restrictions on UI for Objects Exposed as Web Services

2/17/2021 • 2 minutes to read • Edit Online

Pages and code units that are designed to be exposed as Web services must not generate any UI that would cause an exception in the calling code.

SUMMARY AND INTENT: When writing code for Web services, you must not use end-user confirmation dialog boxes, message boxes, or any other page constructs in the code. Because a Web service runs independently of a user interface, running this type of code causes the code to throw an exception. The exception can be caught and handled, but the Web service will not complete.

RESOURCES: For more information, see Microsoft Dynamics NAV Web Services.

HOW TO COMPLY: Ensure that code for pages and code units that are being exposed as Web services do not use any end-user confirmation dialog boxes or message boxes.

TEST METHODOLOGY: To verify this requirement, the following tests will be performed:

1. Identify the pages and code units that are exposed as Web services during the installation of the extension.
2. Using code inspection, verify that methods from the following table are not used by the pages and code units published by the installation without conditional code that is based on GUIALLOWED=FALSE or CurrFieldNo=0 circumventing their call.

AL METHOD	APPLIES TO
CONFIRM	Codeunit/page
STRMENU	Codeunit/page
(Page RunModal)	Page
Page of type Confirmation Dialog	Page
(Request page)	Page
ERROR	Codeunit/page
BEEP	Codeunit/page
YIELD	Codeunit/page

Additionally, when running the page or code unit as a Web service, the following exception should never occur:

Microsoft.Dynamics.Nav.Types.Exceptions.NavNCLCallbackNotAllowedException: Callback functions are not allowed.

See Also

Checklist for Submitting Your App
Rules and Guidelines for AL Code

Replacing OnBeforeCompanyOpen and OnAfterCompanyOpen

2/17/2021 • 2 minutes to read • Edit Online

To improve the login time for Dynamics 365 Business Central, extensions should no longer use the **OnBeforeCompanyOpen** and **OnAfterCompanyOpen** events. Following are some recommended patterns to use in place of these events.

- Move the code to the actual usage of the extension.
- Code written on the subscribers that are called from these triggers must be safe. Every **GET**, **MODIFY**, **DELETE**, and **INSERT** operation must be written with `IF THEN`, to avoid raising conditions that could prevent user from signing in.
- The code should start with check `IF NOT GUIALLOWED THEN EXIT;` to avoid impacting Web services code. Other checks to exit early should be added to the beginning of the code so that login performance is not affected.
- If the extension is subscribing to **OnBeforeCompanyOpen** or **OnAfterCompanyOpen** in order to perform some long-running data update, then either call **Update** when the extension gets called in code for the first time or apply the new task scheduler pattern for Update 6 and later. This applies to code that is not crucial.

  ```
  // Add 15s
  TASKSCHEDULER.CREATETASK(CODEUNIT::"Job Queue User Handler",0,TRUE,COMPANYNAME,CURRENTDATETIME +
  15000);
  ```

- If you are inserting data for a newly created company, we recommend subscribing to **OnCompanyInitialize** from Codeunit 2 instead. Use Installation or Upgrade code for the extension, or to set up on the first usage and if this is not possible only then it should be **OnCompanyInitialize** be used to populate data for new companies. This runs after every upgrade.

See Also

Checklist for Submitting Your App
Rules and Guidelines for AL Code

Building an Advanced Sample Extension

2/17/2021 • 16 minutes to read • Edit Online

It is required to submit tests with your extension in order to pass validation. This walkthrough builds an advanced sample extension which is used as the foundation for writing a test which you can read about in the Testing the Advanced Sample Extension topic. If you are new to building extensions, we suggest that you get familiar with Building your first sample extension that uses new objects and extension objects.

For information about submitting your app to AppSource, see Checklist for Submitting Your App.

This walkthrough will guide you through all the steps that you must follow to create the sample extension in AL. The final result can be published, installed, and tested on your sandbox. After you have built your extension, you must write the test for it.

About this walkthrough

This walkthrough illustrates the following tasks:

- Developing a sample extension that uses codeunits, tables, card pages, list pages, navigate page (Assisted Setup) actions, and events and it includes tooltips and links to context-sensitive Help.
- Creating extension objects that can be used to modify page and table objects.
- Initializing the database during the installation of the extension.
- Developing a sample test that tests external calls to a service, events, permissions, actions, navigate page (Assisted Setup), and other modified pages.
- Running the sample test using the Test Tool.

Prerequisites

To complete this walkthrough, you will need:

- The Dynamics 365 Business Central tenant
- Visual Studio Code
- The AL Language extension for Visual Studio Code

For more information on how to get started with your first extension for Dynamics 365 Business Central, see Getting Started. It is recommended to try out simpler examples, before starting this walkthrough.

Customer Rewards extension overview

This sample extension enables the ability to set up any number of reward levels and the minimum number of rewards points required to attain that level. When the sample extension is installed, customers begin to accrue one reward point per sales order. When no reward levels are set up, the customer's reward level is set to 'NONE' even though the customer may have reward points. To begin using the sample extension, the user must accept the extension terms and activate the extension by entering a valid activation code using the **Customer Rewards Assisted Setup Wizard**. Following all the steps of this walkthrough allows you to publish the extension on your tenant and create a possible new feature for your customers.

Developing the sample Customer Rewards extension

In the following section, you will be adding the objects that are needed for the Customer Rewards extension.

Customer Rewards table objects

First, we will get started with the table objects that store the data.

Reward Level table object

The following code adds a new table 50100 **Reward Level** for storing reward level information set up by the user. The table consists of two fields: **Level** and **Minimum Reward Points**.

```
table 50100 "Reward Level"
{
    fields
    {
        field(1; Level; Text[20]) { }

        field(2; "Minimum Reward Points"; Integer)
        {
            MinValue = 0;
            NotBlank = true;

            trigger OnValidate();
            var
                tempPoints: Integer;
                RewardLevel: Record "Reward Level";
            begin
                tempPoints := "Minimum Reward Points";
                RewardLevel.SetRange("Minimum Reward Points", tempPoints);
                if RewardLevel.FindFirst then
                    Error('Minimum Reward Points must be unique');
            end;
        }
    }

    keys
    {
        key(PK; Level)
        {
            Clustered = true;
        }
        key("Minimum Reward Points"; "Minimum Reward Points") { }
    }

    trigger OnInsert();
    begin

        Validate("Minimum Reward Points");
    end;

    trigger OnModify();
    begin
        Validate("Minimum Reward Points");
    end;
}
```

Activation Code Information table object

The following code adds a new table 50101 **Activation Code Information** for storing activation information for the extension. The table consists of three fields: **ActivationCode**, **Date Activated**, and **Expiration Date**.

```
table 50101 "Activation Code Information"
{
    fields
    {
        field(1; ActivationCode; Text[14])
        {
            Description = 'Activation code used to activate Customer Rewards';
        }

        field(2; "Date Activated"; Date)
        {
            Description = 'Date Customer Rewards was activated';
        }

        field(3; "Expiration Date"; Date)
        {
            Description = 'Date Customer Rewards activation expires';
        }
    }

    keys
    {
        key(PK; ActivationCode)
        {
            Clustered = true;
        }
    }
}
```

Customer Rewards Mgt. Setup table object

The following code adds a new table 50102 **Customer Rewards Mgt. Setup** for storing information about the codeunit that should be used to handle events in the extension. This enables us to mock events in our sample test. The table consists of two fields: **Primary Key** and **Customer Rewards Ext. Mgt. Codeunit ID**.

```
table 50102 "Customer Rewards Mgt. Setup"
{
    fields
    {
        field(1; "Primary Key"; Code[10])
        {
        }

        field(2; "Customer Rewards Ext. Mgt. Codeunit ID"; Integer)
        {
            TableRelation = "CodeUnit Metadata".ID;
        }
    }

    keys
    {
        key(PK; "Primary Key")
        {
            Clustered = true;
        }
    }
}
```

Customer Rewards table extension objects

Customer table extension object

The **Customer** table, like many other tables, is part of the Dynamics 365 Business Central service and it cannot be modified directly by developers. To add additional fields or to change properties on this table, developers

must create a new type of object; a table extension. The following code creates a table extension for the**Customer**table and adds the **RewardPoints** field.

```
tableextension 50100 "CustomerTable Ext." extends Customer
{
    fields
    {
        field(10001; RewardPoints; Integer)
        {
            MinValue = 0;
        }
    }
}
```

Customer Rewards page objects

For each page object, you can specify the target Help page that describes the feature that the page object is part of. The `ContextSensitiveHelpPage` property on the page object works together with the link that is specified in the app.json file. For more information, see Configure Context-Sensitive Help.

Customer Rewards Wizard page object

The following code adds the50100 **Customer Rewards Wizard**page that enables the user to accept the terms for using the extension as well as activating the extension. The page consists of a welcome step, an activation step, and a finish step. The welcome step has a checkbox for the Terms of Use that must be enabled. The activation step has a text box where the activation code must be entered for validation. A valid activation code for this sample extension is any 14 character alphanumeric code.

```
page 50100 "Customer Rewards Wizard"
{
    // Specifies that this page will be a navigate page.
    PageType = NavigatePage;
    Caption = 'Customer Rewards assisted setup guide';
    ContextSensitiveHelpPage = 'sales-rewards';

    layout
    {
        area(content)
        {
            group(MediaStandard)
            {
                Caption = '';
                Editable = false;
                Visible = TopBannerVisible;

                field("MediaResourcesStandard.""Media Reference"""; MediaResourcesStandard."Media
Reference")
                {
                    ApplicationArea = All;
                    Editable = false;
                    ShowCaption = false;
                }
            }

            group(FirstPage)
            {
                Caption = '';
                Visible = FirstPageVisible;

                group("Welcome")
                {
                    Caption = 'Welcome';
                    Visible = FirstPageVisible;

                    group(Introduction)
```

```
                    {
                        Caption = '';
                        InstructionalText = 'This Customer Rewards extension is a sample extension. It adds
rewards tiers support for Customers.';
                        Visible = FirstPageVisible;

                        field(Spacer1; '')
                        {
                            ApplicationArea = All;
                            ShowCaption = false;
                            Editable = false;
                            MultiLine = true;
                        }
                    }

                    group("Terms")
                    {
                        Caption = 'Terms of Use';
                        Visible = FirstPageVisible;

                        group(Terms1)
                        {
                            Caption = '';
                            InstructionalText = 'By enabling the Customer Rewards extension...';
                            Visible = FirstPageVisible;
                        }
                    }

                    group(Terms2)
                    {
                        Caption = '';

                        field(EnableFeature; EnableCustomerRewards)
                        {
                            ApplicationArea = All;
                            MultiLine = true;
                            Editable = true;
                            Caption = 'I understand and accept these terms.';

                            trigger OnValidate();
                            begin
                                ShowFirstPage;
                            end;
                        }
                    }
                }
            }

            group(SecondPage)
            {
                Caption = '';
                Visible = SecondPageVisible;

                group("Activation")
                {
                    Caption = 'Activation';
                    Visible = SecondPageVisible;

                    field(Spacer2; '')
                    {
                        ApplicationArea = All;
                        ShowCaption = false;
                        Editable = false;
                        MultiLine = true;
                    }

                    group(ActivationMessage)
                    {
                        Caption = '';
```

```
                        InstructionalText = 'Enter your 14 digit activation code to continue';
                        Visible = SecondPageVisible;

                        field(Activationcode; ActivationCode)
                        {
                            ApplicationArea = All;
                            ShowCaption = false;
                            Editable = true;
                        }
                    }
                }
            }

            group(FinalPage)
            {
                Caption = '';
                Visible = FinalPageVisible;

                group("ActivationDone")
                {
                    Caption = 'You''re done!';
                    Visible = FinalPageVisible;

                    group(DoneMessage)
                    {
                        Caption = '';
                        InstructionalText = 'Click Finish to setup your rewards level and start using
Customer Rewards.';
                        Visible = FinalPageVisible;
                    }
                }
            }
        }
    }

    actions
    {
        area(Processing)
        {
            action(ActionBack)
            {
                ApplicationArea = All;
                Caption = 'Back';
                Enabled = BackEnabled;
                Visible = BackEnabled;
                Image = PreviousRecord;
                InFooterBar = true;

                trigger OnAction();
                begin
                    NextStep(true);
                end;
            }

            action(ActionNext)
            {
                ApplicationArea = All;
                Caption = 'Next';
                Enabled = NextEnabled;
                Visible = NextEnabled;
                Image = NextRecord;
                InFooterBar = true;

                trigger OnAction();
                begin
                    NextStep(false);
                end;
            }
```

```
        action(ActionActivate)
        {
            ApplicationArea = All;
            Caption = 'Activate';
            Enabled = ActivateEnabled;
            Visible = ActivateEnabled;
            Image = NextRecord;
            InFooterBar = true;

            trigger OnAction();
            var
                CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
            begin
                if ActivationCode = '' then
                    Error('Activation code cannot be blank.');

                if Text.StrLen(ActivationCode) <> 14 then
                    Error('Activation code must have 14 digits.');

                if CustomerRewardsExtMgt.ActivateCustomerRewards(ActivationCode) then
                    NextStep(false)
                else
                    Error('Activation failed. Please check the activtion code you entered.');
            end;
        }

        action(ActionFinish)
        {
            ApplicationArea = All;
            Caption = 'Finish';
            Enabled = FinalPageVisible;
            Image = Approve;
            InFooterBar = true;

            trigger OnAction();
            begin
                FinishAndEnableCustomerRewards
            end;
        }
    }
}

trigger OnInit();
begin
    LoadTopBanners;
end;

trigger OnOpenPage();
begin
    Step := Step::First;
    EnableControls;
end;

local procedure EnableControls();
begin
    ResetControls;

    case Step of
    Step::First :
      ShowFirstPage;

    Step::Second :
      ShowSecondPage;

    Step::Finish :
      ShowFinalPage;
    end;
end;
```

```
local procedure NextStep(Backwards: Boolean);
begin
    if Backwards then
        Step := Step - 1
    ELSE
        Step := Step + 1;
    EnableControls;
end;

local procedure FinishAndEnableCustomerRewards();
var
    CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
begin
    CurrPage.Close;
    CustomerRewardsExtMgt.OpenRewardsLevelPage;
end;

local procedure ShowFirstPage();
begin
    FirstPageVisible := true;
    SecondPageVisible := false;
    FinishEnabled := false;
    BackEnabled := false;
    ActivateEnabled := false;
    NextEnabled := EnableCustomerRewards;
end;

local procedure ShowSecondPage();
begin
    FirstPageVisible := false;
    SecondPageVisible := true;
    FinishEnabled := false;
    BackEnabled := true;
    NextEnabled := false;
    ActivateEnabled := true;
end;

local procedure ShowFinalPage();
begin
    FinalPageVisible := true;
    BackEnabled := true;
    NextEnabled := false;
    ActivateEnabled := false;
end;

local procedure ResetControls();
begin
    FinishEnabled := true;
    BackEnabled := true;
    NextEnabled := true;
    ActivateEnabled := true;
    FirstPageVisible := false;
    SecondPageVisible := false;
    FinalPageVisible := false;
end;

local procedure LoadTopBanners();
begin
    if MediaRepositoryStandard.GET('AssistedSetup-NoText-400px.png', FORMAT(CURRENTCLIENTTYPE))
  then
        if MediaResourcesStandard.GET(MediaRepositoryStandard."Media Resources Ref")
    then
            TopBannerVisible := MediaResourcesStandard."Media Reference".HASVALUE;
end;

var
    MediaRepositoryStandard: Record 9400;
    MediaResourcesStandard: Record 2000000182;
```

```
        Step: Option First, Second, Finish;
        ActivationCode: Text;
        TopBannerVisible: Boolean;
        FirstPageVisible: Boolean;
        SecondPageVisible: Boolean;
        FinalPageVisible: Boolean;
        FinishEnabled: Boolean;
        BackEnabled: Boolean;
        NextEnabled: Boolean;
        ActivateEnabled: Boolean;
        EnableCustomerRewards: Boolean;
}
```

Rewards Level List page object

The following code adds the50101 **Rewards Level List**page that enables the user to view, edit, or add new reward levels and their corresponding minimum required points. The code example includes tooltips for controls and a relative link to context-sensitive Help.

```
page 50101 "Rewards Level List"
{
    PageType = List;
    ContextSensitiveHelpPage = 'sales-rewards';
    SourceTable = "Reward Level";
    SourceTableView = sorting ("Minimum Reward Points") order(ascending);

    layout
    {
        area(content)
        {
            repeater(Group)
            {
                field(Level; Level)
                {
                    ApplicationArea = All;
                    Tooltip = 'Specifies the level of reward that the customer has at this point.';
                }

                field("Minimum Reward Points"; "Minimum Reward Points")
                {
                    ApplicationArea = All;
                    Tooltip = 'Specifies the number of points that customers must have to reach this
level.';
                }
            }
        }
    }

    trigger OnOpenPage();
    begin

        if(not CustomerRewardsExtMgt.IsCustomerRewardsActivated) then
            Error(NotActivatedTxt);
    end;

    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        NotActivatedTxt: Label 'Customer Rewards is not activated';
}
```

Customer Rewards page extension objects

Customer card page extension object

A page extension object can be used to add new functionality to pages that are part of the Dynamics 365 Business Central service. The following page extension object extends the **Customer Card** page object by

adding two field controls: **RewardLevel** and **RewardPoints** after the **Name** field control on the page. The fields are added in the layout section.

```
pageextension 50100 "Customer Card Ext." extends "Customer Card"
{
    layout
    {
        addafter(Name)
        {
            field(RewardLevel; RewardLevel)
            {
                ApplicationArea = All;
                Caption = 'Reward Level';
                Description = 'Reward level of the customer.';
                ToolTip = 'Specifies the level of reward that the customer has at this point.';
                Editable = false;
            }

            field(RewardPoints; RewardPoints)
            {
                ApplicationArea = All;
                Caption = 'Reward Points';
                Description = 'Reward points accrued by customer';
                ToolTip = 'Specifies the total number of points that the customer has at this point.';
                Editable = false;
            }
        }
    }

    trigger OnAfterGetRecord();
    var
        CustomerRewardsMgtExt: Codeunit "Customer Rewards Ext. Mgt.";
    begin
        // Get the reward level associated with reward points
        RewardLevel := CustomerRewardsMgtExt.GetRewardLevel(RewardPoints);
    end;

    var
        RewardLevel: Text;
}
```

Customer list page extension object

A page extension object can be used to add new functionality to pages that are part of the Dynamics 365 Business Central service. The following page extension object extends the **Customer List** page object by adding one action control; **Reward Levels** to the **Customer** group on the page.

```
pageextension 50101 "Customer List Ext." extends "Customer List"
{
    actions
    {
        addfirst("&Customer")
        {
            action("Reward Levels")
            {
                ApplicationArea = All;
                Image = CustomerRating;
                Promoted = true;
                PromotedCategory = Process;
                PromotedIsBig = true;
                ToolTip = 'Open the list of reward levels.';


                trigger OnAction();
                begin
                    if CustomerRewardsExtMgt.IsCustomerRewardsActivated then
                        CustomerRewardsExtMgt.OpenRewardsLevelPage
                    else
                        CustomerRewardsExtMgt.OpenCustomerRewardsWizard;
                end;
            }
        }
    }

    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
}
```

Customer Rewards codeunit objects

Customer Rewards Install Logic codeunit object

The following code adds the50100 **Customer Rewards Install Logic**codeunit that initializes the default codeunit that will be used for handling events. Because this is an install codeunit, it has its **Subtype** property set to **Install**. The **OnInstallAppPerCompany** trigger is run when the extension is installed for the first time and the same version is re-installed.

```
codeunit 50100 "Customer Rewards Install Logic"
{
    // Customer Rewards Install Logic
    Subtype = Install;

    trigger OnInstallAppPerCompany();
    begin
        SetDefaultCustomerRewardsExtMgtCodeunit;
    end;

    procedure SetDefaultCustomerRewardsExtMgtCodeunit();
    var
        CustomerRewardsExtMgtSetup: Record "Customer Rewards Mgt. Setup";
    begin
        CustomerRewardsExtMgtSetup.DeleteAll;
        CustomerRewardsExtMgtSetup.Init;
        // Default Customer Rewards Ext. Mgt codeunit to use for handling events
        CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID" := Codeunit::"Customer Rewards
Ext. Mgt.";
        CustomerRewardsExtMgtSetup.Insert;
    end;
}
```

Customer Rewards Ext. Mgt. codeunit object

The50101 **Customer Rewards Ext. Mgt.** codeunit encapsulates most of the logic and functionality required

for the Customer Rewards extension. This codeunit contains examples of how we can use events to react to specific actions or behavior that occur within our extension. In this sample extension, there is the need to make a call to an external service or API to validate activation codes entered by the user. Typically, you may do this by defining procedures that take in the activation code and then make calls to the API. Instead of using that approach, we use events in AL. Let us look at the following code from the codeunit.

```
    // Activates Customer Rewards if activation code is validated successfully
    procedure ActivateCustomerRewards(ActivationCode: Text): Boolean;
    var
        ActivationCodeInfo: Record "Activation Code Information";
    begin
        // raise event
        OnGetActivationCodeStatusFromServer(ActivationCode);
        exit(ActivationCodeInfo.Get(ActivationCode));
    end;

    // publishes event
    [IntegrationEvent(false, false)]
    procedure OnGetActivationCodeStatusFromServer(ActivationCode: Text);
    begin
    end;

    // Subscribes to OnGetActivationCodeStatusFromServer event and handles it when the event is raised
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Customer Rewards Ext. Mgt.",
'OnGetActivationCodeStatusFromServer', '', false, false)]
    local procedure OnGetActivationCodeStatusFromServerSubscriber(ActivationCode: Text);
    var
        ActivationCodeInfo: Record "Activation Code Information";
        ResponseText: Text;
        Result: JsonToken;
        JsonRepsonse: JsonToken;

    begin
        if not CanHandle then
            exit; // use the mock
        // Get response from external service and update activation code information if successful
        if(GetHttpResponse(ActivationCode, ResponseText)) then begin
            JsonRepsonse.ReadFrom(ResponseText);

            if(JsonRepsonse.SelectToken('ActivationResponse', Result)) then begin

                if(Result.AsValue().AsText() = 'Success') then begin

                    if(ActivationCodeInfo.FindFirst()) then
                        ActivationCodeInfo.Delete;

                    ActivationCodeInfo.Init;
                    ActivationCodeInfo.ActivationCode := ActivationCode;
                    ActivationCodeInfo."Date Activated" := Today;
                    ActivationCodeInfo."Expiration Date" := CALCDATE('<1Y>', Today);
                    ActivationCodeInfo.Insert;
                end;
            end;
        end;
    end;

    // Helper method to make calls to a service to validate activation code
    local procedure GetHttpResponse(ActivationCode: Text; var ResponseText: Text): Boolean;
    begin
        // You will typically make external calls / http requests to your service to validate the activation
code
        // here but for the sample extension we simply return a successful dummy response
        if ActivationCode = '' then
            exit(false);

        ResponseText := DummySuccessResponseTxt;
        exit(true);
    end;
```

We define an event publisher method **OnGetActivationCodeStatusFromServer** that accepts the activation code entered by the user as a parameter, and, a subscriber method **OnGetActivationCodeStatusFromServerSubscriber** to listen for and handle the event. When the

ActivateCustomerRewards procedure is run, the **OnGetActivationCodeStatusFromServer** event is raised. Because the **EventSubscriberInstance** property for the codeunit is set to **Static-Automatic** by default, the **OnGetActivationCodeStatusFromServerSubscriber** procedure is called. In this procedure, we handle the raised event by first checking if the current codeunit has been defined for handling this event. If the codeunit can handle the event, the **GetHttpResponse** helper procedure is called to validate the activation code. Depending on the response, Customer Rewards is activated or not.

By using events when the extension makes external calls to a service, we are able to mock the behavior of what happens when events are raised. This becomes particularly useful when writing tests for the extension.

For more information about events, see Events in Microsoft Dynamics 365 Business Central.

Below is the full code for this codeunit.

```
codeunit 50101 "Customer Rewards Ext. Mgt."
{
    var
        DummySuccessResponseTxt: Label '{"ActivationResponse": "Success"}', Locked = true;
        NoRewardlevelTxt: Label 'NONE';

    // Determines if the extension is activated
    procedure IsCustomerRewardsActivated(): Boolean;
    var
        ActivationCodeInfo: Record "Activation Code Information";
    begin
        if not ActivationCodeInfo.FindFirst then
            exit(false);

        if(ActivationCodeInfo."Date Activated" <= Today) and(Today <= ActivationCodeInfo."Expiration Date")
then
            exit(true);
        exit(false);
    end;

    // Opens the Customer Rewards Assisted Setup Guide
    procedure OpenCustomerRewardsWizard();
    var
        CustomerRewardsWizard: Page "Customer Rewards Wizard";
    begin
        CustomerRewardsWizard.RunModal;
    end;

    // Opens the Reward Level page
    procedure OpenRewardsLevelPage();
    var
        RewardsLevelPage: Page "Rewards Level List";
    begin
        RewardsLevelPage.Run;
    end;

    // Determines the correponding reward level and returns it
    procedure GetRewardLevel(RewardPoints: Integer) RewardLevelTxt: Text;
    var
        RewardLevelRec: Record "Reward Level";
        MinRewardLevelPoints: Integer;
    begin
        RewardLevelTxt := NoRewardlevelTxt;

        if RewardLevelRec.IsEmpty then
            exit;
        RewardLevelRec.SetRange("Minimum Reward Points", 0, RewardPoints);
        RewardLevelRec.SetCurrentKey("Minimum Reward Points"); // sorted in ascending order

        if not RewardLevelRec.FindFirst then
            exit;
        MinRewardLevelPoints := RewardLevelRec."Minimum Reward Points";
```

```
        if RewardPoints >= MinRewardLevelPoints then begin
            RewardLevelRec.Reset;
            RewardLevelRec.SetRange("Minimum Reward Points", MinRewardLevelPoints, RewardPoints);
            RewardLevelRec.SetCurrentKey("Minimum Reward Points"); // sorted in ascending order
            RewardLevelRec.FindLast;
            RewardLevelTxt := RewardLevelRec.Level;
        end;
    end;

    // Activates Customer Rewards if activation code is validated successfully
    procedure ActivateCustomerRewards(ActivationCode: Text): Boolean;
    var
        ActivationCodeInfo: Record "Activation Code Information";
    begin
        // raise event
        OnGetActivationCodeStatusFromServer(ActivationCode);
        exit(ActivationCodeInfo.Get(ActivationCode));
    end;

    // publishes event
    [IntegrationEvent(false, false)]
    procedure OnGetActivationCodeStatusFromServer(ActivationCode: Text);
    begin
    end;

    // Subscribes to OnGetActivationCodeStatusFromServer event and handles it when the event is raised
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Customer Rewards Ext. Mgt.",
'OnGetActivationCodeStatusFromServer', '', false, false)]
    local procedure OnGetActivationCodeStatusFromServerSubscriber(ActivationCode: Text);
    var
        ActivationCodeInfo: Record "Activation Code Information";
        ResponseText: Text;
        Result: JsonToken;
        JsonRepsonse: JsonToken;
    begin
        if not CanHandle then
            exit; // use the mock

        // Get response from external service and update activation code information if successful
        if(GetHttpResponse(ActivationCode, ResponseText)) then begin
            JsonRepsonse.ReadFrom(ResponseText);

            if(JsonRepsonse.SelectToken('ActivationResponse', Result)) then begin

                if(Result.AsValue().AsText() = 'Success') then begin

                    if(ActivationCodeInfo.FindFirst()) then
                        ActivationCodeInfo.Delete;

                    ActivationCodeInfo.Init;
                    ActivationCodeInfo.ActivationCode := ActivationCode;
                    ActivationCodeInfo."Date Activated" := Today;
                    ActivationCodeInfo."Expiration Date" := CALCDATE('<1Y>', Today);
                    ActivationCodeInfo.Insert;

                end;
            end;
        end;
    end;

    // Helper method to make calls to a service to validate activation code
    local procedure GetHttpResponse(ActivationCode: Text; var ResponseText: Text): Boolean;
    begin
        // You will typically make external calls / http requests to your service to validate the activation
code
        // here but for the sample extension we simply return a successful dummy response
        if ActivationCode = '' then
            exit(false);
```

```
            exit(false);

        ResponseText := DummySuccessResponseTxt;
        exit(true);
    end;

    // Subcribes to the OnAfterReleaseSalesDoc event and increases reward points for the sell to customer in
posted sales order
    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Release Sales Document", 'OnAfterReleaseSalesDoc', '',
false, false)]
    local procedure OnAfterReleaseSalesDocSubscriber(VAR SalesHeader: Record "Sales Header"; PreviewMode:
Boolean; LinesWereModified: Boolean);
    var
        Customer: Record Customer;
    begin
        if SalesHeader.Status <> SalesHeader.Status::Released then
            exit;

        Customer.Get(SalesHeader."Sell-to Customer No.");
        Customer.RewardPoints += 1; // Add a point for each new sales order
        Customer.Modify;
    end;

    // Checks if the current codeunit is allowed to handle Customer Rewards Activation requests rather than
a mock.
    local procedure CanHandle(): Boolean;
    var
        CustomerRewardsExtMgtSetup: Record "Customer Rewards Mgt. Setup";
    begin
        if CustomerRewardsExtMgtSetup.Get then
            exit(CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID" = CODEUNIT::"Customer
Rewards Ext. Mgt.");
        exit(false);
    end;
}
```

Conclusion

At this point, the Customer Rewards sample extension can be published and installed on your sandbox. To continue writing tests for the sample extension, see Testing the Advanced Sample Extension.

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension
Converting Extensions V1 to Extensions V2
Configure Context-Sensitive Help

Testing the Advanced Sample Extension

2/17/2021 • 28 minutes to read • Edit Online

It is required to submit tests with your extension in order to pass validation. This walkthrough builds on the advanced sample extension which you can read about here Building an Advanced Sample Extension. If you are new to building extensions, we suggest that you get familiar with Building your first sample extension that uses new objects and extension objects. This walkthrough goes through how you develop the test for the sample CustomerRewards extension.

For information about submitting your app to AppSource, see Checklist for Submitting Your App.

Prerequisites

To complete this walkthrough, you will need:

- Dynamics 365 Business Central Docker container-based development environment For more information, see Get started with the Container Sandbox Development Environment and Running a Container-Based Development Environment
- Visual Studio Code
- The AL Language extension for Visual Studio Code

Identifying the areas of the extension that need to be tested

Before writing tests for your extension, you need to identify all the areas of the extension that need to be tested.

- Ensure that your tests cover all the setup and usage scenario steps found in the user scenario document. This includes Assisted Setup, pages, fields, actions, events, and other controls and objects used by your extension.
- The CRONUS demo company will be used for the purpose of this walkthrough. If your app requires setup within the core product or any additional data, remember to include that in your tests.
- As part of your tests, remember to include tests that verify that the extension works as expected for **a user that does not have SUPER permissions**. For more information, see Special Permission Sets.
- Your tests **should not make any requests to an external service**. Mock your external calls to prevent this from happening.

In the sample test we will consider the following:

- Logic in our **Install** codeunit.
- **Assisted Setup - Customer Rewards Wizard** page. We will verify that the wizard behaves as expected. It can be used to completion without errors. The Assisted Setup contains code that mimics making calls to an external service or API. Because our tests cannot make requests to an external service, we will mock the requests and the responses.
- **Reward Level** page. We will verify that the page behaves as expected when the user opens it whether Customer Rewards is activated or not.
- **Customer List** page. We will verify that our new **Reward Levels** action exists on the page and that it behaves as expected whether the extension is activated or not.
- **Customer Card** page. We will verify that the page has the **Reward Level** and **Reward Points** field that we added.
- **New Customer** should have zero reward points and corresponding reward level if defined.

- Different scenarios involving Customers and Sales Orders to verify that **Reward Points** work as expected and that reward levels for reward points work as defined by the user.
- Each test will also verify that the extension works for a user that does not have SUPER permissions.

Writing the tests

We will first create a new project (CustomerRewardsTest) for the tests. You are required to separate the CustomerRewards extension and the tests into separate projects.

Before we can start writing the tests for the extension, we need to do the following:

- Specify the dependencies between the extension (CustomerRewards) and the test (CustomerRewardsTest) projects.
  Our CustomerRewardsTest project will be referencing objects from the CustomerRewards project and so we will need to specify this in the `dependencies` setting in the CustomerRewardsTest project's app.json file. The `dependencies` setting takes a list of dependencies, where each dependency specifies the `appId`, `name`, `publisher`, and `version` of the base project/package that the current project/package will depend on.

> **NOTE**
> Another prerequisite is to update the app.json with a dependency to the test toolkit.

```
{
 ...
 "dependencies": [
   {
     "appId": "c228bdcf-7112-480b-a832-da81971b6feb",
     "name": "CustomerRewards",
     "publisher": "Microsoft",
     "version": "1.0.0.0"
   }
 ],
 "test": "13.0.0.0"
  ...
}
```

For more information, see JSON Files.

After setting the `dependencies` value, you will be prompted to download the symbols from the base project/package if they are not present.

Application Test Toolkit

We will be using the Application Test Toolkit to automate and run the tests that we write. The toolkit includes:

- Codeunits with test functions to test various application areas.
- Codeunits with generic and application-specific functions to reduce duplication of test code.
- Application objects for running application tests such as the **Test Tool** page.

In order to install the Application Test Toolkit:

1. Open the BCContainerHelper prompt found on the Desktop. You will see a list of functions that you can run on the container.
2. Run the `Import-TestToolkitToBCContainer` function with `-containerName` parameter to import the test toolkit

into the application database.

```
Import-TestToolkitToBCContainer -containerName <name-of-container>
```

Alternatively, if you use the `New-BCContainer` function from the BCContainerHelper PowerShell module to create your containers on Docker, you can add the `-includeTestToolkit` flag. This will install the Application Test Toolkit during the creation of your container.

Without further configuration, the Import-TestToolkitToBCContainer and New-BCContainer with -includeTestToolkit will install the framework, the libraries, and all base application tests. Both the Import-TestToolkitToBCContainer and New-BCContainer cmdlets support two additional parameters, which limits the number of apps installed:

- `-includeTestFrameworkOnly` installs the Test Framework only. This option includes the Test Runner and low-level functions such as *Any* and *Assert*.
- `-includeTestLibrariesOnly` installs the Test Framework and the Test Libraries only. Beside the Test Framework, this option includes functionality that is shared between base application tests.

Describing your tests

To help you design the relevant tests for your functionality, you can write scenarios that outline what you want to test, and you can write test criteria in the GIVEN-WHEN-THEN format. By adding comments based on feature, scenario, and GIVEN-WHEN-THEN, you add structure to your test code and make tests readable.

The following sections provide an overview of the tags that we recommend you to use.

FEATURE Tag

```
// [FEATURE] [<FeatureTag1>][<FeatureTagN>]
```

`FeatureTag` represents the name of the feature, application area, functional area, or another aspect of the application. This list of tags must point to an area of your solution that is touched by the test. Order tags in descending importance. Start with the most important tags referring to the WHEN or THEN steps. The `[FEATURE]` tag can be set for the whole test codeunit. This means all tests in this codeunit will inherit the list of tags set there. If a test is supposed to have the same list of tags as the codeunit has, you do not have to add the `[FEATURE]` tag for this test. Add the tags only if the test has something specific to say.

SCENARIO Tag

```
// [SCENARIO <ScenarioID>] <TestDescription>`
```

`ScenarioID` links the test to a work item for the functionality. For example, if you use Visual Studio Online or Team Foundation Server, `[SCENARIO 12345]` represents a work item with the ID 12345.

TestDescriptionrepresents a short description of the purpose of the test, such as *Annie can apply a deferral template to a purchase order*.

GIVEN-WHEN-THEN Tags

The `GIVEN-WHEN-THEN` tags provide a framework for the specific test criteria.

TAG	DESCRIPTION

TAG	DESCRIPTION
GIVEN	Describes one step in setting up the test. If you feel a need to add an AND, you should probably add a separate GIVEN. In most of cases, in order to run an action under test, you must prepare the database. Tests can be complex, so you can add more than one GIVEN. They can come in one block or comment particular lines of code. Do not try to repeat code and comment each line. Instead, add information of a higher level that would be valuable when reading without the test code.
WHEN	Describes the action under test. A test is to test one thing. There should be only one WHEN in a test. It is the line of code that changes the state of something that we are going to verify. If you feel a need to add more than one WHEN followed by different verification, you should split this test in two or more tests.
THEN	Describes what is verified by the test. All tests must have a verification part. If there is no verification, the test does not test anything. You can add more than one THEN tag.

We can now begin writing the tests for the extension.

MockCustomerRewardsExtMgt codeunit object

The50102 **MockCustomerRewardsExtMgt**codeunit contains all the code that mocks the process of validating the activation code for Customer Rewards. Because we cannot make requests to external services in the tests, we define a subscriber method **MockOnGetActivationCodeStatusFromServerSubscriber** for handling the **OnGetActivationCodeStatusFromServer** event when it is raised in the **Customer Rewards Ext. Mgt.** codeunit. The **EventSubscriberInstance** property for this codeunit is set to **Manual** so that we can control when the subscriber function is called. We want the subscriber method to be called only during our tests. We also define a Setup procedure that modifies the **Customer Rewards Ext. Mgt. Codeunit ID** in the **Customer Rewards Mgt. Setup** table so that the actual **OnGetActivationCodeStatusFromServerSubscriber** will not handle **OnGetActivationCodeStatusFromServer** event when it is raised.

```
codeunit 50102 MockCustomerRewardsExtMgt

{

    // When set to Manual subscribers in this codeunit are bound to an event by calling the BINDSUBSCRIPTION
method.
    // This enables you to control which event subscriber instances are called when an event is raised.
    // If the BINDSUBSCRIPTION method is not called, then nothing will happen when the published event is
raised.

    EventSubscriberInstance = Manual;
    var
        DummyResponseTxt: Text;
        DummySuccessResponseTxt: Label '{"ActivationResponse": "Success"}', Locked = true;
        DummyFailureResponseTxt: Label '{"ActivationResponse": "Failure"}', Locked = true;

    // Mocks the response text for testing success and failure scenarios

    procedure MockActivationResponse(Success: Boolean);
    begin
        if Success then
            DummyResponseTxt := DummySuccessResponseTxt
        else
            DummyResponseTxt := DummyFailureResponseTxt;
    end;
```

```
    // Modifies the default Customer Rewards Ext. Mgt codeunit to this codeunit to prevent the
    // OnGetActivationCodeStatusFromServerSubscriber in Customer Rewards Ext. Mgt from handling
    // the OnGetActivationCodeStatusFromServer event when it is raised

    procedure Setup();
    var
        CustomerRewardsExtMgtSetup: Record "Customer Rewards Mgt. Setup";
    begin
        CustomerRewardsExtMgtSetup.Get;
        CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID" :=
Codeunit::MockCustomerRewardsExtMgt;
        CustomerRewardsExtMgtSetup.Modify;
    end;

    // Subscribes to OnGetActivationCodeStatusFromServer event and handles it when the event is raised

    [EventSubscriber(ObjectType::Codeunit, Codeunit::"Customer Rewards Ext. Mgt.",
'OnGetActivationCodeStatusFromServer', '', false, false)]

    local procedure MockOnGetActivationCodeStatusFromServerSubscriber(ActivationCode: Text);
    var
        ActivationCodeInfo: Record "Activation Code Information";
        ResponseText: Text;
        Result: JsonToken;
        JsonRepsonse: JsonToken;
    begin
        if(MockGetHttpResponse(ActivationCode, ResponseText)) then begin
            JsonRepsonse.ReadFrom(ResponseText);

            if(JsonRepsonse.SelectToken('ActivationResponse', Result)) then begin
                if(Result.AsValue().AsText() = 'Success') then begin
                    if ActivationCodeInfo.FindFirst then
                        ActivationCodeInfo.Delete;
                        ActivationCodeInfo.Init;
                        ActivationCodeInfo.ActivationCode := ActivationCode;
                        ActivationCodeInfo."Date Activated" := Today;
                        ActivationCodeInfo."Expiration Date" := CALCDATE('<1Y>', Today);
                        ActivationCodeInfo.Insert;
                end;
            end;
        end;
    end;

    // Mocks making calls to external service

    local procedure MockGetHttpResponse(ActivationCode: Text; var ResponseText: Text): Boolean;
    begin
        if ActivationCode = '' then
            exit(false);

        ResponseText := DummyResponseTxt;

        exit(true);
    end;
}
```

Customer Rewards Test codeunit object

A test codeunit must have its **Subtype** property set to **Test** and the test methods must be decorated with the `[Test]` attribute. When a test codeunit runs, it executes the **OnRun** trigger, and then executes each test method in the codeunit. By default, each test function runs in a separate database transaction, but you can use the **TransactionModel** attribute on test methods to control the transactional behavior. The outcome of a test method is either SUCCESS or FAILURE. If any error is raised by either the code that is being tested or the test code, then the outcome is FAILURE and the error is included in the results log file. Even if the outcome of one

test method is FAILURE, the next test methods are still executed.

In addition to the Application Test Toolkit, the following features are available to help you test your extension:

Test pages

Test pages mimic actual pages, but do not present any UI on a client computer. Test pages let you test the code on a page by using AL to simulate user interaction with the page. You can access the fields on a page and the properties of a page or a field by using the dot notation. You can open and close test pages, perform actions on the test page, and navigate around the test page by using AL methods.

UI handlers

To create tests that can be automated, you must handle cases when user interaction is requested by code that is being tested. UI handlers run instead of the requested UI. UI handlers provide the same exit state as the UI. For example, a test method that has a ConfirmHandler handles CONFIRM method calls. If code that is being tested calls the CONFIRM method, then the ConfirmHandler method is called instead of the CONFIRM method. You write code in the ConfirmHandler method to verify that the expected question is displayed by the CONFIRM method and you write AL code to return the relevant reply. The following table describes the available UI handlers.

FUNCTION TYPE	SYNTAX EXAMPLE	PURPOSE	
MessageHandler	`[MessageHandler]` `procedure MessageHandler(Msg : Text[1024]);`	This handler is called when a message function is invoked in the code. The parameter type, **Text**, contains the text of the function.	
ConfirmHandler	`[ConfirmHandler]` `procedure ConfirmHandlerNo(Question: Text[1024]; var Reply: Boolean);`	This handler is called when a confirm function is invoked in the code. The parameter type, **Text**, contains the text of the function and the parameter **Reply** if the response to confirm is *yes* or *no*.	
StrMenuHandler	`[StrMenuHandler]` `procedure StrMenuHandler(Option: Text[1024]; var Choice: Integer; Instruction: Text[1024]);`	This handler is called when a StrMenu function is invoked in code. The parameter type, **Text**, contains the text of the function and **Choice** is the option chosen in the StrMenu. **Options** is the list of the different option values and **Instruction** is the leading text.	
PageHandler	`[PageHandler]` `procedure MappingPageHandler(var MappingPage: TestPage 1214);`	This handler is called when a non-modal page is invoked in the code. **TestPage** is the specific page in this case.	

FUNCTION TYPE	SYNTAX EXAMPLE	PURPOSE	
ModalPageHandler	`[ModalPageHandler]` `procedure DevSelectedObjectPageHandler(var DevSelectedObjects: TestPage 89015);`	This handler is called when a modal page is invoked in the code. **TestPage** is the specific page in this case.	
ReportHandler	`[ReportHandler]` `procedure VendorListReportHandler(var VendorList: Report 301);`	This handler is called when a report is invoked in the code. **Report** is the specific report in this case.	
RequestPageHandler	`[RequestPageHandler]` `procedure SalesInvoiceReportRequestPageHandler(var SalesInvoice: TestRequestPage 206);`	This handler is called when a report is invoked in the code. **TestRequestPage** refers to the specific report ID.	

You must create a specific handler for each page that you want to handle. Any unhandled UI in the test methods of the test codeunit causes a failure of the test.

ASSERTERROR statement

When you test your extension, you should test that your code performs as expected under both successful and failing conditions. These are called positive and negative tests. To test how your extension performs under failing conditions, you can use the ASSERTERROR keyword. The ASSERTERROR keyword specifies that an error is expected at run time in the statement that follows the ASSERTERROR keyword. If a simple or compound statement that follows the ASSERTERROR keyword causes an error, then execution successfully continues to the next statement in the test function. If a statement that follows the ASSERTERROR keyword does not cause an error, then the ASSERTERROR statement itself fails with an error, and the test function that is running produces a FAILURE result.

The50103 **Customer Rewards Test** codeunit contains all the tests for the Customer Rewards extension. For each test method, we follow the following pattern:

- Initialize and set up the conditions for the test.
- Invoke the business logic that you want to test.
- Validate that the business logic performed as expected.

Let us look some of the sample tests.

TestOnInstallLogic Test

This test verifies that the logic we defined in our Install codeunit works as expected. We first call a helper method **Initialize** which initializes and cleans up any objects that will be needed for the test. The Initialize method also binds our mock codeunit **MockCustomerRewardsExtMgt** to our test codeunit so that any events raised during our test can be handled by the subscriber methods specified in our mock codeunit.

Next, we invoke the **SetDefaultCustomerRewardsExtMgtCodeunit** method, which is the method defined in our Install codeunit.

And finally, we verify using the **Assert** codeunit from the Application Test Toolkit, that the **Customer Rewards Mgt. Setup** table contains the expected codeunit ID.

TestCustomerRewardsWizardActivationPageErrorsWhenInvalidActivationCodeEntered Test

This is one of the tests that focus on the **Customer Rewards Assisted Setup Guide**. The test verifies that an

error message is displayed when a not valid activation code is entered in the wizard.

First, **Initialize** is called to clean up previous state and bind our mock subscriber methods to the test codeunit. Additionally, we set our MockActivationResponse to return FAILURE since we are mocking a not valid validation of the activation code. We also use the **Library - Lower Permissions** codeunit to restrict the users permission to one that does not have the SUPER permission.

Next, we open the **Customer Rewards Wizard** by using a Customer Rewards Wizard, the TestPage object is used to mimic the actual page. On the page, the activation code is entered and then the Activate action is invoked.

And finally, we verify that an error message is displayed because the validation of the activation code failed. If no other error is reported then we are also able to conclude that the functionality in this test can be run without the need for a SUPER permission.

TestRewardLevelsActionExistsOnCustomerListPage Test

This test verifies that the new **Reward Levels** action exists on the Customer List page.

TestCustomerHasBronzeRewardLevelAfterPostedSalesOrders Test

This is one of the tests that considers the interaction between Customers, Sales Orders, and Reward Levels. This test verifies that when two sales orders are made for a new customer, that customer accrues two reward points. Consequently, he attains the corresponding reward level for two points, which is the BRONZE reward level.

First, the test is initialized by calling Initialize. The extension is activated and then a BRONZE reward level for two points or more is set up in the **Reward Level** table.

Next, a new **Customer** is created using the **LibrarySales** codeunit from the Application Test Toolkit. And then, the **LibrarySales** codeunit is used again to create and post two sales orders for the previously created customer.

Finally, to verify that the customer got the correct reward points and level, we open the **Customer Card** using its corresponding TestPage and then verify the values in the **Reward Points** and **Reward Level** fields.

There are many more areas that we look at in the sample test. See the full codeunit below for the rest of the tests.

```
codeunit 50103 "Customer Rewards Test"

{
    // [FEATURE] [Customer Rewards]

    Subtype = Test;
    TestPermissions = Disabled;

    var
        Assert: Codeunit Assert;
        LibraryLowerPermissions: Codeunit "Library - Lower Permissions";
        LibrarySales: Codeunit "Library - Sales";
        MockCustomerRewardsExtMgt: Codeunit MockCustomerRewardsExtMgt;
        ActivatedTxt: Label 'Customer Rewards should be activated';
        NotActivatedTxt: Label 'Customer Rewards should not be activated';
        BronzeLevelTxt: Label 'BRONZE';
        SilverLevelTxt: Label 'SILVER';
        GoldLevelTxt: Label 'GOLD';
        NoLevelTxt: Label 'NONE';

    [Test]

    procedure TestOnInstallLogic();
    var
        CustomerRewardsExtMgtSetup: Record "Customer Rewards Mgt. Setup";
        CustomerRewardsInstallLogic: Codeunit "Customer Rewards Install Logic";
```

```
    begin
        // [Scenario] Check default codeunit is specified for handling events on install
        // [Given] Customer Rewards Mgt. Setup table

        Initialize;

        // [When] Install logic is run
        CustomerRewardsInstallLogic.SetDefaultCustomerRewardsExtMgtCodeunit;

        // [Then] Default Customer Rewards Ext. Mgt codeunit is specified
        Assert.AreEqual(1, CustomerRewardsExtMgtSetup.Count, 'CustomerRewardsExtMgtSetup must have exactly
one record.');

        CustomerRewardsExtMgtSetup.Get;

        Assert.AreEqual(Codeunit::"Customer Rewards Ext. Mgt.", CustomerRewardsExtMgtSetup."Customer Rewards
Ext. Mgt. Codeunit ID", 'Codeunit does not match default');

    end;

    [Test]

    procedure TestCustomerRewardsWizardTermsPage();
    var
        CustomerRewardsWizardTestPage: TestPage "Customer Rewards Wizard";

    begin
        // [Scenario] Check Terms Page on Wizard
        // [Given] The Customer Rewards Wizard
        Initialize;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;

        // [When] The Wizard is opnened
        CustomerRewardsWizardTestPage.OpenView;

        // [Then] The terms page and fields behave as expected
        Assert.IsFalse(CustomerRewardsWizardTestPage.EnableFeature.AsBoolean, 'Enable feature should be
unchecked');
        Assert.IsFalse(CustomerRewardsWizardTestPage.ActionNext.Visible, 'Next should not be visible');
        Assert.IsFalse(CustomerRewardsWizardTestPage.ActionBack.Visible, 'Back should not be visible');
        Assert.IsFalse(CustomerRewardsWizardTestPage.ActionFinish.Enabled, 'Finish should be disabled');

        CustomerRewardsWizardTestPage.EnableFeature.SetValue(true);

        Assert.IsTrue(CustomerRewardsWizardTestPage.EnableFeature.AsBoolean, 'Enable feature should be
checked');
        Assert.IsTrue(CustomerRewardsWizardTestPage.ActionNext.Visible, 'Next should be visible');
        Assert.IsFalse(CustomerRewardsWizardTestPage.ActionFinish.Enabled, 'Finish should be disabled');

        CustomerRewardsWizardTestPage.Close;
    end;

    [Test]
    procedure TestCustomerRewardsWizardActivationPageErrorsWhenNoActivationCodeEntered();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerRewardsWizardTestPage: TestPage "Customer Rewards Wizard";

    begin
        // [Scenario] Error message when user tries to activate Customer Rewards without activation code.
        // [Given] The Customer Rewards Wizard
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
```

```
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);

        // [When] User invokes activate action without entering activation code
        OpenCustomerRewardsWizardActivationPage(CustomerRewardsWizardTestPage);
        Assert.IsTrue(CustomerRewardsWizardTestPage.ActionBack.Visible, 'Back should be visible');
        Assert.IsFalse(CustomerRewardsWizardTestPage.ActionFinish.Enabled, 'Finish should be disabled');

        // [Then] Error message displayed
        asserterror CustomerRewardsWizardTestPage.ActionActivate.Invoke;
        Assert.AreEqual(GETLASTERRORTEXT, 'Activation code cannot be blank.', 'Invalid error message.');
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
    end;

    [Test]
    procedure TestCustomerRewardsWizardActivationPageErrorsWhenShorterActivationCodeEntered();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerRewardsWizardTestPage: TestPage "Customer Rewards Wizard";

    begin
        // [Scenario] Error message when user tries to activate Customer Rewards with short activation code.
        // [Given] The Customer Rewards Wizard
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);

        // [When] User invokes activate action after entering short activation code
        OpenCustomerRewardsWizardActivationPage(CustomerRewardsWizardTestPage);
        CustomerRewardsWizardTestPage.Activationcode.SetValue('123456');

        // [Then] Error message displayed
        asserterror CustomerRewardsWizardTestPage.ActionActivate.Invoke;
        Assert.AreEqual(GETLASTERRORTEXT, 'Activation code must have 14 digits.', 'Invalid error message.');
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
    end;

    [Test]
    procedure TestCustomerRewardsWizardActivationPageErrorsWhenLongerActivationCodeEntered();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerRewardsWizardTestPage: TestPage "Customer Rewards Wizard";

    begin
        // [Scenario] Error message when user tries to activate Customer Rewards with long activation code.
        // [Given] The Customer Rewards Wizard
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);

        // [When] User invokes activate action after entering long activation code
        OpenCustomerRewardsWizardActivationPage(CustomerRewardsWizardTestPage);
        CustomerRewardsWizardTestPage.Activationcode.SetValue('123456789012345');

        // [Then] Error message displayed
        asserterror CustomerRewardsWizardTestPage.ActionActivate.Invoke;
        Assert.AreEqual(GETLASTERRORTEXT, 'Activation code must have 14 digits.', 'Invalid error message.');
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
    end;

    [Test]
    procedure TestCustomerRewardsWizardActivationPageErrorsWhenInvalidActivationCodeEntered();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
```

```
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerRewardsWizardTestPage: TestPage "Customer Rewards Wizard";

    begin
        // [Scenario] Error message when user tries to activate Customer Rewards with invalid activation
code.
        // [Given] The Customer Rewards Wizard
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
        MockCustomerRewardsExtMgt.MockActivationResponse(false);

        // [When] User invokes activate action after entering invalid but correct length activation code
        OpenCustomerRewardsWizardActivationPage(CustomerRewardsWizardTestPage);
        CustomerRewardsWizardTestPage.Activationcode.SetValue('12345678901234');

        // [Then] Error message displayed
        asserterror CustomerRewardsWizardTestPage.ActionActivate.Invoke;
        Assert.AreEqual(GETLASTERRORTEXT, 'Activation failed. Please check the activtion code you entered.',
'Invalid error message.');
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
    end;

    [Test]
    procedure TestCustomerRewardsWizardActivationPageDoesNotErrorWhenValidActivationCodeEntered();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerRewardsWizardTestPage: TestPage "Customer Rewards Wizard";

    begin
        // [Scenario] Customer Rewards is activated when user enters valid activation code.
        // [Given] The Customer Rewards Wizard
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
        MockCustomerRewardsExtMgt.MockActivationResponse(true);

        // [When] User invokes activate action after entering valid activation code
        OpenCustomerRewardsWizardActivationPage(CustomerRewardsWizardTestPage);
        CustomerRewardsWizardTestPage.Activationcode.SetValue('12345678901234');
        CustomerRewardsWizardTestPage.ActionActivate.Invoke;
        CustomerRewardsWizardTestPage.Close;

        // [Then] Customer Rewards is activated
        Assert.IsTrue(CustomerRewardsExtMgt.IsCustomerRewardsActivated, ActivatedTxt);
    end;

    [Test]
    procedure TestRewardsLevelListPageDoesNotOpenWhenNotActivated();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        RewardLevelListTestPage: TestPage "Rewards Level List";

    begin
        // [Scenario] Error opening Reward Level Page when Customer Rewards is not activated
        // [Given] Unactivated Customer Rewards
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
```

```
        // [When] User opens Reward Level Page
        // [Then] Error message
        asserterror RewardLevelListTestPage.OpenView;
        Assert.AreEqual(GETLASTERRORTEXT, 'Customer Rewards is not activated', 'Invalid error message.');
    end;

    [Test]
    procedure TestRewardsLevelListPageOpensWhenActivated();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        RewardLevelListTestPage: TestPage "Rewards Level List";

    begin
        // [Scenario] Reward Level Page opens when Customer Rewards is activated
        // [Given] Activated Customer Rewards
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
        ActivateCustomerRewards;
        Assert.IsTrue(CustomerRewardsExtMgt.IsCustomerRewardsActivated, ActivatedTxt);

        // [When] User opens Reward Level Page
        // [Then] No error
        RewardLevelListTestPage.OpenView;
    end;

    [Test]
    procedure TestRewardLevelsActionExistsOnCustomerListPage();
    var
        CustomerListTestPage: TestPage "Customer List";

    begin
        // [Scenario] Reward Level action exists on customer list page
        // [Given] Customer List Page

        CustomerListTestPage.OpenView;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;

        // [Then] Reward levels action exists on custome list page
        Assert.IsTrue(CustomerListTestPage."Reward Levels".Visible, 'Reward Levels action should be
visible');
    end;

    [Test]

    [HandlerFunctions('CustomerRewardsWizardModalPageHandler')]

    procedure TestRewardLevelsActionOnCustomerListPageOpensCustomerRewardsWizardWhenNotActivated();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerListTestPage: TestPage "Customer List";

    begin
        // [Scenario] Reward Levels Action Opens Customer Rewards Wizard When Not Activated
        // [Given] Unactivated Customer Rewards
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);

        // [When] User opens Customer List page and invokes action
        CustomerListTestPage.OpenView;
```

```
        CustomerListTestPage."Reward Levels".Invoke;

        // [Then] Wizard opens. Caught by CustomerRewardsWizardModalPageHandler
    end;

    [Test]

    [HandlerFunctions('RewardsLevelListlPageHandler')]
    procedure TestRewardLevelsActionOnCustomerListPageOpensRewardsLevelListPageWhenActivated();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerListTestPage: TestPage "Customer List";

    begin
        // [Scenario] Reward Levels Action Opens Reward Level Page When Activated
        // [Given] Activated Customer Rewards
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        Assert.IsFalse(CustomerRewardsExtMgt.IsCustomerRewardsActivated, NotActivatedTxt);
        ActivateCustomerRewards;
        Assert.IsTrue(CustomerRewardsExtMgt.IsCustomerRewardsActivated, ActivatedTxt);

        // [When] User opens Customer List page and invokes action
        CustomerListTestPage.OpenView;
        CustomerListTestPage."Reward Levels".Invoke;

        // [Then] Wizard opens. Caught by RewardsLevelListlPageHandler
    end;

    [Test]
    procedure TestCustomerCardPageHasRewardsFields();
    var
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] Customer Card Page Has Reward Fields When Opened
        // [Given] Customer Card Page

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;

        // [When] Customer card page is opened
        CustomerCardTestPage.OpenView;

        // [Then] Reward fields are exist
        Assert.IsTrue(CustomerCardTestPage.RewardLevel.Visible, 'Reward Level should be visible');
        Assert.IsFalse(CustomerCardTestPage.RewardLevel.Editable, 'Reward Level should not be editable');
        Assert.IsTrue(CustomerCardTestPage.RewardPoints.Visible, 'Reward Points should be visible');
        Assert.IsFalse(CustomerCardTestPage.RewardPoints.Editable, 'Reward Points should not be editable');
    end;

    [Test]
    procedure TestNewCustomerHasZeroRewardPointsAndNoRewardLevel();
    var
        Customer: Record Customer;
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] A new customer Has Zero Reward Points And No Reward Level
        // [Given] Activated Customer Rewards
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
```

```
        LibraryLowerPermissions.SetO365BusFull;
        ActivateCustomerRewards;

        // [When] New Customer
        LibrarySales.CreateCustomer(Customer);
        CustomerCardTestPage.OpenView;
        CustomerCardTestPage.GoToRecord(Customer);

        // [Then] No Reward level
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, NoLevelTxt);

        // [Then] Reward Point is zero
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 0);
    end;

    [Test]
    procedure TestCustomerHasCorrectRewardPointsAfterPostedSalesOrders();
    var
        Customer: Record Customer;
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] Customer Has Correct Reward Points After 4 Posted Sales Orders
        // [Given] Activated Customer Rewards and Customer
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        ActivateCustomerRewards;

        // New Customer
        LibrarySales.CreateCustomer(Customer);

        // [When] 4 Sales Orders
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");

        // [Then] Customer has 4 reward points
        CustomerCardTestPage.OpenView;
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 4);
    end;

    [Test]
    procedure TestCustomerHasNoRewardLevelAfterPostedSalesOrders();
    var
        Customer: Record Customer;
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] Customer Has 1 Reward Point and No Reward Level After 1 Posted Sales Orders
        // [Scenario] Because Lowest Level requires at least 2 points
        // [Given] Activated Customer Rewards,  Customer, Bronze level for 2 points and above
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        ActivateCustomerRewards;
        AddRewardLevel(BronzeLevelTxt, 2); // 2 points required for BRONZE level

        // New Customer
        LibrarySales.CreateCustomer(Customer);
        CustomerCardTestPage.OpenView;
```

```
        CustomerCardTestPage.GoToRecord(Customer);

        // Verify 0 points and no reward level before sales order
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 0);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, NoLevelTxt);

        // [When] 1 Sales Order
        CreateAndPostSalesOrder(Customer."No.");

        // [Then] Customer has 1 points and no reward level after sales order
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 1);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, NoLevelTxt);
    end;

    [Test]
    procedure TestCustomerHasBronzeRewardLevelAfterPostedSalesOrders();
    var
        Customer: Record Customer;
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] Customer Has 2 Reward Points and Bronze Reward Level After 2 Posted Sales Orders
        // [Scenario] Because Bronze Level requires at least 2 points
        // [Given] Activated Customer Rewards,  Customer, Bronze level for 2 points and above
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        ActivateCustomerRewards;
        AddRewardLevel(BronzeLevelTxt, 2); // 2 points required for BRONZE level

        // New Customer
        LibrarySales.CreateCustomer(Customer);

        // [When] 2 Sales Order
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");

        // [Then] Customer has 2 points and bronze reward level
        CustomerCardTestPage.OpenView;
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 2);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, BronzeLevelTxt);
    end;

    [Test]
    procedure TestCustomerHasSilverRewardLevelAfterPostedSalesOrders();
    var
        Customer: Record Customer;
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] Customer Has 3 Reward Points and Silver Reward Level After 3 Posted Sales Orders
        // [Scenario] Because Silver Level requires at least 3 points
        // [Given] Activated Customer Rewards,  Customer, Bronze level from 2 points, Silver level from 3
points
        Initialize;
        Commit;

        // Using permissions that do not include SUPER
        LibraryLowerPermissions.SetO365BusFull;
        ActivateCustomerRewards;
        AddRewardLevel(BronzeLevelTxt, 2); // 2 points required for BRONZE level
        AddRewardLevel(SilverLevelTxt, 3); // 3 points required for SILVER level
```

```
        // New Customer
        LibrarySales.CreateCustomer(Customer);

        // 2 Sales Order
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");

        // Verify 2 points and bronze reward level
        CustomerCardTestPage.OpenView;
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 2);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, BronzeLevelTxt);

        // [When] 3rd Sales Order
        CreateAndPostSalesOrder(Customer."No.");

        // [Then] Customer has 3 points and silver reward level
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 3);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, SilverLevelTxt);
    end;

    [Test]
    procedure TestCustomerHasGoldRewardLevelAfterPostedSalesOrders();
    var
        Customer: Record Customer;
        CustomerRewardsExtMgt: Codeunit "Customer Rewards Ext. Mgt.";
        CustomerCardTestPage: TestPage "Customer Card";

    begin
        // [Scenario] Customer Has 4 Reward Points and Gold Reward Level After 4 Posted Sales Orders
        // [Scenario] Because Gold Level requires at least 4 points
        // [Given] Activated Customer Rewards,  Customer
        // [Given] Bronze level from 2 points, Silver level from 3 points, Gold level from 4 points
        Initialize;
        Commit;

        // Using permissions that do not inlcude SUPER
        LibraryLowerPermissions.SetO365BusFull;
        ActivateCustomerRewards;
        AddRewardLevel(BronzeLevelTxt, 2); // 2 points required for BRONZE level
        AddRewardLevel(SilverLevelTxt, 3); // 3 points required for SILVER level
        AddRewardLevel(GoldLevelTxt, 4); // 4 points required for GOLD level

        // New Customer
        LibrarySales.CreateCustomer(Customer);

        // 3 Sales Order
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");
        CreateAndPostSalesOrder(Customer."No.");

        // Verify 3 points and silver reward level
        CustomerCardTestPage.OpenView;
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 3);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, SilverLevelTxt);

        // [When] 4th Sales Order
        CreateAndPostSalesOrder(Customer."No.");

        // [Then] Customer has 4 points and gold reward level
        CustomerCardTestPage.GoToRecord(Customer);
        VerifyCustomerRewardPoints(CustomerCardTestPage.RewardPoints.AsInteger, 4);
        VerifyCustomerRewardLevel(CustomerCardTestPage.RewardLevel.Value, GoldLevelTxt);
    end;

    local procedure OpenCustomerRewardsWizardActivationPage(VAR CustomerRewardsWizardTestPage: TestPage
"Customer Rewards Wizard");
```

```
Customer Rewards Wizard');
    begin
        CustomerRewardsWizardTestPage.OpenView;
        CustomerRewardsWizardTestPage.EnableFeature.SetValue(true);
        CustomerRewardsWizardTestPage.ActionNext.Invoke;
    end;

    local procedure Initialize();
    var
        ActivationCodeInfo: Record "Activation Code Information";
        RewardLevel: Record "Reward Level";
        Customer: Record Customer;

    begin
        Customer.ModifyAll(RewardPoints, 0);
        ActivationCodeInfo.DeleteAll;
        RewardLevel.DeleteAll;
        UnbindSubscription(MockCustomerRewardsExtMgt);
        BindSubscription(MockCustomerRewardsExtMgt);
        MockCustomerRewardsExtMgt.Setup;
    end;

    local procedure ActivateCustomerRewards();
    var
        ActivationCodeInfo: Record "Activation Code Information";

    begin
        ActivationCodeInfo.Init;
        ActivationCodeInfo.ActivationCode := '12345678901234';
        ActivationCodeInfo."Date Activated" := Today;
        ActivationCodeInfo."Expiration Date" := CALCDATE('<1Y>', Today);
        ActivationCodeInfo.Insert;
    end;

    local procedure CreateAndPostSalesOrder(SellToCustomerNo: Code[20]);
    var
        SalesHeader: Record "Sales Header";
        SalesLine: Record "Sales Line";
        LibraryRandom: Codeunit "Library - Random";
        SalesOrderTestPage: TestPage "Sales Order";

    begin
        LibrarySales.CreateSalesHeader(SalesHeader, SalesHeader."Document Type"::Order, SellToCustomerNo);
        LibrarySales.CreateSalesLine(SalesLine, SalesHeader, SalesLine.Type::Item, '', 1);
        SalesLine.VALIDATE("Unit Price", LibraryRandom.RandIntInRange(5000, 10000));
        SalesLine.MODIFY(TRUE);
        LibrarySales.PostSalesDocument(SalesHeader, true, true);
    end;

    local procedure AddRewardLevel(Level: Text; MinPoints: Integer);
    var
        RewardLevel: Record "Reward Level";

    begin
        if RewardLevel.Get(Level) then begin
            RewardLevel."Minimum Reward Points" := MinPoints;
            RewardLevel.Modify;
        end else begin
            RewardLevel.Init;
            RewardLevel.Level := Level;
            RewardLevel."Minimum Reward Points" := MinPoints;
            RewardLevel.Insert;
        end;
    end;

    local procedure VerifyCustomerRewardLevel(ExpectedLevel: Text; ActualLevel: Text);
    begin
        Assert.AreEqual(ExpectedLevel, ActualLevel, 'Reward Level should be the same.');
    end;
```

```
    local procedure VerifyCustomerRewardPoints(ExpectedPoints: Integer; ActualPoints: Integer);
    begin
        Assert.AreEqual(ExpectedPoints, ActualPoints, 'Reward Points should be the same.');
    end;

    [ModalPageHandler]
    procedure CustomerRewardsWizardModalPageHandler(var CustomerRewardsWizard: TestPage "Customer Rewards
Wizard");
    begin
    end;

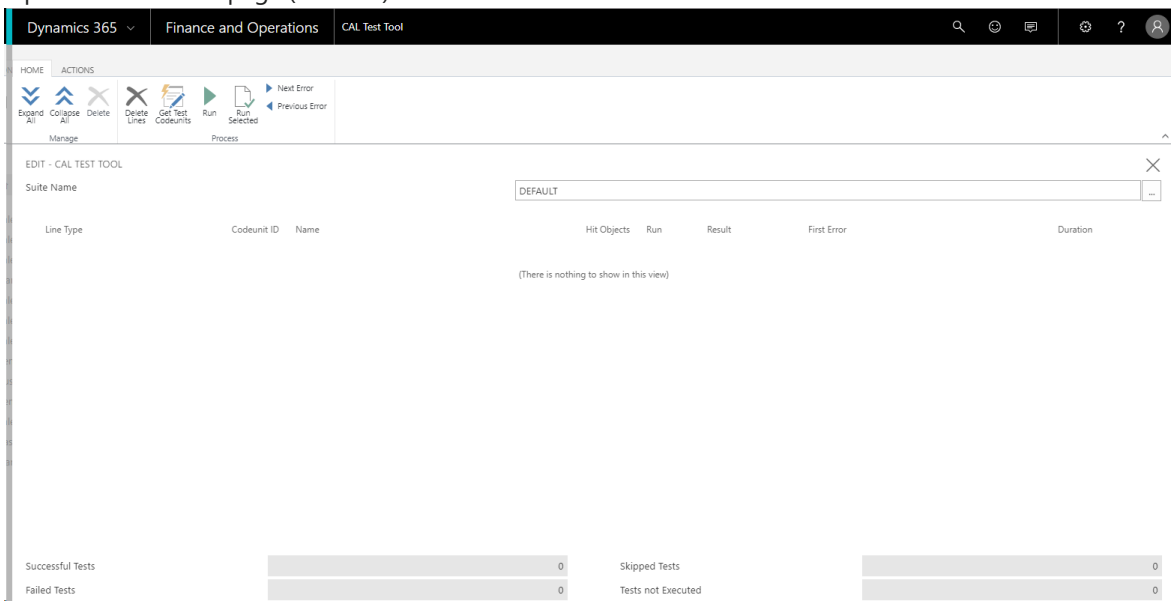
    [PageHandler]
    procedure RewardsLevelListlPageHandler(var RewardsLevelList: TestPage "Rewards Level List");
    begin
    end;
}
```

At this point you can publish and run your tests on your tenant by pressing Ctrl+F5.

Run the tests

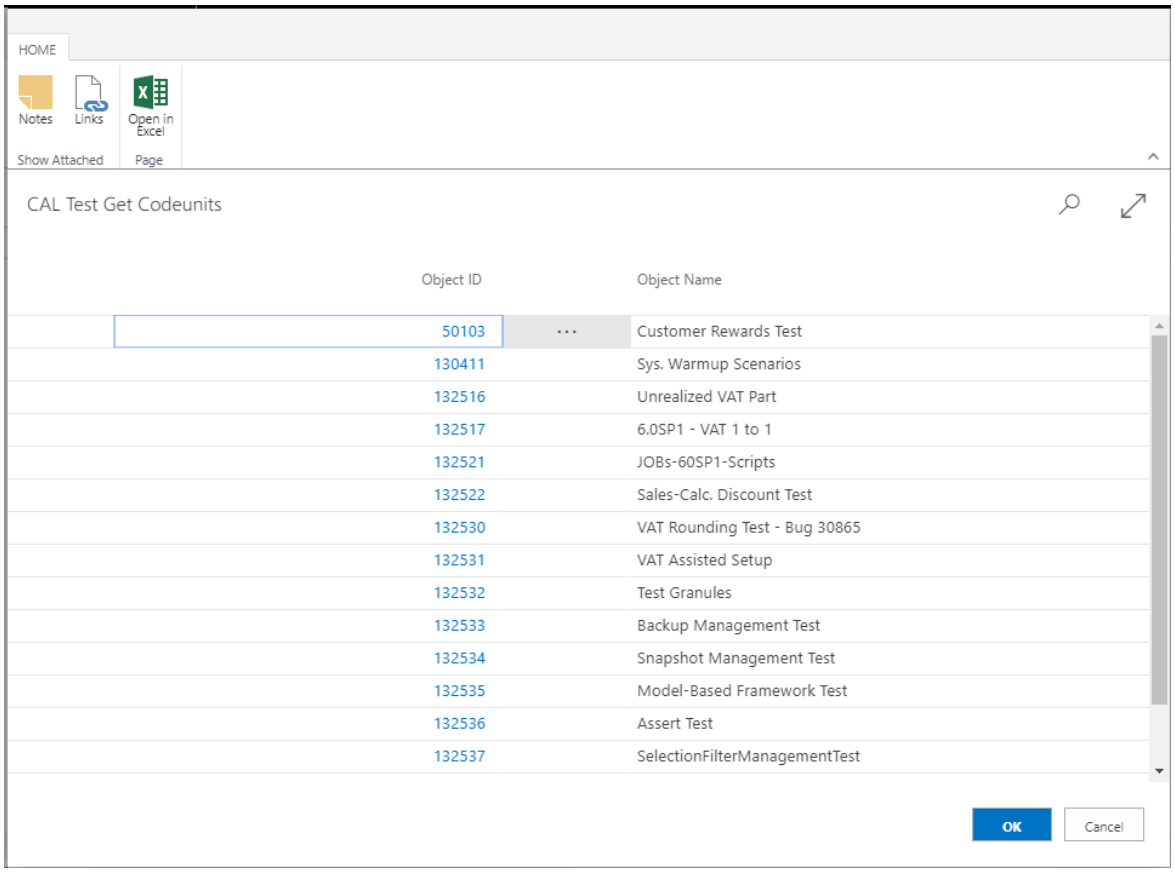
In order to run the tests, follow the steps below.

1. Open the **Test Tool** page (130401).



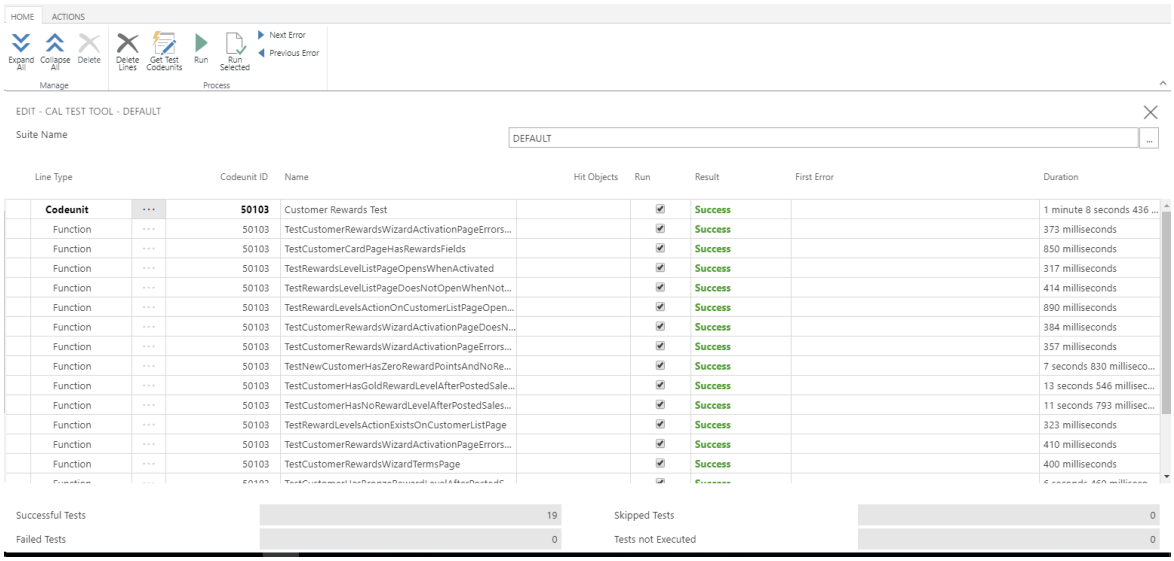
2. Choose **Get Test Codeunits** and then choose **Select Test Codeunits**.

3. Select your test codeunits and then choose the **OK** button.



You can now see all the test methods from your test codeunits.

4. Now, choose **Run** or **Run Selected** to run all the tests in the test codeunit or only the selected tests. The **Result** column indicates whether a test was a SUCCESS or FAILURE. A summary is also presented at the bottom of the page.



Failing Tests

Let us look at what to do if you have a failing test. To create a failing test, we will modify the **SetDefaultCustomerRewardsExtMgtCodeunit** method in codeunit 50100 **Customer Rewards Install Logic** to the following:

```
procedure SetDefaultCustomerRewardsExtMgtCodeunit();
    var
        CustomerRewardsExtMgtSetup: Record "Customer Rewards Mgt. Setup";

    begin
        CustomerRewardsExtMgtSetup.DeleteAll;
        CustomerRewardsExtMgtSetup.Init;
        // Default Customer Rewards Ext. Mgt codeunit to use for handling events
        // Changing
        // CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID" := Codeunit::"Customer
Rewards Ext. Mgt.";
        // To
        CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID" := 0;
        CustomerRewardsExtMgtSetup.Insert;
    end;
```

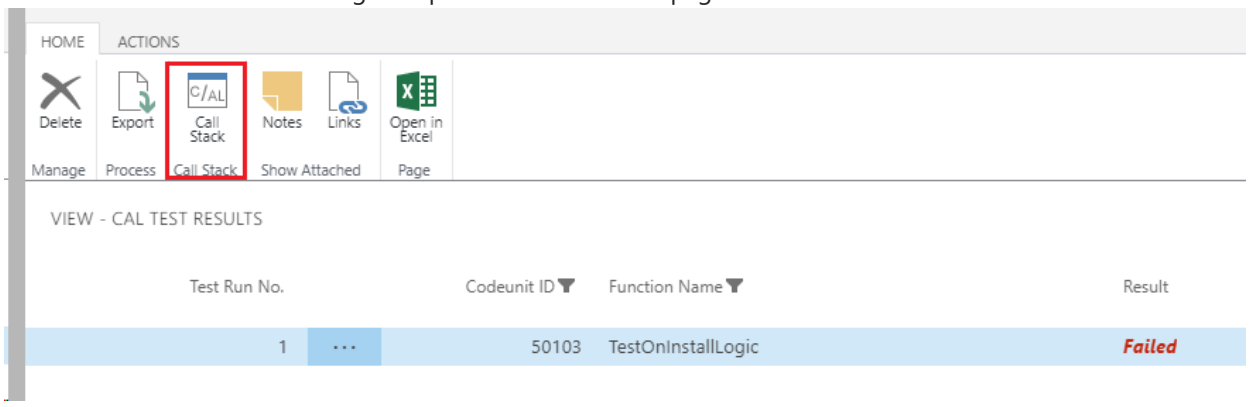
Now, anytime the **SetDefaultCustomerRewardsExtMgtCodeunit** method in the install codeunit is run, the **Customer Rewards Ext. Mgt. Codeunit ID** in the **Customer Rewards Mgt. Setup** table will be set to 0.

Press Ctrl+F5 to publish the updated tests to your tenant and then run them.

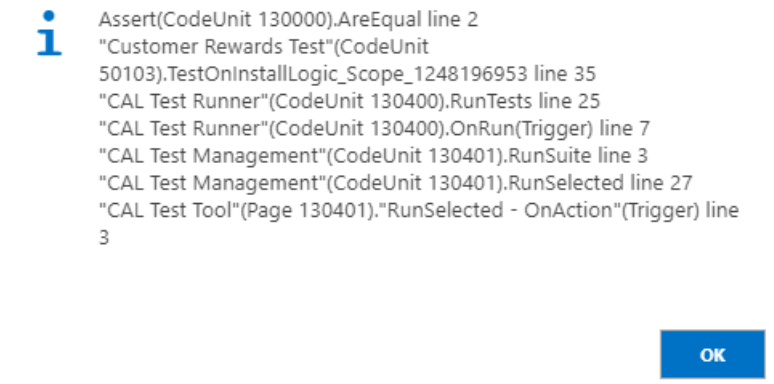
The test TestOnInstallLogic should now have a Failure result with the error message:

```
"Assert.AreEqual failed. Expected:<50101> (Integer). Actual:<0> (Integer). Codeunit does not match default."
```

The error message shows that the actual result in one of our Assert statements differed from what was expected. According to the error message, the Assert statement was expecting a value of 50101 but actually got a value of 0. We can also tell where in our code this is happening because of the message; "Codeunit does not match default", which we defined earlier when we wrote our tests. If we had no idea where the error occurred, we can click on the error message to open the **Test Results** page and then choose the **Call Stack** action.



Choosing the **Call Stack** action will give you a message alert that contains an ordered list of method calls up to the one that caused the error.



The list of method calls is arranged from the most recent at the top to the oldest at the bottom. In our example, we can tell that the `Assert(CodeUnit 130000).AreEqual` (the first on the list) was the last method to be run, indicating where the error was found. Because we did not modify the Assert codeunit, then the wrong values or results must have been passed to it. The next item on the list, `"Customer Rewards Test"(CodeUnit 50103).TestOnInstallLogic_Scope_1248196953` line 35 points to the method that was run before the final one that caused the error. This time, it is in the TestOnInstallLogic method of codeunit 50103 Customer Rewards Test after line 35.

```
35      CustomerRewardsExtMgtSetup.Get;
36      Assert.AreEqual(Codeunit::"Customer Rewards Ext. Mgt.", CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID", 'Codeunit does not match default');
37  end;
```

On line 36 of codeunit 50103 **Customer Rewards Test**, we can see the Assert statement that throws the error. We tested that the result should be `Codeunit::"Customer Rewards Ext. Mgt."` which is 50101, when our install logic is run, however, the result of the test indicated that we got a result of 0. This implies that our install logic is not working as expected. To fix this, we need to examine all the previous lines of code in the method to figure out where we went wrong. This will lead us to line 31, where the **SetDefaultCustomerRewardsExtMgtCodeunit** method call is made.

```
31      CustomerRewardsInstallLogic.SetDefaultCustomerRewardsExtMgtCodeunit;
```

When you go into the **SetDefaultCustomerRewardsExtMgtCodeunit** method, codeunit 50100 **Customer Rewards Install Logic**, you will see the change we made to cause the test to fail. Revert it so that `CustomerRewardsExtMgtSetup."Customer Rewards Ext. Mgt. Codeunit ID"` now stores `Codeunit::"Customer Rewards Ext. Mgt."`, instead of 0. Publish the updated extension and tests to your tenant and run the tests again. The test **TestOnInstallLogic** should pass now because the actual result matches what is expected.

Conclusion

At this point, the Customer Rewards sample extension can be published and installed on your sandbox.

See Also

Developing Extensions
Getting Started with AL
How to: Publish and Install an Extension
Converting Extensions V1 to Extensions V2

AL Development Environment

2/17/2021 • 2 minutes to read • Edit Online

This section describes all of the objects that are available with the AL Language development environment for Dynamics 365 Business Central.

> **TIP**
> If you are looking for the C/SIDE documentation, visit our Dynamics NAV library.

Defining the AL data model

TO	SEE
Learn about how to define new table objects for your extension.	Table Object
Learn about how to modify and extend existing table objects.	Table Extension Object

Presenting the AL data

TO	SEE
Learn about how to create new page objects for your extension.	Page Object
Learn about how to modify and extend existing page objects.	Page Extension Object
Learn about how to create page customization objects.	Page Customization Object
Learn about how to create profile objects.	Profile Object
Learn about how to create report objects.	Report Object
Learn about how to create xmlport objects.	XmlPort Object
Learn about how to create query objects.	Query Object
Learn about how to create control add-in objects.	Control Add-In Object

Writing AL code

TO	SEE
Learn about writing codeunits for your extension.	Codeunit Object
Get an overview of methods in AL grouped by the data type that they support.	Data Types and Methods in AL

TO	SEE
Get an overview of properties in AL grouped by the objects that they support.	Properties Overview
Get an overview of triggers in AL grouped by the objects that they support.	Triggers Overview

API for HTTP, JSON, TextBuilder, and XML

For information about the HTTP, JSON, TextBuilder, and XML classes, see HTTP, JSON, TextBuilder, and XML API Overview.

See Also

Developing Extensions
Getting Started with AL
FAQ for Developing in AL
AL Language Extension Configuration

Programming in AL

2/17/2021 • 2 minutes to read • Edit Online

AL is the programming language that is used for manipulating data (such as retrieving, inserting, and modifying records) in a Dynamics 365 Business Central database, and controlling the execution of the various application objects, such as pages, reports, or codeunits.

With AL, you can create business rules to ensure that the data which is stored in the database is meaningful and consistent with the way customers do business. Through AL programming, you can:

- Add new data or transfer data from one table to another, for example, from a journal table to a ledger table.
- Combine data from multiple tables into one report or display it on one page.

Where to write AL code

Almost every object in Dynamics 365 Business Central contains triggers where you can add your AL code. Triggers exist for the following objects:

- Tables and table extensions
- Table fields
- Pages and page extensions
- Reports
- Data items
- XMLports
- Queries

You can initiate the execution of your AL code from the following:

- Actions
- Any object that has an instantiation of the object that contains AL code. An example of an instantiation is a variable declaration.

> **NOTE**
> If the AL code is in a `local` method, then you cannot run it from another object.

Variable declarations

Variables in AL are declared using the `var` keyword, and the syntax looks like this:

```
var
        myInt: Integer;
```

If you have multiple variables of the same type, these can be declared in one line, such as:

```
var
        myInt, nextInt, thirdInt : Integer;
        isValid, doCheck : Boolean;
```

The `protected` keyword can be used to make variables accessible between tables and table extensions and between pages and page extensions. For more information, see Protected Variables.

Guidelines for placing AL code

We recommend the following guidelines for AL code:

- In general, write the code in codeunits instead of on the object on which it operates. This promotes a clean design and provides the ability to reuse code. It also helps enforce security. For example, typically users do not have direct access to tables that contain sensitive data, such as the **General Ledger Entry** table, nor do they have permission to modify objects. If you put the code that operates on the general ledger in a codeunit, give the codeunit access to the table, and give the user permission to run the codeunit, then you will not compromise the security of the table and the user will be able to access the table.
- If you must put code on an object instead of in a codeunit, then put the code as close as possible to the object on which it operates. For example, put code that modifies records in the triggers of the table fields.

Reusing code

Reusing code makes developing applications both faster and easier. More importantly, if you organize your AL code as suggested, your applications will be less prone to errors. By centralizing the code, you will not unintentionally create inconsistencies by performing the same calculation in many places, for example, in several triggers that have the same table field as their source expression. If you have to change the code, you could either forget about some of these triggers or make a mistake when you modify one of them.

See Also

Simple Statements
Control Statements
Methods
System-Defined Variables
Developing Extensions
Getting Started with AL

AL Simple Statements

2/17/2021 • 5 minutes to read • Edit Online

AL simple statements are single-line statements that are executed sequentially and do not alter the flow of execution of code. This article explains some of the simple statements in AL.

Assignment statements

Assignment statements assign a value to a variable. The value that you assign to the variable is an AL expression. It can be a constant or a variable, or it can consist of multiple elements of AL expressions. If you use a method call as the value to assign to a variable in an assignment statement, then the value that is assigned is the return value of the method.

You use the ":=" operator for assignment statements.

Example

The following example assigns a constant integer value to an integer variable that you have defined.

```
Count := 1;
```

Example

The following example assigns a value that consists of a constant, an operator, and a variable.

```
Amount := 2 * Price;
```

Example

The following example assigns the return value of the Open Method (File) to a Boolean variable that you have defined.

> **NOTE**
> This method is supported only in Business Central on-premises.

```
OK := TestFile.Open('C:\temp\simple.xml');
```

The return value of the `Open` method is optional. If you do not handle the return value in your code, then a run-time error occurs when a method returns **false**. The following example causes a run-time error if the file `C:\temp\simple.xml` cannot be opened.

```
TestFile.Open('C:\temp\simple.xml');
```

You can handle the return value by using an if-then statement.

```
if TestFile.Open('C:\temp\simple.xml') then begin
  // continue running
else
  Error(Text001);
```

Method statements

You use method statements to execute either built-in system methods or user-defined (custom) methods. Method calls may include parameters, which are passed to the method. For more information, see Calling Methods.

AssertError statements

You use AssertError statements in test methods to test how your application behaves under failing conditions. The AssertError keyword specifies that an error is expected at run time in the statement that follows the AssertError keyword.

If a simple or compound statement that follows the AssertError keyword causes an error, then execution successfully continues to the next statement in the test method. You can get the error text of the statement by using the GetLastErrorText method.

If a statement that follows the AssertError keyword does not cause an error, then the AssertError statement causes the following error and the test method that is running produces a FAILURE result:

```
TestAsserterrorFail: FAILURE

An error was expected inside an AssertError statement.
```

Example

To create a test method to test the result of a failure of a `CheckDate` method that you have defined, you can use the following code. This example requires that you create a method called `CheckDate` to check whether the date is valid for the customized application.

```
InvalidDate := 19000101D;
InvalidDateErrorMessage := Text001;
AssertError CheckDate(InvalidDate);

IF GetLastErrorText <> InvalidDateErrorMessage then
  Error('Unexpected error: %1', GetLastErrorText);
```

This example requires the following variables.

```
var
    InvalidDate : Date;
    InvalidDateErrorMessage : Text;
    Text001 : Label 'The date is outside the valid date range.';
```

With statements (to be deprecated)

IMPORTANT

Using the `with` statement is being deprecated with Dynamics 365 Business Central 2020, release wave 2. With this release it is a warning, which will become an error in a future release.
Using `with` statements introduces possible uniqueness collisions when multiple extensions contribute to the same objects because it allows working with members using just simple names instead of qualifying them. To avoid this going forward, we are marking the use of `with`, be it implicit or explicit as warnings. With this release, you can use a quick action to fix these files, as well as suppress obsolete warnings for now. Code that contains `with` statements will, however, need to be refactored before `with` statements are compiled with errors. For more information, see Deprecating Explicit and Implicit With Statements. For information about using directives in code, see Directives in AL and Pragma ImplicitWith Directive in AL.

The following syntax shows a with-do statement.

```
with <Record> do
  <Statement>
```

When you work with records, addressing is created as record name, dot (period), and field name:

<Record>.<Field>

If you work continuously with the same record, then you can use `with` statements. When you use a `with` statement, you can only specify the record name one time.

Within the scope of *<Statement>*, fields in *<Record>* can be addressed without having to specify the record name.

You can nest several `with` statements. If you have identical names, then the inner `with` statement overrules the outer `with` statement.

Example

This example shows two ways to write the same code that creates a record variable that you can commit later.

```
CustomerRec."No." := '1234';
CustomerRec.Name := 'Windy City Solutions';
CustomerRec."Phone No." := '555-444-333';
CustomerRec.Address := '1241 Druid Avenue';
CustomerRec.City := 'Windy City';
Message('A variable has been created for this customer.');
```

This example requires the following variables.

```
var
    CustomerRec : Record Customer;
```

The following example shows another way to create a record variable that you can commit later:

```
with CustomerRec do begin
  "No." := '1234';
  Name := 'Windy City Solutions';
  "Phone No." := '555-444-333';
  Address := '1241 Druid Avenue';
  City := 'Windy City';
  Message('A variable has been created for this customer.');
end;
```

Programming conventions

Within `with-do` blocks, do not repeat the name of the object by using the member variable or method.

If you nest a `with-do` block within another explicit or implicit `with-do` block, then the `with-do` block that you create within another `with-do` block must always be attached to a variable of the same type as the variable that is attached to the surrounding `with-do` block. Otherwise, it can be difficult to see what variable that a member variable or method refers to. For example, implicit `with-do` blocks occur in table objects and in pages that have been attached to a record.

Example

The following example demonstrates nested `with-do` blocks. Both `with-do` blocks are attached to a Customer Ledger Entry record variable.

```
with CustLedgEntry do begin
  Insert;
  ...;
  with CustLedgEntry2 do begin
    Insert;
    ...;
  end;
end;
```

Incorrect example

The following example demonstrates incorrect code in which you cannot directly tell which record variable that the MyField field refers to.

```
with CustLedgEntry do begin
  ...;
  with VendLedgEntry do begin
    MyField := <Some Value>;
    ...;
  end;
end;
```

See Also

Control Statements
Methods
Directives in AL
AL Essential Methods

AL Control Statements

2/17/2021 • 13 minutes to read • Edit Online

AL code consists of one or more statements, which are executed sequentially in top-down order. However, you will often need to control the direct top-down flow of the execution. One or more statements may have to be repeated more than once, or you may have to make the execution of a certain statement conditional. To do this, you use control structures.

The control structures in AL are divided into the following main groups, as described in this article:

- AL Compound Statements
- AL Conditional Statements
- AL Repetitive Statements

> **NOTE**
> In the following sections conventions for how to structure and align AL code are presented to introduce best practices. In many cases the structure is not necessary to get the code to compile, but rather to improve readability.

AL compound statements

In some cases, the AL syntax only lets you use a single statement. However, if you have to execute more than one simple statement, the statements can be written as a compound statement by enclosing the statements between the `begin` and `end` keywords.

```
begin
    <Statement 1>;
    <Statement 2>;
    ..
    <Statement n>;
end;
```

The individual statements are separated by a semicolon. In AL, a semicolon is used to separate statements and not, as in other programming languages, as a terminator symbol for a statement. Nevertheless, an extra semicolon before an end does not cause an error because it is interpreted by the compiler as an empty statement.

Blocks

The begin-end structure is also called a *block*. Blocks can be very useful to refer to the other control structures in AL.

When begin follows then, else, or do, it should be on the same line and preceded by one space character.

Example

```
if (x = y) and (a = b) then begin
    x := a;
    y := b;
end;
```

Example

```
if (xxx = yyyyyyyyyy) and
    (aaaaaaaaaa = bbb)
then begin
    x := a;
    x := y;
    a := y;
end else begin
    y := x;
    y := a;
end;
```

AL conditional statements

You use conditional statements to specify a condition and one or more commands to execute if the condition is evaluated as true or false. There are two types of conditional statements in AL:

- if-then-else, where there are two choices
- case, where there are more than two choices

If-then else statements

if-then-else statements have the following syntax.

```
if <Condition> then
    <Statement1>
[else
    <Statement2>]
```

If `<Condition>` is true, then `<Statement1>` is executed. If `<Condition>` is false, then `<Statement2>` is executed.

The square brackets around else `<Statement2>` mean that this part of the statement is optional. The else statement is used when different actions are executed depending on how `<Condition>` is evaluated.

You can build more complex control structures by nesting if-then-else statements. The following example is a typical if-then-else statement.

```
if <Condition1> then
    if <Condition2> then
        <Statement1>
    else
        <Statement2>
```

If `<Condition1>` is false, then nothing is executed. If `<Condition1>` and `<Condition2>` are both true, then `<Statement1>` is executed. If `<Condition1>` is true and `<Condition2>` is false, then `<Statement2>` is executed.

> **NOTE**
> A semicolon in front of an else statement is not allowed.

Reading several nested if-then-else statements can be very confusing but generally, an else statement belongs to the last if statement that lacks an else statement.

Programming conventions

- If and then should be on the same line. else should be on a separate line.
- If there are many or long expressions, then should be on a new line and be aligned with if.

- When you write if expressions with then and else parts, write them so that the then result is more probable than the else one.
- If the last statement in the then part of an if-then-else statement is an exit or an error, do not continue with an else statement.

Example

```
if x = y then
    x := x + 1
else
    x := -x - 1;
```

Example

```
if (xxxxxxxxxx = yyy) and
    (aaa = bbbbbbbbbb)
then
    x := a
else
    y := b;
```

Example

```
if x <> y then
    exit(true);
x := x * 2;
y := y - 1;
```

Incorrect example

```
if x < y then
    exit(true)
else begin
    x := x * 2;
    y := y - 1;
end;
```

Example

The following example shows an if-then statement without the optional else statement.

```
if Amount < 1000 then
    Total := Total + Amount;
```

Example

The following example shows a nested if-then-else statement.

```
...
if Amount < 1000 then begin
    if I > J then
        Max := I
    else
        Max := J;
    Amount := Amount * Max;
    end;
else
...
```

Case statements

Case statements have the following syntax.

```
case <Expression> of
    <Value set 1>:
        <Statement 1>;
    <Value set 2>:
        <Statement 2>;

    <Value set n>:
        <Statement n>;
[else
    <Statement n+1>]
end;
```

In this definition, `<Expression>` cannot be a record and `<Value set>` must be an expression or a range.

Case statements are also called multiple option statements and are typically used when you must choose between more than two different actions. The method of the case statement is as follows:

- The `<Expression>` is evaluated, and the first matching value set executes the associated statement, if there is one.
- If no value set matches the value of the expression and the optional else part has been omitted, then no action is taken. If the optional else part is used, then the associated statement is executed.

The data type of the value sets must be the same as the data type of `<Expression>` or at least be convertible to the same data type.

In most cases, the data type of the value sets are converted to the date type of the evaluated expression. The only exception is if the evaluated expression is a Code variable. If the evaluated expression is a Code variable, then the value sets are not converted to the Code data type.

> **NOTE**
> This type conversion can cause an overflow at run time if the resulting data type cannot hold the values of the datasets.

For more information about Code variables, see Code Data Type.

Programming conventions

When you use a case statement, indent the value sets by four character spaces. If you have two or more value sets on the same line, then separate them by commas without spaces. The last value set on a line is immediately followed by a colon without a preceding space. The action starts on the line after the value set and is further indented by four character spaces. If there is a begin, then it should be put on a separate line unless it follows else. If a begin follows an else, then it should be on the same line as else.

If there are more than two alternatives, use a case statement. Otherwise, use an if-then-else statement.

Example

```
case Field of
    Field::A:
        begin
            x := x + 1;
            y := -y - 1;
        end;
    Field::B:
        x := y;
    Field::C,Field::D:
        y := x;
    else begin
        y := x;
        a := b;
    end;
end;
```

Example

The following AL code prints various messages depending on the value of *Number*. If the value of *Number* does not match any of the entries in the case structure, then the else entry is used as the default.

```
case Number of
    1,2,9:
        message('1, 2, or 9.');
    10..100:
        message('In the range from 10 to 100.');
else
    message('Neither 1, 2, 9, nor in the range from 10 to 100.');
end;
```

Example

The following AL code shows how value sets in a case statement are evaluated if the expression is a Code data type.

```
MyCode := 'ABC';
case MyCode of
    'abc':
        message('This message is not displayed.');
    'def':
        message('This message is not displayed.');
else
    message('The value set does not match the expression.');
end;
```

This example requires that you create the following code data type variable.

```
var
    MyCode : Code[10];
```

The value set 'abc' is not converted because the evaluated expression MyCode is a code variable.

AL repetitive statements

A repetitive statement is also known as a loop. The following table shows the looping mechanisms in AL.

LOOPING MECHANISM	DESCRIPTION

LOOPING MECHANISM	DESCRIPTION
for	Repeats the inner statement until a counter variable equals the maximum or minimum value specified.
foreach	Repeats the inner statement for each statement in a List, XmlNodeList, XmlAttributeCollection, or JsonArray.
while	Repeats the inner statement as long as the specified condition is **true**. The statement in a loop of this kind is repeated 0 or more times.
repeat	Repeats the inner statements until the specified conditions evaluate to **true**. The statements in a loop of this kind are always executed at least one time.

For-to and for-downto control structure

The following syntax shows the for-to and for-downto statement.

```
for <Control Variable> := <Start Number> to <End Number> do
    <Statement>
```

```
for <Control Variable> := <Start Number> downto <End Number> do
    <Statement>
```

The data type of `<Control Variable>`, `<Start Number>`, and `<End Number>` must be Boolean, number, time, or date.

Use for-to and for-downto statements when you want to execute code a specific number of times. The *<Control Variable>* controls the number of times that the code of the inner statement is executed according to the following:

- In a for-to loop statement, the `<Control Variable>` value is increased by one after each iteration. The inner `<Statement>` is executed repeatedly until the `*<Start Number>*` value is greater than the `*<End Number>*` value.
- In a for-downto loop statement, the `<Control Variable>` value is decreased by one after each iteration. The inner `<Statement>` is executed repeatedly until the `<Start Number>` value is less than the `<End Number>` value.

> **NOTE**
> When the for statement is executed, `<Start Number>` and `<End Number>` are converted to the same data type as `<Control Variable>` if it is required. This type conversion can cause a run-time error.

> **NOTE**
> If the value of the `<Control Variable>` is changed inside the for loop, then the behavior is not predictable. Furthermore, the value of the `<Control Variable>` is undefined outside the scope of the for loop.

Example 1

The following code initiates a for loop that uses the integer control variable named Count.

```
for Count := 1000 to 100000000000000 do
```

This example requires the following Integer data type variable.

```
var
    Count : Integer;
```

When this statement is executed, a run-time error occurs because the start and end values are converted to the same data type as the Count control variable. Count has been declared as an integer variable. The end number 100000000000000 is outside the valid range for integers, and an error occurs.

Example 2

The following example shows how to nest for statements.

Set the Dimensions property of variable A to 5;7.

The following for statements could be used to initialize every element in a 5x7 array with the value 23.

```
for I := 1 to 5 do
    for J := 1 to 7 do
        A[I,J] := 23;
```

This example requires the following Integer data type variables.

```
var
    I : Integer;
    J : Integer;
```

Foreach control structure

You can use the foreach statement to iterate through List, XmlNodeList, XmlAttributeCollection, and JsonArray expressions. The foreach statement has the following syntax.

```
foreach <Element> in <List> do
    <Statement>
```

The `<List>` variable must be of the List, XmlNodeList, XmlAttributeCollection, or JsonArray type. The `<Element>` variable must be a data type that is compatible with elements specified by the `<List>`.

The following code example iterates through a list of customer names and returns each customer name in a message.

```
procedure PrintCustomerNames(customerNames : List of [Text]);
var
    customerName : Text;
begin
    foreach customerName in customerNames do
        message(customerName);
end;
```

While-do control structure

The following syntax shows the while-do statement.

```
while <Condition> do
    <Statement>
```

If `<Condition>` is true, then `<Statement>` is executed repeatedly until `<Condition>` becomes false. If `<Condition>` is false from the start, then `<*Statement>` is never executed.

The while do statement can be used when some code should be repeated as long as an expression is true.

Programming conventions

When there is only one condition, put while and do on the same line. Put the statements on separate lines and indented by two spaces.

When there are multiple conditions, put the conditions on separate lines and indented by two spaces and put do on a separate line that is aligned with while.

Example

```
while <expr> do
    <Statement>;
```

Example

```
while <expr> do begin
    <Statement>;
    <Statement>;
end;
```

Example

```
while <expr> and
      <expr> and
      <expr>
do begin
    <Statement>;
    <Statement>;
end;
```

Example

The following AL code increases the variable I until it equals 1000 and displays a message when it is finished.

```
while I < 1000 do
    I := I + 1;
message(format(I));
```

This example requires the following integer data type variable.

```
var
    I : integer
```

Repeat-until control structure

The following syntax shows the repeat-until statement.

```
repeat
    <Statements> until <Condition>
```

`<Statements>` is executed repeatedly until `<Condition>` is true.

The repeat until control structure resembles the while control structure. The difference is that because the repeat until statement is executed from left to right, the `<Statements>` is always executed at least one time, regardless of what the `<Condition>` is evaluated to. This contrasts with the while control structure, which performs the evaluation before the `<Statement>` is executed. In the while control structure, if the first evaluation of `<Condition>` returns false, then no statements are executed.

Programming conventions

Always put repeat on a separate line.

Example

```
if x < y then begin
    repeat
        x := x + 1;
        a := a - 1;
    until x = y;
        b := x;
end;
```

Example

This code uses a repeat-until loop to count the number of entries in the Customer table.

```
Count := 0;
if Customer.find('-') then
repeat
    Count := Count + 1;
until Customer.next <= 0;
message('The Customer table contains %1 records.',Count);
```

This example requires the following variables.

```
var
    Count : Integer;
    Customer : Record Customer;
```

The `find` method finds the first entry in the table. Each time NEXT is called, it steps one record forward. When NEXT equals 0, there are no more entries in the table. The loop is exited, and a message displays how many entries were found.

Exit statement

The exit statement is used to control the flow of the execution. The following syntax shows an exit statement.

```
exit([<Value>])
```

An exit statement is used to interrupt the execution of a AL trigger. The interruption occurs even when the code is executed inside a loop or a similar structure. The exit statement is also used when a local method should return a value.

Using exit without a parameter in a local method corresponds to using the parameter value 0. The AL method will return the value 0 or '' (empty string).

A compile-time error occurs if exit is called by using a return parameter from either of the following:

- System-defined triggers.

- Local methods that do not return a value.

Example

The following example shows the use of the exit statement in a local method. Assume that the if statement is used to detect an error. If the error condition is met, then execution is stopped and the local method returns the error code 1.

```
for I := 1 to 1000 do begin
    if Amount[I] < Total[I] then
        exit(1);
    A[I] := Amount[I] + Total[I];
end;
```

Break statement

You use the break statement to terminate the iterative statement in which it appears.

```
break;
```

You typically use the break statement in the repeating statements such as for, while, or repeat to stop an iteration or loop when certain conditions are met.

> **NOTE**
> The break statement is different than the Break Method (Report, XMLport). Although both stop an iteration or loop, the break method will also terminate the trigger in which it is run.

Example

The following AL code increases the variable I by one for each iteration, and terminates the iteration when I equals 10.

```
while Count < 1000 do
    begin
    Count := Count + 1;
    message(FORMAT(Count));
    if Count = 10 then
        break;
    end;
end;
```

This example requires the following integer data type variable.

```
var
    I : integer
```

See Also

Programming in AL
AL Simple Statements
Directives in AL
AL Essential Methods

AL methods

2/17/2021 • 5 minutes to read • Edit Online

Like other languages, AL methods are a fundamental programming element. A method, also known as a procedure, is a named group of statements that perform an operation or task. Depending on the scope, methods can be run, or *called*, from the same object in which they are declared or from other parts of the application.

There are two types of methods: system methods (built-in) and user-defined (custom) methods.

- Built-in methods are part of the platform. Built-in methods can be used for different purposes, such as string handling, text formatting, database handling, and so on. For information about the available built-in methods, see AL method Reference and Essential AL methods.
- Custom methods are specialized methods for your application to bind the objects, such as tables, pages, and codeunits, together to form a unified whole. You can create special methods for use anywhere in the database.

Declaring methods

The method declaration defines the method and has the following syntax:

```
[Attributes(arguments list)]
local procedure <method_name>(parameter list) <return_value_name> : <data_type>[<length>]
```

Snippet support

Typing the shortcut `tprocedure` will create the basic structure for a method when using the AL Language extension in Visual Studio Code.

Attributes (optional)

An attribute is a modifier on a method declaration that specifies information that controls the method's use and behavior. Adding an attribute on a method declaration is also known as *decorating* a method. For example, decorating a method with the Integration attribute sets the method to be an event publisher. An attribute can have one or more arguments that set properties for the method instance.

Attributes are placed before the method. For information about the available attributes, see Method Attributes.

Local and global scope

A method can be a *local* method or *global* method. A local method can only be accessed or called from inside the object in which it is declared. A global method can be called from inside the object in which it is declared and from other objects.

To declare a local method, start the declaration with `local`:

```
local procedure Mymethod();
```

To declare a global method, *omit* `local`:

```
procedure Mymethod();
```

Parameters (optional)

A parameter is one or more variables or expressions that are sent to the method through the method call. The parameter provides information to the method, and the method can modify that information. In the method declaration, you place the parameters in parentheses `()`. If there is more than one parameter, the parameters are separated by semicolons. A parameter is defined by a data type. Some data types, such as `Record`, require an additional subtype.

For example, the following method declaration includes two parameters: `MyCustomer` and `MyDimension`:

```
procedure  Mymethod(MyCustomer : Record Customer;var MyDimension : ARRAY [2] OF Boolean)
```

Return values (optional)

A method can return data that can be then coded against. A return value is defined by a name (optional), data type, and optional length depending on the data type.

For example, if the return value is a Text DataType, the text might have a length of 50.

```
    procedure MyMethod() ReturnValue: Text[50]
        var result : Text[50];
    begin
        // do something important where result is calculated
        ReturnValue := result;
    end;
```

Calling methods

You can run, or call, a built-in or a custom method by using its name in a method call statement. When a method is called the current application sequence is suspended and the code on the method is run. When the method code is completed, the application code sequence returns to where the method was called from. How the method is called determines what happens when it returns.

A method can be used as part of an expression. For example, the following code uses a method named `CalculatePrice` as an expression:

```
TotalCost := Quantity * CalculatePrice;
```

In this case, the `CalculatePrice` method must return a value that is used in evaluating the expression. This return value is then multiplied by the Quantity variable and that result is assigned to the TotalCost variable.

A method can also be run by using a method call statement. This statement only calls the method and does not return any value. The following is an example of calling a method named `MyRunMethod`:

```
if Quantity > 5 then
MyRunMethod;
```

The `MyRunMethod` returns no data back to the calling code.

Parameters

In a method call, the parameters are separated by commas, and the optional parameters may be omitted starting from the right. For example, this means that if a method has three optional parameters, then you cannot omit the second parameter without omitting the third parameter.

You can specify that a parameter is passed to a method by value or by reference.

- If a parameter is passed by value, then a copy of the variable is passed to the method. Any changes that the method makes to the value of the variable are local changes that affect only the copy, not the variable itself.
- If a parameter is passed by reference, then a reference to the variable is passed to the method. The method can change the value of the variable itself.

Example 1

The following shows the syntax for a method. The first example shows a method with two mandatory parameters.

```
method(Parameter1, Parameter2)
```

Some built-in methods have optional parameters, the syntax is shown below. The optional parameters may be omitted starting from the right.

```
method([Optional1] [, Optional2] [, Optional3])
```

The method that uses the syntax above can be called by using the following code.

```
method(Optional1, Optional2)
```

Example 2

ABS is an example of an AL method that has a fixed number of parameters (1).

```
Value := -1033; //A negative integer value
PositiveValue := ABS(Value); //Calculate the positive value 1033
```

Example 3

The method `DMY2DATE` is an example of a method that can be called by using a variable number of parameters.

```
NewDate := DMY2DATE(5, 11, 1992); //Returns the date November 5, 1992
```

Depending on the use of the `DMY2DATE` method, one, two, or three parameters can be passed to the method because the second and third parameters are optional. When the second and third parameters are not used, values from the system date are used as default values.

Example 4

You can assign the return value of a method to a variable.

```
ReturnVal := MyMethod(Param1);
```

Example 5

In this example, `MyMethod` returns a Boolean value. You can use the return value in a conditional statement.

```
if (MyMethod(Param1)) then
  <Statement1>
else
  <Statement2>
```

See Also

Development Overview
AL Methods
AL Simple Statements
AL Control Statements

Preprocessor Directives in AL

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

In AL, like in other programming languages, preprocessor directives can be used to make code conditional, to suppress warnings, or to enable expand and collapse in code. Preprocessor directives can be divided into the following groups. For more information about each type, use the links provided below.

- Conditional directives
- Regions
- Pragmas

Any code can be made conditional, including table fields, and checked using a conditional directive. To check code using a conditional directive, you must define a symbol to check. A symbol returns a boolean value; `true` or `false`. Symbols can be defined at the beginning of a source file and the scope of the specific symbol is the file that it is defined within. You can also define symbols in the `app.json` file, and then the scope is global for the extension.

> **NOTE**
> Built-in symbols are currently not supported in AL. Symbols must be defined in a specific file or in the `app.json` file.

Conditional directives

The following conditional preprocessor directives are supported in AL.

CONDITIONAL PREPROCESSOR DIRECTIVE	DESCRIPTION
#if	Specifies the beginning of a conditional clause. The `#endif` clause ends it. Compiles the code between the directives if the specified symbol being checked is defined.
#else	Specifies a compound conditional clause. If none of the preceding clauses evaluates to `true`, the compiler will evaluate code between `#else` and `#endif`.
#elif	Combines `else` and `if`. If `#elif` is `true` the compiler evaluates all code between `#elif` and the next conditional directive.
#endif	Specifies the end of a conditional clause that begins with `#if`.
#define	Defines a symbol that can be used to specify conditions for a compilation. For example, `#define DEBUG`. The scope of the symbol is the file that it was defined in.
#undef	Undefines a symbol.

Defining symbols

Symbols can be defined globally in the `app.json` file. A symbol can also be defined using the `#define` directive in code, but if symbols are defined in the `app.json` file, they can be used globally. The following example defines `DEBUG` as a global symbol. This can then be used from code as illustrated in the Conditional code example below. A symbol has a boolean value, that means it evaluates to `true` or `false`.

```
"preprocessorSymbols": [ "DEBUG" ]
```

For more information, see JSON Files.

Example

```
#if DEBUG
    trigger OnOpenPage()
    begin
        Message('Only in debug versions');
    end;
#endif
```

See Also

Development in AL
AL Development Environment
Conditional directives
Region Directive in AL
Pragma Directive in AL
Deprecating Explicit and Implicit With Statements
Best Practices for Deprecation of Code in the Base App

Region Directive in AL

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Region

The `#region` directive is used to mark a block of code that you can expand or collapse. This can, for example, be useful for larger files for better readability or for focusing on code that you are currently working on. The `#endregion` specifies the end of a `#region` block of code.

Syntax

```
#region code
```

```
#endregion
```

Remarks

A `#region` block must be terminated with a `#endregion` directive.

A `#region` block cannot overlap with an `#if` block. However, a `#region` block can be nested in an `#if` block, and an `#if` block can be nested in a `#region` block.

Example

In this example the `#region` directive makes a code block that is up for refactoring collapsible.

```
#region Ugly code - let's not look at this
    procedure UglyCode()
    begin
        // No one should look at this
    end;
#endregion
```

See Also

Development in AL
AL Development Environment
Pragma Directive in AL
Conditional directives
Deprecating Explicit and Implicit With Statements

Pragma Directive in AL

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Pragma

The `#pragma` directive gives the compiler special instructions for the compilation of the file in which it appears. The `#pragma` directive has a number of actions that can be used with the pragma instructions below, these are `disable`, `restore`, and `enable`.

AL supports the following pragma instructions:

- Pragma ImplicitWith
- Pragma Warning

See Also

Development in AL
AL Development Environment
Region Directive in AL
Conditional directives
Deprecating Explicit and Implicit With Statements

Using Access Modifiers in AL

2/17/2021 • 2 minutes to read • Edit Online

Access modifiers are used to set accessibility of tables, table fields, codeunits, and queries, which controls whether the object can be used from other code in your module or other modules. Access modifiers in AL are designed to create solid APIs, by limiting the symbols that dependant modules can take a reference on. Limiting the API surface can hide implementation details and allow for later refactoring of code without breaking external code.

You set the object accessibility by using the Access Property. If the `Access` property is not specified; default is `Public`.

> **NOTE**
> In AL access modifiers are primarily intended for designing APIs and *cannot* be used as a security boundary.

Access modifiers

The access modifiers that are available in AL are:

ACCESS MODIFIER	DESCRIPTION
`internal`	The object or field can be accessed only by code in the same module, but not from another module. **Note:** This accessibility level is controlled by the `internalsVisibleTo` setting. For more information, see JSON Files
`local`	The field can be accessed only by code in the same table or table extension where the field is defined. **Note:** Applies to tables fields only.
`protected`	The field can be accessed only by code in the same table or tableextensions of that table. **Note:** Applies to table fields only.
`public`	The object or field can be accessed by any other code in the same module and in other modules that references it. **Note:** This is the default value.

Setting access to `internal` is linked to the JSON Files setting `internalsVisibleTo`.

> **IMPORTANT**
> Access modifiers are only taken into consideration at compile time. For example, at compile time, a table with `Access = Internal` cannot be used from other modules that do not have access to the internals of the module where the table is defined, but at runtime, any module can access the table by using reflection-based mechanisms such as `RecordRef`, or `TransferFields`. And the `OnRun` trigger can be run on `internal` codeunits by using `Codeunit.Run`. Setting the object accessibility level as `Access = Internal;` *cannot* be used as a security boundary. Also see JSON Files.

See Also

AL Development Environment
Access Property

XML Comments in Code

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

In Dynamics 365 Business Central, you can add documentation to your code by including XML elements in special comment fields directly in the source code before the block of code that the comment refers to. The documentation comment must immediately precede a user-defined type that it annotates, for example a codeunit, table, or interface, or a member such as a field or method. The syntax for adding XML comments in your code is triple slashes `///` followed by one of the supported XML tags. There is IntelliSense support for writing documentation comments. Most importantly providing a template documentation comment when writing the third slash in the triple slash.

Documentation comments are visible when hovering over source symbols, in completion lists, and in signature help. By adding XML comments in code, you can improve readability, add useful information about the implementation, and help others take over code that you wrote. With XML comments you also enable IntelliSense in Visual Studio Code on the AL objects that you add in your code as a help to other developers, working with or extending your code. This means that when you have built an extension and someone extends this code, they will get inline documentation when they call the given object.

> **NOTE**
> Integration with documentation generator tools like DocFx and SandCastle is currently not supported.

> **NOTE**
> If you have the `showMyCode` setting in the app.json file set to `false` and download an app package; the app package will not contain any XML comments.

Supported XML tags

The following table lists the XML elements that are supported for AL.

TOP-LEVEL XML TAG	DESCRIPTION	SYNTAX
`<summary>`	A summary of the object	`<summary>description</summary>`
`<param>`	Used in method declarations to describe one or more parameters defined in the method. For each parameter, specify the name and a description.	`<param name="name">description</param>`
`<returns>`	Used in method declarations to describe the return value of the method.	`<returns>description</returns>`
`<example>`	Used to specify an example of how to use a given codeunit or object.	`<example>description</example>`

TOP-LEVEL XML TAG	DESCRIPTION	SYNTAX
`<remarks>`	Used to supplement information given in the `<summary>` section.	`<remarks>description</remarks>`

FORMATTING XML TAG	DESCRIPTION	SYNTAX
`<paramref>`	Specifies a reference to a parameter in a `<summary>` or `<remarks>` block.	`<paramref name="name"/>`
`<para>`	Allows structuring text inside a `<summary>`, `<remarks>`, or `<returns>` tag.	`<para>paragraph</para>`
`<b>`	Allows formatting text as **bold** inside a top-level tag.	`<b>bold</b>`
`<i>`	Allows formatting text as *italic* inside a top-level tag.	`<i>italic</i>`
`<c>`	Specifies that text within a description should be marked as code inside a top-level tag.	`<c>inline code</c>`
`<code>`	Specifies that multiline text within a description should be marked as code inside a top-level tag.	`<code>code block</code>`
`<list>`	Specifies a list formatted as a bulleted or numbered list, or as a table in a `<summary>` or `<remarks>` block.	`<list type="bullet|number|table">`. See full List syntax below.

List syntax

```
<list type="bullet|number|table">
    <listheader>
        <term>term</term>
        <description>description</description>
    </listheader>
    <item>
        <term>term</term>
        <description>description</description>
    </item>
</list>
```

Example

The following example is taken from the *Email.Codeunit.al* file in the System Application. In this example, the parameter `EmailMessageId` is documented using the `<param>` syntax.

```
/// <summary>
/// Provides functionality to create and send e-mails.
/// </summary>

codeunit 8901 "Email"
{
    Access = Public;

    /// <summary>
    /// Enqueues an email in the outbox to be sent in the background.
    /// </summary>
    /// <param name="EmailMessageId">The ID of the email to enqueue</param>
    procedure Enqueue(EmailMessageId: Guid)
    begin
        EmailImpl.Enqueue(EmailMessageId);
    end;
...
```

Special symbols

For special symbols, such as angle brackets, to appear in text of a documentation comment, use the HTML encoding of `<` and `>` which is `<` and `>` respectively. The following example illustrates how.

```
/// <summary>
/// This property always returns a value &lt; 1.
/// </summary>
```

See Also

AL Development Environment
Developing Extensions in AL
Pages Overview

FAQ for Developing in AL

2/17/2021 • 2 minutes to read • Edit Online

This topic contains a number of frequently asked questions and answers to these questions.

How do I get started?

For an overview of developing apps for Dynamics 365 Business Central, see aka.ms/GetStartedWithApps

Next, follow the Getting Started with AL to set up the tools.

Which version of the AL Language extension should I use?

For Dynamics 365 Business Central cloud sandboxes you must use the AL Language extension available in the Visual Studio Code Marketplace.

For the latest Developer Preview releases you must use the AL Language extension that is available in the *next major* artifact through the "Ready! for Dynamics 365 Business Central" program on Microsoft Collaborate.

How do I enable the debugger?

To read about enabling debugging in AL, see here Debugging. To read about snapshot debugging, see Snapshot Debugging.

Can I create something similar to Menusuites?

In the AL Language extension, the concept of Menusuites is not supported. The two primary purposes of Menusuites are:

- Making pages searchable
- Making pages accessible through a navigation structure

The first purpose can be achieved in Extensions by using the new properties added to Pages and Reports. For more information, see Adding Pages and Reports to Search.

The second purpose can be achieved by extending the Navigation Pane page and/or by adding Actions to other existing pages that can serve as a navigation starting point. For more information, see Adding Menus to the Navigation Pane.

How do I upgrade Extensions V1 to Extensions V2?

For information on upgrading, see the following topics: Upgrading Extensions v2 and Converting from Extensions v1 to Extensions v2.

File APIs are not available in Extensions V2. What do I do?

Code that relies on temporary files must be rewritten to rely on `InStream` and `OutStream` types. Code that relies on permanent files must be rewritten to use another form of permanent storage.

DotNet types are not available in Extensions V2. What now?

For cloud solutions .NET interop is not available due to safety issues in running arbitrary .NET code on cloud

servers.

With the AL Language extension, you can find AL types that replace the most typical usages of .NET like HTTP, JSON, XML, StringBuilder, Dictionaries, and Lists. Many .NET usages can be replaced directly by the AL types resulting in much cleaner code. For more information, see HTTP, JSON, TextBuilder, and XML API Overview.

For things that are not possible to achieve in AL code, the recommendation is to use Azure Functions to host the DLL or C# code previously embedded and call that service from AL.

Extensions published from Visual Studio Code or created using Designer have disappeared from a sandbox environment. Why?

Extensions that have been published to a sandbox environment from Visual Studio Code or created using Designer are removed when the sandbox environment is updated or relocated within our service. However, the data of an app is not removed, so you only have to re-publish and install the app to make it available.

For more information, see Sandbox Environments.

See Also

Getting Started with AL
Keyboard Shortcuts
AL Development Environment

Using the Code Analysis Tool

2/17/2021 • 3 minutes to read • Edit Online

This topic shows how you can use static code analysis tool on an AL project from within Visual Studio Code.

Enabling code analysis

First, follow the steps below to create a simple project in AL.

1. Press **Alt + A, Alt + L** to create a new project.
2. Open the Command Palette **Ctrl+Shift+P** and choose either **User Settings** or **Workspace Settings**.
3. Copy the setting `al.enableCodeAnalysis` to the settings file and set it to `true`: `"al.enableCodeAnalysis": true`.
4. Copy the setting `al.codeanalyzers` to the settings file and then use **Ctrl+Space** to pick from the available code analyzers. Separate the list of code analyzers with commas. For more information about the available analyzers, see AppSourceCop, CodeCop, PerTenantExtensionCop, and UICop.

At this point, the selected analyzers will be run on your project. Next, add some code to the project that will, in the following example, be used to demonstrate a violation of the AA0001 **"There must be exactly one space character on each side of a binary operator such as := + - AND OR =."** code analysis rule.

> **NOTE**
> By default, code analysis is run in the background.

Adding your own code to the project

In the Visual Studio Code Explorer, open the `HelloWorld.al` file and replace the existing code with the following:

```
pageextension 50100 CustomerListExt extends "Customer List"
{
    trigger OnOpenPage();
    var
        result: Integer;
    begin
        // The following line will trigger the warning
        // AA0001 "There must be exactly one space character on each side
        // of a binary operator such as := + - AND OR =."
        result := 2+2;
        Message('2 + 2 = ' + Format(result));
    end;
}
```

Viewing the results of the code analysis

The code analysis tools will run in the background. You will see the faulty expression underlined and the warning **"There must be exactly one space character on each side of '+'."** will be displayed if you mouse over the underlined code. You can also view the list of issues by selecting the **View** tab of Visual Studio Code and choosing the **Problems** option.

Using the **Ctrl+Shift+B** shortcut to build your project will run the code analysis tools on the entire project and

the detected issues will be displayed in the **Output** window of Visual Studio Code. For more information about AL keyboard shortcuts, see Keyboard shortcuts.

Code analyzers

A code analyzer is a library that builds on the compiler's functionality to offer enhanced analysis of the syntax and semantics of your code at build time. The AL Language extension for Visual Studio Code contains four analyzers:

- **CodeCop** is an analyzer that enforces the official AL Coding Guidelines. For more information about the CodeCop rules, see CodeCop Analyzer Rules.
- **PerTenantExtensionCop** is an analyzer that enforces rules that must be respected by extensions meant to be installed for individual tenants. For more information about the PerTenantExtensionCop rules, see PerTenantExtensionCop Analyzer Rules.
- **AppSourceCop** is an analyzer that enforces rules that must be respected by extensions meant to be published to Microsoft AppSource. For more information about the AppSourceCop rules, see AppSourceCop Analyzer Rules.
- **UICop** is an analyzer that enforces rules that must be respected by extensions meant to customize the Web Client. For more information about the UserInterfaceCop rules, see UICop Analyzer Rules.

Enabling code analysis on large projects

In order to improve performance when running code analysis on large projects, you can switch off running code analysis in the background. To do so, open the Command Palette **Ctrl+Shift+P** and choose either **User Settings** or **Workspace Settings**. Then, specify the setting `"al.backgroundCodeAnalysis": false`.

See Also

Using the Code Analysis Tools with the Ruleset
Ruleset for the Code Analysis Tool
Development in AL
Directives in AL
Debugging in AL
AL Language Extension Configuration

Ruleset for the Code Analysis Tool

2/17/2021 • 2 minutes to read • Edit Online

In an AL project, you can use a custom ruleset file to specify how code analysis will report the issues it encounters. Different settings can affect how rules are applied and each ruleset file name must follow the pattern `<name>.ruleset.json` to benefit from IntelliSense in Visual Studio Code.

> **NOTE**
> Use the `truleset` and `trule` snippets provided by the AL Language extension to create your ruleset.

The following table describes the schema of a ruleset object:

SETTING	MANDATORY	TYPE	VALUE
name	Yes	String	The name of the ruleset.
description	No	String	The description of the ruleset. You can use this to document the purpose of the ruleset.
generalAction	No	Error | Warning | Info | Hidden	The action to apply to all the diagnostics that have rules defined in this file or in other files that have a **Default** action specified and to all the diagnostics generated by the current set of analyzers that do not have a rule defined. If an included file has a stricter **generalAction**, that one will be used.
includedRuleSets	No	Array of **IncludedRuleSet**	List of external ruleset files to include in the current ruleset. The order in which the files are processed is undefined.
rules	No	Array of **Rule**	Collection of rules to apply to diagnostics generated by analyzers.

An **IncludedRuleSet** is a complex JSON object that defines the inclusion of an external ruleset file in the current ruleset, and has the following properties:

SETTING	MANDATORY	TYPE	VALUE

SETTING	MANDATORY	TYPE	VALUE
path	Yes	String	The path to the included file. For includes specified in the file to which the **al.ruleSetPath** is set, the path can be absolute or relative to the project folder. For files included from the root ruleset file, the path is relative to the file. Adding an entry to the ruleset path and saving it will automatically be applied to all projects that are using the ruleset.
action	Yes	Error | Warning | Info | Hidden | None | Default	The action to apply for all the diagnostics that have an action specified in the included ruleset that is different from **None** and **Hidden**.

A **Rule** is a complex JSON object that specifies how you can process a specific diagnostic. A **Rule** object has the following properties:

SETTING	MANDATORY	TYPE	VALUE
id	Yes	String	The string that uniquely identifies a diagnostic.
action	Yes	Error | Warning | Info | Hidden | None	The action to apply if the diagnostic is emitted. There cannot be two rules with the same id and different actions in the same rule file.

Examples

The following example shows a ruleset that sets the severity of rule **AA0001 : There must be exactly one space character on each side of a binary operator such as := + - AND OR =.** provided by the **CodeCop** analyzer to **Error**.

```
{
    "name": "Company ruleset",
    "description": "These rules must be respected by all the AL code written within the company.",
    "rules": [
        {
            "id": "AA0001",
            "action": "Error",
            "justification": "This diagnostic helps to improve readability. It must be respected in all
cases."
        }
    ]
}
```

The following example shows a project-specific ruleset that extends a company-wide ruleset contained in the file **company.ruleset.json** and sets the severity of the rule **AA0005 : Only use BEGIN..END to enclose**

compound statements. provided by the **CodeCop** analyzer to **Info**.

```
{
    "name": "Personal Project ruleset",
    "description": "A list of project specific rules",
    "includedRuleSets": [
        {
            "action": "Default",
            "path": "./company.ruleset.json"
        }
    ],
    "rules": [
        {
            "id": "AA0005",
            "action": "Info",
            "justification": "For this specific project, this diagnostic should be informational."
        }
    ]
}
```

See Also

Using the Code Analysis Tools
Using the Code Analysis Tools with the ruleset
AL Development Environment
Directives in AL
AL Language Extension Configuration

Using the Code Analysis Tools with the Ruleset

2/17/2021 • 3 minutes to read • Edit Online

This topic shows how you can use a custom ruleset to customize the severity of diagnostics produced by the code analysis tools that are part of the AL Language extension for Visual Studio Code.

Using rulesets with code analysis

First, create a simple project in AL.

1. Press **Alt + A, Alt + L** to create a new project.
2. Open the Command Palette by using the **Ctrl+Shift+P** shortcut and choose either **User Settings** or **Workspace Settings**.
3. Copy the setting `al.enableCodeAnalysis` to the settings.json file and set it to `true`: `"al.enableCodeAnalysis": true`.
4. Copy the setting `al.codeanalyzers` to the settings file and then use **Ctrl+Space** to pick from the available code analyzers. Separate the list of code analyzers with commas. For more information about the available analyzers, see AppSourceCop, CodeCop, PerTenantExtensionCop, and UICop.

At this point, the selected analyzers will be run on your project. Next, add some code to the project that will, in the following example, be used to demonstrate violations of the AA0001 **"There must be exactly one space character on each side of a binary operator such as := + - AND OR =."** code analysis rule.

Add your own code to the project

In the Visual Studio Code Explorer, open the `HelloWorld.al` file and replace the existing code with the following:

```
pageextension 50100 CustomerListExt extends "Customer List"
{
    trigger OnOpenPage();
    var
        result: Integer;
    begin
        // The following line will trigger the warning
        // AA0001 "There must be exactly one space character on each side
        // of a binary operator such as := + - AND OR =."
        result := 2+2;
        Message('2 + 2 = ' + Format(result));
    end;
}
```

On the **View** tab of Visual Studio Code, select the **Problems** option and you will see a warning with the message **"There must be exactly one space character on each side of '+'."**. In this case, the problem can be fixed by running the AL Formatter command. For more information, see AL Formatter.

Creating and customizing a ruleset

To create and customize a ruleset of your own, follow the next steps:

1. On the **File** tab in Visual Studio Code, choose **New File**.
2. Save the empty file with a name, for example `<name>.ruleset.json` and make a note of the file path.

3. Add the following code to the `<name>.ruleset.json` file:

```
{
    "name": "My Custom ruleset",
    "rules": [
        {
            "id": "AA0001",
            "action": "None"
        }
    ]
}
```

4. In your project settings set **al.ruleSetPath** to the path to the `<name>.ruleset.json` file, relative to the project root. For more information about custom rules, see Ruleset for the Code Analysis Tool.

> **NOTE**
>
> Use the `truleset` and `trule` snippets provided by the AL Language extension to create your ruleset. The ruleset will be applied to all the analyzers enabled for the current project. For more information about selectively enabling analyzers, see Using the Code Analysis Tools.

Running the code analysis

The code analysis will run in the background and you will see the warning **"There must be exactly one space character on each side of '+'."** disappear from the **Problems** option in Visual Studio Code.

To trigger a new compilation manually, use the **Ctrl+Shift+B** shortcut to build your project. For more information about AL keyboard shortcuts, see Keyboard shortcuts.

Limitations

Changing the contents of the ruleset file will not be detected by the AL Language extension. To see the effects of changing the ruleset file, you can try any of the following:

- Reload the window.
- In the project settings, change the **al.ruleSetPath** setting to an invalid path. Save the settings file, change back the setting, and save it.

See also

Ruleset for the Code Analysis Tool
Using the Code Analysis Tools
Directives in AL
Development in AL
Debugging in AL
AL Language Extension Configuration

AppSourceCop Analyzer Rules

2/17/2021 • 9 minutes to read • Edit Online

AppSourceCop is an analyzer that enforces rules that must be respected by extensions meant to be published to Microsoft AppSource.

Rules

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0001	Tables and table extensions that have been published must not be deleted.	Upgrade	Error
AS0002	Fields must not be deleted.	Upgrade	Error
AS0003	The previous version of the extension could not be found.	Upgrade	Error
AS0004	Fields must not change type, since dependent extensions may break	Upgrade	Error
AS0005	Fields must not change name	Upgrade	Error
AS0006	Tables that have been published must not change name.	Upgrade	Error
AS0009	Key fields must not be changed	Upgrade	Error
AS0010	Keys must not be deleted, since dependent extensions may break	Upgrade	Error
AS0011	An affix is required	Extensibility	Error
AS0013	The field identifier must be within the allowed range	Extensibility	Error
AS0014	The project manifest must contain the allocated identifier range	Extensibility	Error
AS0015	TranslationFile must be enabled.	Extensibility	Error

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0016	Fields of field class 'Normal' must use the DataClassification property and its value should be different from ToBeClassified	Extensibility	Error
AS0018	A procedure belonging to the public API cannot be removed	Upgrade	Error
AS0019	Event attributes cannot be removed	Upgrade	Error
AS0020	The type of events cannot be changed.	Upgrade	Error
AS0021	An argument in an event attribute cannot be changed to false.	Upgrade	Error
AS0022	An external scope cannot be removed	Upgrade	Error
AS0023	A return type cannot be modified in external procedures	Upgrade	Error
AS0024	Parameters cannot be removed or added in external procedures	Upgrade	Error
AS0025	Parameters cannot be modified, renamed, or removed from events.	Upgrade	Error
AS0026	The type and subtype of parameters cannot be modified in events and external procedures	Upgrade	Error
AS0027	Modifying the array size of a parameter in events and external procedures is not allowed	Upgrade	Error
AS0028	Reducing the array size of a parameter in events and external procedures is not allowed	Upgrade	Error
AS0029	Pages and PageExtensions that have been published must not be deleted, since dependent extensions may break	Upgrade	Error

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0030	Pages that have been published must not be renamed.	Upgrade	Error
AS0031	Actions that have been published must not be deleted.	Upgrade	Error
AS0032	Controls that have been published must not be deleted/	Upgrade	Error
AS0033	Views that have been published must not be deleted.	Upgrade	Error
AS0034	Unsupported table property change	Upgrade	Error
AS0036	Unsupported table field property change	Upgrade	Error
AS0038	Unsupported table key property change	Upgrade	Error
AS0039	Removing properties that cause destructive changes is not allowed	Upgrade	Error
AS0041	Table field property changes that cause destructive changes must not be removed	Upgrade	Error
AS0042	Table key property changes that cause destructive changes must not be removed	Upgrade	Error
AS0043	Unique keys must not be deleted	Upgrade	Error
AS0044	Property changes that cause destructive changes are not allowed	Upgrade	Error
AS0047	The extension name is too long.	Extensibility	Error
AS0048	The publisher name is too long.	Extensibility	Error

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0049	The access modifier of an application object cannot be changed to a value that provides less access	Extensibility	Error
AS0050	The extensibility of an application object cannot be removed	Extensibility	Error
AS0051	Manifest property is required for AppSource submission	Extensibility	Error
AS0052	The property 'url' must be set to a valid URL	Extensibility	Error
AS0053	The compilation target of an application must be a value that is allowed in a multi-tenant SaaS environment	Extensibility	Error
AS0054	The AppSourceCop configuration must specify the set of affixes used by the application	Configuration	Error
AS0055	The AppSourceCop configuration must specify the list of countries targeted by the application	Configuration	Hidden
AS0056	The country codes specified in the 'supportedCountries' property must be valid ISO 3166-1 alpha-2 codes	Configuration	Warning
AS0057	Translations must be provided for all the locales in which the application will be available	Extensibility	Hidden
AS0058	Only use AssertError in Test Codeunits	Extensibility	Error
AS0059	Reserved database tables are read-only in a multi-tenant environment	Extensibility	Error
AS0060	Unsafe methods cannot be invoked in an AppSource application	Extensibility	Error
AS0061	Procedures must not subscribe to CompanyOpen events	Extensibility	Error

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0062	Page controls and actions must use the ApplicationArea property	Extensibility	Error
AS0063	Removing a var modifier in events is not allowed	Upgrade	Error
AS0064	Interface implementations that have been published must not be deleted.	Upgrade	Error
AS0065	Interfaces that have been published must not be deleted.	Upgrade	Error
AS0066	A new method to an interface that has been published must not be added.	Upgrade	Error
AS0067	Adding an interface to an enum that has been published must have a default implementation.	Upgrade	Error
AS0068	Changing a table extension's target is not allowed.	Upgrade	Error
AS0069	An enum field replacing an option field should have at least the same number of members.	Upgrade	Error
AS0070	An enum field replacing an option field should preserve the member names.	Upgrade	Error
AS0071	An enum field replacing an option field should preserve the member ordinal values.	Upgrade	Error
AS0072	The ObsoleteTag property and the Tag in the Obsolete attribute must be set to the next release version.	Design	Hidden
AS0073	Obsolete Tag must be set.	Design	Hidden
AS0074	The Obsolete Tag must be the same across branches.	Design	Hidden
AS0075	Obsolete Reason must be set.	Design	Warning

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0076	Obsolete Tag format.	Design	Hidden
AS0077	Adding a var modifier in events is not allowed	Upgrade	Warning
AS0078	Adding or removing a var modifier in external procedures is not allowed	Upgrade	Warning
AS0079	An affix is required for procedures defined in extension objects.	Extensibility	Warning
AS0080	Fields must not decrease in length	Upgrade	Error
AS0081	InternalsVisibleTo should not be used as a security feature.	Extensibility	Warning
AS0082	It is not allowed to rename an enum value.	Upgrade	Error
AS0083	It is not allowed to delete a value from an enum.	Upgrade	Error
AS0084	The ID range assigned to the extension must be within the allowed range	Extensibility	Error
AS0085	The 'application' property must be used instead of explicit dependencies	Extensibility	Warning
AS0086	Fields must not increase in length	Upgrade	Warning
AS0087	Translations of enum value captions must not contain commas	Extensibility	Warning
AS0088	Objects with an ID that can be referenced and which have been published must not be deleted.	Upgrade	Error
AS0089	Objects that can be referenced and which have been published must not be deleted.	Upgrade	Error
AS0090	Objects that can be referenced and which have been published must not be renamed.	Upgrade	Error

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AS0091	One or more dependencies of the previous version of the extension could not be found.	Upgrade	Error
AS0092	The 'applicationInsightsKey' property must specify the AAD instrumentation key.	Configuration	Warning

NOTE

Several rules enforced by the AppSourceCop analyzer are incompatible with rules enforced by the PerTenantExtensionCop. Make sure to enable only one of these at a time.

NOTE

Failing to comply with the rules whose default severity is set to `Error` will fail the submission of your extension to the AppSource marketplace. It is recommended, but not mandatory to comply with the rules whose severity is marked as `Warning` or `Info`.

Configuration

The AppSourceCop analyzer can be further configured by adding a file named `AppSourceCop.json` in the project's root folder. The AL Language extension will offer IntelliSense for this file.

The following table describes the settings in the `AppSourceCop.json` file:

SETTING	MANDATORY	VALUE
name	No	The name of a previous version of this package with which you want to compare the current package for breaking changes.
publisher	No	The publisher of a previous version of this package with which you want to compare the current package for breaking changes.
version	Yes	The version of a previous version of this package with which you want to compare the current package for breaking changes.
mandatoryAffixes	No	Affixes that must be prepended or appended to the name of all new application objects, extension objects, and fields.
supportedCountries	No	The set of country codes, in the alpha-2 ISO 3166 format, in which the application will be available.

SETTING	MANDATORY	VALUE
targetVersion	No	Specifies the next Major.Minor version of the extension in the current branch in order to validate the ObsoleteTag values with AS0072. This is only relevant when the default obsoleteTagPattern '(\d+)\.(\d+)' is used. This property is being deprecated in favor of obsoleteTagVersion.
obsoleteTagVersion	No	Specifies the next Major.Minor version of the extension in the current branch in order to validate the ObsoleteTag values with AS0072. This is only relevant when the default obsoleteTagPattern '(\d+)\.(\d+)' is used.
obsoleteTagPattern	No	The Obsolete tag pattern used by AS0076. This should be a valid regular expression. By default, the pattern '(\d+)\.(\d+)' is used.
obsoleteTagPatternDescription	No	A human-readable description for the ObsoleteTagPattern regular expression. This is used in diagnostics reported by AS0076. By default, 'Major.Minor' is used.
baselinePackageCachePath	No	The path to the folder containing the baseline and its dependencies with which you want to compare the current package for breaking changes. By default, the package cache path for the current project is used (see 'al.packageCachePath' setting).

The `name`, `publisher`, `version` properties are used for specifying a previous version of the current package. This package must be located in the baseline package cache folder of your extension. This cache can be specified using the `baselinePackageCachePath` property. If this property is not specified, the dependency package cache path of the extension will be used instead. The `al.packageCachePath` setting allows you to specify the path to the folder that will act as the cache for the dependencies symbol files used by your project. AppSourceCop will compare the previous version of your extension with its current version and will report any breaking changes introduced by the current package.

The `mandatoryAffixes` property specifies strings that must be prepended or appended to the names of all new objects, extension objects and fields. By using these affixes, you can prevent clashes between objects added by your extension and objects added by other extensions.

The `supportedCountries` property specifies the codes that correspond to the countries for which the product allows AppSource submissions. For more information, see Availability and supported Countries/Regions and Translations

The properties `obsoleteTagVersion`, `obsoleteTagPattern`, and `obsoleteTagPatternDescription` can be used to enable additional validation on object obsoletion. These are not required for AppSource submissions.

Example

In the following example, we will configure AppSourceCop to validate that all new elements have a name that contains one of the specified affixes.

> **NOTE**
> Make sure that code analysis is enabled and `${AppSourceCop}` is specified in the list of enabled code analyzers. For more information see AL Language Extension Configuration.

We start by creating the default "Hello world" extension.

```
pageextension 50100 CustomerListExt extends "Customer List"
{
    trigger OnOpenPage();
    begin
        begin
            Message('App published: Hello world');
        end;
    end;
}
```

We continue by adding the configuration file `AppSourceCop.json` in the project's root folder and setting its content to the following.

```
{
    "mandatoryAffixes": [ "Foo", "Bar" ]
}
```

> **IMPORTANT**
> If you are running a multi-root workspace environment, you must have one `AppSourceCop.json` file in the root folder of each of the projects. For more information, see Working with multiple AL project folders within one workspace.

You are immediately greeted by the following error message:

```
AS0011: The identifier 'CustomerListExt' must have at least one of the mandatory affixes 'Foo, Bar'.
```

Prepending **Foo** to the name of the page extension object will fix this error and prevent clashes between this page extension and page extensions added by other developers.

> **NOTE**
> It is still possible to use the `mandatoryPrefix` and `mandatorySuffix` properties in the `AppSourceCop.json`. For more information see AS0011.

See Also

Using the Code Analysis Tool
Ruleset for the Code Analysis Tool
Using the Code Analysis Tools with the Ruleset

CodeCop Analyzer Rules

2/17/2021 • 5 minutes to read • Edit Online

CodeCop is an analyzer that enforces the official AL Coding Guidelines.

Rules

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AA0001	There must be exactly one space character on each side of a binary operator such as := + - AND OR =.	Readability	Warning
AA0002	There must be no space character.	Readability	Warning
AA0003	There must be exactly one space character between the NOT operator and its argument.	Readability	Warning
AA0005	Only use BEGIN..END to enclose compound statements.	Readability	Warning
AA0008	Function calls should have parenthesis even if they do not have any parameters.	Readability	Warning
AA0013	When BEGIN follows THEN, ELSE, DO, it should be on the same line, preceded by one space character.	Readability	Warning
AA0018	The END, IF, REPEAT, UNTIL, FOR, WHILE, and CASE statement should always start a line.	Readability	Warning
AA0021	Variable declarations should be ordered by type.	Readability	Warning
AA0022	Substitute the IF THEN ELSE structure with a CASE.	Readability	Warning
AA0040	Avoid using nested WITH statements.	Readability	Warning
AA0072	The name of variables and parameters must be suffixed with the type or object name.	Readability	Info

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AA0073	The name of temporary variable must be prefixed with Temp.	Readability	Warning
AA0074	TextConst and Label variable names should have an approved suffix.	Readability	Warning
AA0087	Lowering permissions should only be used in tests	Design	Warning
AA0100	Do not have identifiers with quotes in the name.	Design	Warning
AA0101	Use camel case property values in pages of type API.	Design	Warning
AA0102	Use camel case name for field controls in pages of type API.	Design	Warning
AA0103	Use camel case property values in queries of type API.	Design	Warning
AA0104	Use camel case name for column controls in queries of type API.	Design	Warning
AA0105	PagePart controls must not refer to parent pages.	Design	Error
AA0106	A page of type API can only refer to the same subpage once.	Design	Error
AA0131	String parameters must match placeholders.	Design	Warning
AA0136	Do not write code that will never be hit.	Design	Warning
AA0137	Do not declare variables that are unused.	Design	Warning
AA0139	Do not assign a text to a target with smaller size.	Design	Warning
AA0150	Do not declare parameters by reference if their values are never changed.	Design	Warning
AA0161	Only use AssertError in Test Codeunits.	Design	Warning

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AA0175	Only find record if you need to use it.	Design	Warning
AA0181	The FindSet() or Find() methods must be used only in connection with the Next() method.	Design	Warning
AA0189	Only use a correct values of ApplicationArea.	Design	Warning
AA0194	Only write actions that have an effect.	Design	Warning
AA0198	Do not use identical names for local and global variables.	Design	Warning
AA0199	Use only a correct order for ApplicationArea.	Design	Warning
AA0200	When ApplicationArea is set to 'All', no other values for ApplicationArea should be specified.	Design	Warning
AA0201	When ApplicationArea is set to 'Basic', you must also specify 'Suite'.	Design	Warning
AA0462	The CalcDate should only be used with DataFormula variables. Alternatively the string should be enclosed using the <> symbols.	Localizability	Warning
AA0202	To avoid confusion, do not give local variables the same name as fields, methods or actions in the same scope.	Design	Warning
AA0203	To avoid confusion, do not give methods the same name as fields or actions in the same scope.	Design	Warning
AA0204	To avoid confusion, do not give global variables the same name as fields, methods or actions in the same scope.	Design	Warning
AA0205	Variables must be initialized before usage.	Design	Warning

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AA0206	The value assigned to a variable must be used.	Design	Warning
AA0207	The EventSubscriber method must be local.	Design	Warning
AA0210	Avoid non-indexed fields into filtering.	Design	Info
AA0211	Avoids a runtime error from using CalcFields on a field that is not a FlowField or a field of type Blob.	Design	Warning
AA0213	Obsoleted object must have a state 'Pending' or 'Removed' and a justification specifying why this field is being obsoleted.	Design	Warning
AA0214	The local record should be modified before saving to the database.	Design	Warning
AA0215	Follow the style guide about the best practices for naming.	Readability	Warning
AA0216	Use a text constant for passing user messages and errors without concatenations.	Localizability	Warning
AA0217	Use a text constant or label for format string in StrSubstNo.	Localizability	Warning
AA0218	You must write a tooltip in the Tooltip property for all controls of type Action and Field that exist on page objects.	Localizability	Warning
AA0219	The Tooltip property of Fields must start with 'Specifies'.	Localizability	Warning
AA0220	The value of the Tooltip property of Fields must be filled.	Localizability	Warning
AA0221	You must specify a OptionCaption property for all fields which source expressions is not a table field.	Localizability	Warning

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AA0222	SIFT index should not be used on primary or unique key.	Design	Warning
AA0223	The value of the OptionCaption property of Fields must be filled in.	Localizability	Warning
AA0224	The count of option captions specified in the OptionCaption property is wrong.	Localizability	Warning
AA0225	You must specify a caption in the Caption property for Fields that exist on page objects.	Localizability	Warning
AA0226	The value of the Caption property of Fields must be filled in.	Localizability	Warning
AA0227	Optional return value should not be omitted in upgrade codeunits.	Design	Warning
AA0228	The local method must be used; otherwise removed.	Design	Warning
AA0230	Version should not be specified for internal assemblies.	Design	Warning
AA0231	StrSubstNo or string concatenation must not be used as a parameter in the Error method.	Design	Warning
AA0232	The FlowField of a table should be indexed.	Design	Info
AA0233	Use Get(), FindFirst() and FindLast() without Next() method.	Design	Warning
AA0235	When using 'OnInstallPerCompany' you should also add 'Company - Initialize'::'OnCompanyInitialize' event subscriber.	Design	Info
AA0237	The name of non-temporary variables must not be prefixed with Temp.	Readability	Warning

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AA0240	Email and Phone No must not be present in any part of the source code.	Design	Warning
AA0241	Use all lowercase letters for reserved language keywords.	Readability	Warning
AA0448	You must use the FieldCaption method instead of the FieldName method and TableCaption method instead of TableName method.	Localizability	Warning
AA0470	Placeholders should have a comment explaining their content.	Localizability	Warning
AA0242	Limit JIT loads by selecting all fields for load.	Design	Warning

See Also

Using the Code Analysis Tool
Ruleset for the Code Analysis Tool
Using the Code Analysis Tools with the Ruleset

PerTenantExtensionCop Analyzer Rules

2/17/2021 • 2 minutes to read • Edit Online

PerTenantExtensionCop is an analyzer that enforces rules that must be respected by extensions meant to be installed for individual tenants.

Rules

ID	TITLE	CATEGORY	DEFAULT SEVERITY
PTE0001	Object ID must be in free range.	ObjectValidation	Error
PTE0002	Field ID must be in free range.	ObjectValidation	Error
PTE0003	Functions must not subscribe to CompanyOpen events.	ObjectValidation	Error
PTE0004	Table definitions must have a matching permission set.	ObjectValidation	Error
PTE0005	Property 'target' has invalid value.	PackageValidation	Error
PTE0006	Encryption key functions must not be invoked.	PackageValidation	Error
PTE0007	Test assertion functions are not allowed in a non-test context.	PackageValidation	Error
PTE0008	Fields must use ApplicationArea property.	PackageValidation	Error
PTE0009	This app.json property must not be used for per-tenant extensions.	PackageValidation	Error
PTE0010	The extension name length must not exceed the specified limit.	PackageValidation	Error
PTE0011	The extension publisher length must not exceed the specified limit.	PackageValidation	Error
PTE0012	InternalsVisibleTo should not be used as a security feature.	Extensibility	Warning

See Also

Using the Code Analysis Tool
Ruleset for the Code Analysis Tool
Using the Code Analysis Tools with the Ruleset

UICop Analyzer Rules

2/17/2021 • 2 minutes to read • Edit Online

UICop is an analyzer that enforces rules that must be respected by extensions meant to customize the Web Client.

Rules

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AW0001	The Web client does not support displaying the Request page of XMLPorts.	WebClient	Warning
AW0002	The Web client does not support displaying both Actions and Fields in Cue Groups. Only Fields will be displayed.	WebClient	Warning
AW0003	The Web client does not support displaying Repeater controls containing Parts.	WebClient	Warning
AW0004	A Blob cannot be used as a source expression for a page field.	WebClient	Warning
AW0005	Actions should use the Image property.	WebClient	Info
AW0006	Pages and reports should use the UsageCategory and ApplicationArea properties to be searchable.	WebClient	Info
AW0007	The Web client does not support displaying Repeater controls that contain FlowFilter fields.	WebClient	Error
AW0008	The Web client does not support displaying Repeater controls in pages of type Card, Document, and ListPlus.	WebClient	Warning
AW0009	Using a Blob with subtype Bitmap on a page field is deprecated. Instead use the Media/MediaSet data types.	WebClient	Warning

ID	TITLE	CATEGORY	DEFAULT SEVERITY
AW0010	A Repeater control used on a List page must be defined at the beginning of the area(Content) section.	WebClient	Warning
AW0011	Add PromotedOnly="true" to some or all promoted actions to avoid identical actions from appearing in both the promoted and default sections of the command bar.	WebClient	Info

See Also

Using the Code Analysis Tool
Ruleset for the Code Analysis Tool
Using the Code Analysis Tools with the Ruleset

Isolated Storage

2/17/2021 • 2 minutes to read • Edit Online

Isolated Storage is a data storage that provides isolation between extensions, so that you can keep keys/values in one extension from being accessed from other extensions. Keys/values in the Isolated Storage are accessible through an API. The involved option type is DataScope Option Type.

The methods supported for IsolatedStorage are:

METHOD	DESCRIPTION	FOR MORE INFORMATION, SEE
Set(String, String, [DataScope])	Sets the value associated with the specified key within the extension.	Set(String, String, [DataScope]) Method
Get(String, [DataScope], var Text)	Gets the value associated with the specified key within the extension.	Get(String, [DataScope], var Text) Method
Contains(String, [DataScope])	Determines whether the storage contains a value with the specified key within the extension.	Contains(String, [DataScope]) Method
Delete(String, [DataScope])	Deletes the value with the specified key from the isolated storage within the extension.	ISOLATEDSTORAGE.DELETE Method
SetEncrypted(String, String, [DataScope])	Encrypts and sets the value associated with the specified key. The input string cannot exceed a length of 215 plain characters; be aware that special characters take up more space.	SetEncrypted(String, String, [DataScope]) Method

See Also

DataScope Option Type

File Handling and Text Encoding

2/17/2021 • 4 minutes to read • Edit Online

There are several AL methods that you can use to open files, import and export files to and from Dynamics 365 Business Central, and more. For a list of methods, see File Data Type.

The following are recommended best practices for working with files:

- Use fully qualified paths to eliminate ambiguity.
- Be aware of operating system file access restrictions when designing applications that use files. Consider which users have access to files and directories and what Access Control List (ACL) that you need to apply to file directories.

Text encoding

Text encoding is the process of transforming bytes of data into readable characters for users of a system or program. When you import a file as text or as a stream, the text encoding format ensures that all the language-specific characters are represented correctly in Dynamics 365 Business Central. When you export a file as text or as a stream, the text encoding format ensures that all the language-specific characters are represented correctly in the system or program that will read the exported file.

Encoding formats

You can specify text encoding for the following objects.

OBJECT OR DATA TYPE	FOR MORE INFORMATION, SEE
XMLports	TextEncoding Property (XMLports)
File	OPEN Method (File)
BLOB	CREATEINSTREAM Method (BLOB) CREATEOUTSTREAM Method (BLOB)

There are several industry text encoding formats and different systems support different formats. Internally, Dynamics 365 Business Central uses Unicode encoding. For exporting and importing data with an XMLport, it supports MS-DOS, UTF-8, UTF-16, and Windows encoding formats.

Data is imported and exported as follows:

- When data is imported from an external file, it is read using the format that is specified by the **TextEncoding** property or parameter, and then converted to Unicode in Dynamics 365 Business Central.
- When data is exported to an external file, it is converted from Unicode in Dynamics 365 Business Central, and then written to the file in the format that is specified by the **TextEncoding** property or parameter.

You should set the text encoding to the encoding format that is compatible with the system or program that you will be exporting to or importing from. The following sections describe the available text encoding formats.

MS-DOS encoding format

MS-DOS encoding, which is also referred to as OEM encoding, is an older format than UTF-8 and UTF-16, but it

is still widely supported.

MS-DOS encoding requires a different character set for each language. When the property is set to MS-DOS, text is encoded by using the system locale language of the computer that is running Dynamics 365 Business Central service instance. So if you use MS-DOS encoding, you should set the system locale language of server instance computer to match the language of the data that is being imported or exported. For example, if an XMLport includes text in Danish, then you should set the system locale language of the server instance computer to Danish before the XMLport is run.

You should choose **MS-DOS** with XMLports that were created in earlier versions of Dynamics 365 Business Central.

UTF-8 encoding format

UTF-8 encoding is a Unicode Transformation Format that uses one byte (8 bits) to encode each character. UTF-8 is based on the Unicode character set, which includes most characters of all languages in a single character set.

Unlike MS-DOS, when you use UTF-8, you do not have to consider the language settings of Dynamics 365 Business Central service instance or the external system or program that will read or write the data.

UTF-8 is compatible with ASCII so that it will understand files written in ASCII format.

UTF-8 is the most common encoding format and the recommended setting if you are not sure of the format that is supported by the system that you are integrating with.

UTF-16 encoding format

UTF-16 encoding resembles UTF-8 except that UTF-16 uses 2 bytes (16 bits) to encode each character. UTF-16 is also based on the Unicode character set, so you do not have to consider the language setting of Dynamics 365 Business Central service instance or the external system or program that reads or writes the data.

UTF-16 includes two encoding schemes which mandate the byte order: UTF-16LE and UTF-16BE. The schemas are supported as follows:

- When exporting, the file is written using UTF-16LE encoding.
- When importing, the file is read using the UTF-16, UTF-16LE, or UTF-16BE, depending on encoding scheme of the file itself.

A UTF-16 encoded file will typically be larger than the same file encoded with UTF-8, except for Eastern language character sets, which will typically be smaller.

UTF-16 is incompatible with ASCII so that it will not understand files written in ASCII format.

Windows format

Windows encoding is also referred to as ANSI encoding. If you set the text encoding to **Windows**, you can import and export text files that are based on the Windows codepage on the user's computer. As a result, you do not have to consider the language setting of Dynamics 365 Business Central service instance computer or the external system or program that reads or writes the data.

For example, if an XMLport can import bank files from a foreign bank in addition to a domestic bank, use Windows encoding instead of MS-DOS encoding to avoid changing the language of Dynamics 365 Business Central service instance computer.

Windows encoding is compatible with ASCII so that it will understand files written in ASCII format.

See Also

TextEncoding Property (XMLports)
File Data Type

FlowFields

2/17/2021 • 2 minutes to read • Edit Online

FlowFields display the result of the calculation described in the CalcFormula Property. For example, the **Account Balance** field in the General Ledger Account table shows the balance of the account and is calculated as the sum of the NetAmount fields for all General Journal entries in the account.

FlowFields increase performance in activities such as calculating the balance of your customers. In traditional database systems, this involves a series of accesses and calculations before a result is available. By using FlowFields, the result is immediately available. You can further optimize the performance of Flowfields by enabling or disabling SIFT. For more information, see SumIndexField Technology (SIFT).

FlowFields are not physical fields that are stored in the database. They are a description of a calculation and a location for the result to be displayed. Because the information in FlowFields exists only at run time, values in FlowFields are automatically initialized to 0 (zero). To update a FlowField, use the CalcFields Method (Record). If a FlowField is the direct source expression of a control on a page, then the FlowField is automatically calculated when the page is displayed.

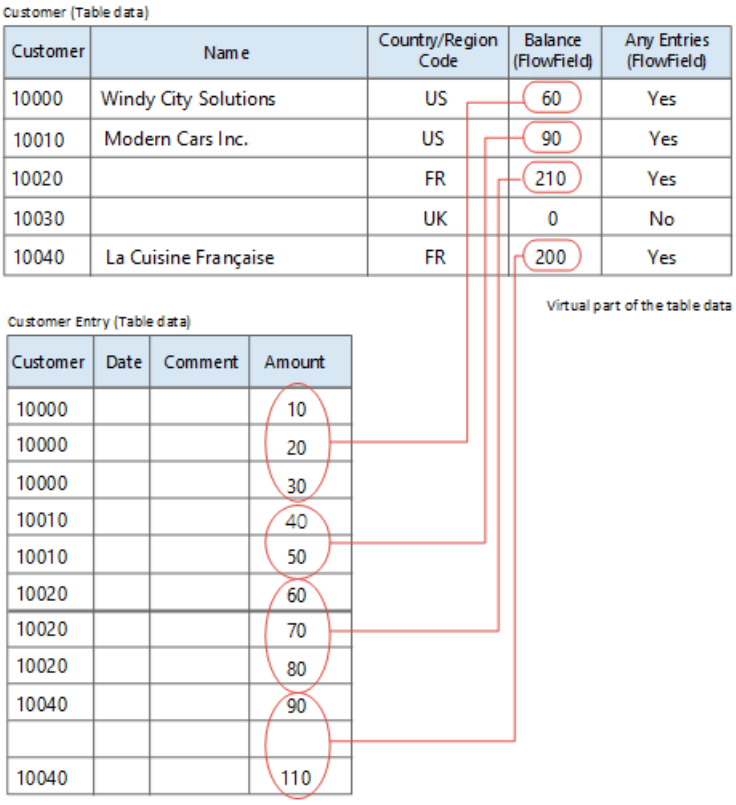
FlowField types

There are seven types of FlowFields. Each is described in the following table.

FLOWFIELD TYPE	FIELD TYPE	DESCRIPTION
Sum	Decimal, Integer, BigInteger, or Duration	The sum of a specified set in a column in a table.
Average	Decimal, Integer, BigInteger, or Duration	The average value of a specified set in a column in a table.
Exist	Boolean	Indicates whether any records exist in a specified set in a table.
Count	Integer	The number of records in a specified set in a table.
Min	Any	The minimum value in a column in a specified set in a table.
Max	Any	The maximum value in a column in a specified set in a table.
Lookup	Any	Looks up a value in a column in another table.

Example

Consider the Customer table in the following illustration. This table contains two FlowFields. The field named **Any Entries** is a FlowField of the Exist type, and the **Balance** field is a FlowField of the Sum type.



The figure shows that the value in the Balance FlowField for customer number 10000 (Windy City Solutions) is retrieved from the Amount column in the Customer Entry table. The value is the sum of the amount fields for the entries that have the customer number 10000.

```
Sum = 10 + 20 + 30 = 60.
```

The values shown in the **Balance** column in the Customer table for customers 10010, 10020, and 10040 are found in the same way. For customer number 10030 the value is 0 (zero), as there are no entries in the Customer Entry table that have a Customer No. that equals 10030.

In this example, the Balance FlowField in the Customer table reflects the sum of a specific subset of the Amount fields in the Customer Entry table. How the calculation of a FlowField is to be made, is defined in a calculation formula. The calculation formula for the **Balance** field is:

```
Sum("Customer Entries".Amount WHERE(CustNo=FIELD(CustNo)))
```

Correspondingly, the **Any Entries** field, which indicates whether any entries exist, has the following definition:

```
Exist("Customer Entries" WHERE(CustNo=FIELD(CustNo)))
```

See Also

CalcFields Method (Record)
Create FlowFields and FlowFilters

Create FlowFields and FlowFilters

2/17/2021 • 3 minutes to read • Edit Online

This topic describes the procedure and the properties used to create FlowFields and FlowFilters.

A FlowField performs a set of calculations and displays the results immediately. A FlowFilter displays the results based on the user input to calculate the filtered values that will affect the calculation of a FlowField. The FlowFields and FlowFilters are not physical fields; these fields act as a virtual field which does not actually exist in the database. They are a description of a calculation and a location for the result to be displayed which is typically derived in the CalcFormula Property.

For more information about the FlowField type, see FlowFields, and for more information about the FlowFilter type, see FlowFilter Overview.

Classifying the field type

In order to create FlowFields and FlowFilters, you must first classify the field type by using the FieldClass Property. For more information, see FieldClass Property. By classifying the field as a FlowField or a FlowFilter type, you enable the fields to act as a virtual field whose value can be dynamically derived based on the calculation formula.

Calculation formula

A FlowField type is always associated with a calculation formula that determines how the FlowField is calculated. Likewise, the FlowFilter type is associated with the calculation formula. To perform the calculations by using the FlowField and FlowFilter type, you must derive those fields in the calculation formula which you classify in the table. You can choose from several methods of calculations including sum (total), average, maximum value, minimum value, record count, lookup, and more, by using the CalcFormula Property. For more information about the syntax and formulas, see Calculation Formulas and the CalcFormula Property.

Example

In the following example, `MyTable` sets the `Global Dimension 1 Filter` and `Global Dimension 2 Filter` fields whose values are based only on the dimension values included in the filter. Also, some of the following fields formulate the currency filter to one single currency because you do not store the filter value on the entries, hence you define the `Currency Filter` as a FlowFilter type. `Total Amount`, `Amount upper bound`, `Amount lower bound`, `First Entry`, and `Customer Balance` are classified as a FlowField type and here you specify the calculations. These fields display the results immediately based on the filters that you apply in the user interface.

```
table 50123 MyTable
{
    fields
    {
        field(1;MyField; Decimal)
        {
            Description='New field';
        }

        field(2;"No."; Code[20])
        {
            Description='Serial number of the service';
        }
```

```
        field(3;"Global Dimension 1 Filter"; Code[20])
        {
            FieldClass = FlowFilter;
        }

        field(4;"Global Dimension 2 Filter"; Code[20])
        {
            FieldClass = FlowFilter;
        }

        field(5;"Currency Filter"; Code[10])
        {
            FieldClass = FlowFilter;
        }

        field(6; "Total Amount"; Decimal)
        {
            FieldClass = FlowField;
            CalcFormula = Sum("Detailed Cust. Ledg. Entry"."Amount (LCY)"
            WHERE ("Customer No."=Field("No."),
            "Initial Entry Global Dim. 1"=Field("Global Dimension 1 Filter"),
            "Initial Entry Global Dim. 2"=Field("Global Dimension 2 Filter"),
            "Currency Code"=Field("Currency Filter")
            ) );
        }

        field(7; "Amount upper bound"; Decimal)
        {
            FieldClass = FlowField;
            CalcFormula = max ("Detailed Cust. Ledg. Entry"."Amount (LCY)"
            WHERE ("Customer No." = Field ("No."),
            "Initial Entry Global Dim. 1" = Field ("Global Dimension 1 Filter"),
            "Initial Entry Global Dim. 2" = Field ("Global Dimension 2 Filter"),
            "Currency Code" = Field ("Currency Filter")
            ));
        }

        field(8; "Amount lower bound"; Decimal)
        {
            FieldClass = FlowField;
            CalcFormula = min ("Detailed Cust. Ledg. Entry"."Amount (LCY)"
            WHERE ("Customer No." = Field ("No."),
            "Initial Entry Global Dim. 1" = Field ("Global Dimension 1 Filter"),
            "Initial Entry Global Dim. 2" = Field ("Global Dimension 2 Filter"),
            "Currency Code" = Field ("Currency Filter")
            ));
        }

        field(9; "First entry"; Boolean)
        {
            CalcFormula = exist (Customer where (Payments = const (0),
                                                 "No." = field ("No.")));
            FieldClass = FlowField;
            Caption = 'Specifies whether it is the customer''s first entry';
        }

        field(10; "Customer Balance"; Decimal)
        {
            FieldClass = FlowField;
            CalcFormula = lookup (Customer.Balance where ("No." = field ("No.")));
        }
    }
}
```

See Also

FlowFields
FlowFilter Overview
Calculation Formulas and the CalcFormula Property

SumIndexField Technology (SIFT)

2/17/2021 • 2 minutes to read • Edit Online

SumIndexField Technology (SIFT) lets you quickly calculate the sums of numeric data type (Decimal, Integer, BigInteger, and Duration) columns in tables, even in tables with thousands of records. SIFT is used to optimize the performance of FlowFields and query results in a Business Central application. SIFT involves performing a series of database calls and calculations before arriving at a result. The topics in this section describes how SIFT is implemented in Business Central.

See Also

SIFT and SQL Server
SIFT Tuning and Tracing
SIFT Performance
FlowFields

SIFT and SQL Server

2/17/2021 • 2 minutes to read • Edit Online

A SumIndexField is always associated with a key and each key can have a maximum of 20 SumIndexFields associated with it. In this topic, a key that has at least one SumIndexField associated with it is a SIFT key.

> **IMPORTANT**
> Any field that has a numeric data type of Decimal, Integer, BigInteger, or Duration, can be associated with a key as a SumIndexField.

Implementing SIFT

Business Central uses *Indexed Views* to maintain SIFT totals. Indexed views are a standard SQL Server feature. An indexed view is like a SQL Server view except that the contents have been materialized (computed and stored) to speed up the retrieval of data. For more information about indexed views, see SQL Server Documentation.

Business Central creates one indexed view for each SIFT key that is enabled. When you create a SIFT key for a table, you must set the MaintainSIFTIndex Property for that key to **True** to enable the SIFT key and create the indexed view. After SQL Server has created the indexed view, it maintains the contents of the view when any changes are made to the base table. If you set the MaintainSIFTIndex Property for that key to **False**, SQL Server drops the indexed view and stops maintaining the totals.

The indexed view that is generated for a SIFT key is always created at the level of finest granularity. Therefore, if you create a SIFT key for `AccountNo., PostingDate`, the database will store an aggregated value for each account for each date. This means that in the worst case scenario, 365 records must be summed to generate the total for each account for a year.

The following is an example of how Business Central creates an indexed view for a SIFT key that consists of the **AccountNo.** and **PostingDate** fields with a total for the **Amount** field.

```
CREATE VIEW GLEntry$VSIFT$1 AS
SELECT SUM(Amount) as SUM$Amount, AccountNo, PostingDate
FROM GLEntry
GROUP BY AccountNo, PostingDate
;

CREATE UNIQUE CLUSTERED INDEX VSIFTIDX ON
GLEntry$VSIFT$1(AccountNo, PostingDate)
;
```

The following AL code example retrieves a total.

```
GLEntry.SETCURRENTKEY("G/L Account No.","Posting Date");
GLEntry.SETRANGE("G/L Account No.",'1110');
GLEntry.SETRANGE("Posting Date",DMY2DATE(1,1,2007),DMY2DATE(15,12,2007));
GLEntry.CALCSUMS(Amount);
```

The following code example shows how the same total is retrieved through an indexed view.

```
SELECT SUM(SUM$Amount)
FROM GLEntry$VSIFT$1 WITH(NOEXPAND)
WHERE AccountNo=?
AND PostingDate>=?
AND PostingDate<=?
```

See Also

SumIndexField Technology (SIFT)
Tuning and Tracing
SIFT and Performance

Tuning and Tracing

2/17/2021 • 2 minutes to read • Edit Online

As a result of using indexed views, SIFT keys are exposed to SQL Server tracing and tuning tools. For example, the SQL Server profiler can display information about which indexed views are maintained for a specific table. This makes it easy for you to assess the cost of maintaining SIFT keys and allows you to make informed decisions about any adjustments that might be required.

SIFT Keys

When data is inserted, updated, or deleted in a table, the SIFT keys that have been defined and enabled for that table are maintained. Maintaining these SIFT indexes has performance overhead. The size of the performance overhead depends on the number of keys and SumIndexFields that have been defined for each table. You should therefore give careful consideration to the number of SIFT keys that you define and only maintain the SIFT keys that are important for your application.

There is no need to maintain a SIFT key for a total that is only used periodically and can be easily generated by a report. You can still program Business Central to calculate the SIFT based totals even if the SIFT key is disabled. The calculation is performed directly on the base table.

You should consider combining indexes wherever possible.

Example

Maintaining two SIFT keys:

1. Key: "WareHouseId, ItemId, Color" SumField: "OnStock"
2. Key: "WareHouseId, ItemId, Size" SumField: "OnStock"

If there are only a few combinations of Size and Color (for example, less than 200), then one combined index/SIFT key should be sufficient.

1. The Combined Key:
2. "WareHouseId, ItemId, Color, Size" SumFIeld: "OnStock"

When you set the MaintainSIFTIndex Property of a key to **True**, this will be the SIFT key and create the indexed view to support it. However, disabling the SIFT key by setting MaintainSIFTIndex Property to **False** can improve performance in certain circumstances. Setting this property to **False** means that the SIFT functionality must be implemented by calculating the totals online instead of using the precalculated sums that are maintained by SIFT.

See Also

SumIndexField Technology (SIFT)
SIFT and SQL Server
SIFT Performance
FlowFields

SIFT and Performance

2/17/2021 • 2 minutes to read • Edit Online

This topic looks at the factors you must take into consideration when you deal with SIFT and performance.

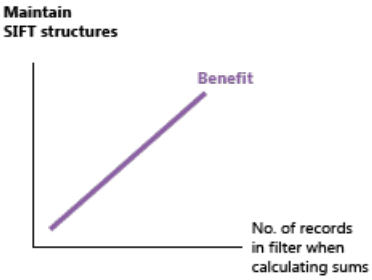
Performance Factors to Consider

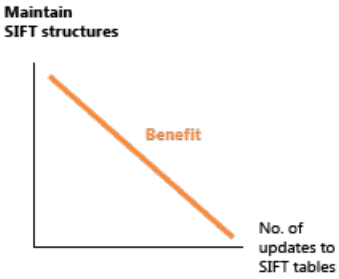
The factors that you must take into consideration when you deal with any performance problems that arise include:

- Designing your SIFT indexes optimally.
- Supporting too many SIFT indexes will affect performance.
- Having unnecessary date fields in the SIFT indexes of the base table will affect performance because they create three times as many entries as an ordinary field.
- Supporting too many fields in the SIFT indexes will also affect performance.
- The fields in the SIFT index that are used most regularly in queries must be positioned to the left in the SIFT index. Usually, the field that contains the greatest number of unique values must be placed on the left, with the field that contains the second greatest number of unique values on the right and so on. Integer fields generally contain the greatest number of unique values and Option fields contain a relatively small number of values.

Consider the costs and the benefits of maintaining SIFT indexes.

COST	BENEFIT
Updates to the SIFT indexes	Fast calculation of sums
Potential locking conflicts	





You can prevent the SIFT indexes from being updated by setting the MaintainSIFTIndex Property of the index in the base table to **False**. This means that you no longer benefit from SIFT's ability to calculate sums quickly. However, the SIFT functionality is still available. If the base table does not grow or only grows slowly, there is no need to set the MaintainSIFTIndex Property to **True** for any indexes that contain SumIndexFields. If the base table does grow, you should set the MaintainSIFTIndex Property to **True** for any indexes that contain SumIndexFields.

In Business Central , changes have been made to improve performance when accessing the database. One of these changes is that Business Central automatically maintains a count for all SIFT indexes. For more information about how this affects the **Count** and **Average** methods on FlowFields, see CalcFields Method. For more information about other data access changes, see Data Access.

IMPORTANT

It is important that you remember to perform tests every time you make any changes to the SIFT structures. You must ensure that the changes that you have made do not cause problems in any other areas of the application. You must also ensure that your changes do not have a negative effect on performance.

See Also

SumIndexField Technology (SIFT)
SIFT and SQL Server
SIFT Tuning and Tracing
FlowFields

Number Sequences in Business Central

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

In AL, you can create and manage number sequences that generate numeric identifiers for data and records. Business Central number sequences are built on SQL Server number sequences, which means that they are not associated with any tables. Instead, the application code references the number sequence object and coordinates the values across records.

The numbers in a number sequence are generated in ascending order at a defined interval. Numbers are used sequentially, but numbers can be skipped. This means that numbers used on records in tables can have gaps. These gaps, for example, can occur when transactions are rolled back or numbers are allocated but not used.

Number sequences and number series in the application

Unlike number sequences, number series are specially designed for scenarios where you have to use a continuous numbering system on transactional records. This typically includes documents such as purchase orders, sales orders, and invoices. The drawback of using continuous numbers is that when a number is requested by a transaction, the **No Series Line** table in the database is locked until the transaction completes. This can potentially block other users from being able to work.

However, number series also include an option called **Allow Gaps in Nos.**. This option actually uses number sequences in the underlying code, and allows you to implement non-continuous, non-blocking numbering. So, if there are no regulatory requirements for using continuous numbering, you can improve performance by allowing gaps and using a number sequence instead. Customer cards, sales quotes, and warehouse activities are examples of entities that typically do not require continuous numbering.

For more information about number series, see Create Number Series in the Business Central application help.

Creating and managing number sequences in AL

To create and manage number sequences, you use the `NumberSequence` data type and the following methods.

METHOD NAME	DESCRIPTION
[Insert(String, BigInteger][, BigInteger], [Boolean])	Creates a number sequence in the database, with the given parameters.
Exists(String[, Boolean])	Checks whether a specific number sequence exists.
Delete(String[, Boolean])	Deletes a specific number sequence.
Next(String[, Boolean])	Retrieves the next value from the number sequence.
Current(String[, Boolean])	Gets the current value from the number sequence, without doing any increment. The value is retrieved out of transaction. The value will not be returned on transaction rollback.

Examples

```
// Creates a NumberSequence object that starts with the value '0' and increments by '1'
NumberSequence.Insert('DefaultSequence');

// Creates a NumberSequence object that starts with the value '10' and increments by '1'
NumberSequence.Insert('StartsWithTenSequence', 10);

// Creates a NumberSequence object that starts with the value '0' and increments by '10'
NumberSequence.Insert('StartsWithZeroIncrementTenSequence', 0, 10);

// Creates a NumberSequence object that starts with the value '0', increments by '10', and is company-
specific
NumberSequence.Insert('MyCompanySequence', 0, 1, true);

// Verifies whether a specific NumberSequence object exists, and if so, deletes it

if NumberSequence.Exists('MySequence', true) then
    NumberSequence.Delete('MySequence',true);

// Gets and returns the current value in a NumberSequence object
number := NumberSequence.Current('MySequence',true);

// Gets and returns the next value in a NumberSequence object
number := NumberSequence.Next('MySequence',true);
```

See Also

Number Sequence data type
SQL Server Sequence Numbers

Extensible Enums

2/17/2021 • 3 minutes to read • Edit Online

An enumeration type, also known as an enum in programming, is a keyword used to declare a type that consists of a set of named constants. The list of named constants is called the enumeration list. Enums can be used as table fields, local and global variables, and parameters.

To declare an `enum` in AL you must specify an ID and a name. The enumeration list consists of values and each of the values are declared with an ID and a value. The value ID is the ordinal value on the enumeration list and must be unique. The following example shows the declaration of an enum, which can be extended, and has the four values; **None**, **Bronze**, **Silver**, and **Gold**.

```
enum 50121 Loyalty
{
    Extensible = true;

    value(0; None) { }
    value(1; Bronze) { }
    value(2; Silver) { }
    value(3; Gold)
    {
        Caption = 'Gold Customer';
    }
}
```

NOTE

While enums and enumextension objects have object IDs, these are not enforced by the license. In previous versions they reused the range for tables, and were checked against the license at deployment time, but this is no longer the case. Uniqueness validation is now enforced during installation, which will fail if an enum object ID clashes with an already installed enum. Thus, as always, it is important that you use object IDs in your assigned range. This is enforced for AppSource apps, but not for per-tenant extensions, or on-premise. The enum does not have to use the same ID as the table it is put on.

IMPORTANT

Only enums with the Extensible Property set to **true** can be extended.

Enumextension object

Enums can be extended in order to add more values to the enumeration list in which case the `Extensible` property must be set to `true`. The syntax for an enum extension, which extends the **Loyalty** enum with the value **Diamond**, is shown below.

```
enumextension 50130 LoyaltyWithDiamonds extends Loyalty
{
    value(50130; Diamond)
    {
        Caption = 'Diamond Level';
    }
}
```

Usage

When referencing a defined enum from code, you use the syntax as illustrated below.

`enum Loyalty`

If you want to define an enum as a table field type, use the syntax illustrated below:

```
field(50100; Loyal; enum Loyalty) {}
```

Or, as a variable:

```
var
    LoyaltyLevel: enum Loyalty;
```

In code, you address a specific enum value like in the following example:

```
codeunit 50140 EnumUsage
{
    procedure Foo(p: enum Loyalty)
    var
        LoyaltyLevel: enum Loyalty;
    begin
        if p = p::Gold then begin
            LoyaltyLevel := p;
        end;
    end;
}
```

Example

The following example illustrates how to define an enum extension of `TypeEnum`, using this in a table extension `TableWithRelationExt` and displaying this as a control on a new page.

```
enumextension 50133 TypeEnumExt extends TypeEnum
{
    value(10; Resource) { }
}

tableextension 50135 TableWithRelationExt extends TableWithRelation
{
    fields
    {
        modify(Relation)
        {
            TableRelation = if (Type = const (Resource)) Resource;
        }
    }
}

page 50133 PageOnRelationTable
{
    SourceTable = TableWithRelation;
    SourceTableView = where (Type = const (Resource));
    PageType = List;

    layout
    {
        area(Content)
        {
            repeater(MyRep)
            {
                field(Id; Id)
                {
                    ApplicationArea = All;
                }
                field(Type; Type)
                {
                    ApplicationArea = All;
                }
                field(Relation; Relation)
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}
```

TIP

For another example of how to extend the usage of the `TableRelation` property in connection with enums, see TableRelation Property.

Business Central On-Premises

If you want to extend an existing Dynamics 365 on-premises enum, it is possible to mark a table field in C/SIDE as extensible. To enable running C/SIDE and AL side-by-side, see Running C/SIDE and AL Side-by-Side.

Table field options in C/SIDE have three properties to enable enum support:

PROPERTY NAME	DATA TYPE
Extensible	Boolean, default value is **No**.

PROPERTY NAME	DATA TYPE
EnumTypeId	Integer
EnumTypeName	Text

Some table fields share options that are semantically identical. In those cases the **EnumTypeId** and **EnumTypeName** must be the same across all the fields. There is no design or runtime check for collision of IDs, but loading generated symbols, see Running C/SIDE and AL Side-by-Side, into the compiler will show collision errors.

Conversions

Conversion to and from `enum` is more strict than for `Options` in C/SIDE.

- An enum can be assigned/compared to an enum of the same type.
- To be backwards compatible we support conversion to/from any `Option` for now.

For information about assigment compatibility, see AssignmentCompatibility Property.

See Also

AL Data Types
TableRelation Property
Extensible Property
Enum Data Type
AssignmentCompatibility Property

Protected Variables

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

The `protected` keyword can be used to make variables accessible between tables and table extensions and between pages and page extensions. If you want to only expose some variables as `protected`, you must create two sections of `var` declarations. See the syntax below.

Syntax

```
protected var
        myInt: Integer; // protected var

var
        myLocalInt: Integer; // local var
```

Example

The example below illustrates how to declare and use a protected variable.

```
page 50100 MyPage
{
    SourceTable = Customer;
    PageType = Card;

    layout
    {
        area(Content)
        {
            group(General)
            {
                field(Name; Name)
                {
                    ApplicationArea = All;
                }
            }
            group(Advanced)
            {
                Visible = ShowBalance;

                field(Balance; Balance)
                {
                    ApplicationArea = All;
                }
            }
        }
    }

    protected var
        [InDataSet]
        ShowBalance: Boolean;
}

pageextension 50101 MyPageExt extends MyPage
{
    layout
    {
```

```
    {
        addlast(Content)
        {
            group(MoreBalance)
            {
                Visible = ShowBalance; // ShowBalance from MyPage

                field("Balance (LCY)"; "Balance (LCY)")
                {
                    ApplicationArea = All;
                }
            }
        }

    }

    actions
    {
        addlast(Navigation)
        {
            action(ToggleBalance)
            {
                ApplicationArea = All;
                trigger OnAction()
                begin
                    ShowBalance := not ShowBalance; // Toggle ShowBalance from MyPage.
                end;
            }
        }
    }
}
```

See Also

AL Method Reference
Properties
Access Property
Extensible Property

Working with Labels

2/17/2021 • 2 minutes to read • Edit Online

Labels are string constants displayed in the Business Central client that can be translated into multiple languages, such as captions, descriptions, or messages. This way, the user interface can be displayed in different languages. For more information on how translation is carried out in Dynamics 365 Business Central, see Multilanguage development.

Label syntax

Labels have a specific syntax defined by a text constant followed by three optional parameters. They must be comma-separated, but the order of the parameters is not enforced. The parameters that you can set are described in the following table.

PARAMETER	TYPE	DESCRIPTION
Comment	Text	Used for general comments about the label, specifically about the placeholders in that label.
Locked	Boolean	When Locked is set to **true**, the label should not be translated. Default value is **false**.
MaxLength	Integer	Determines how much of the label is used. If no maximum length is specified, the string can be any length.

Using labels

A label can take the form of four different AL structures. It can be the property value of certain page and report properties, the label data type variable and a report or a page label. The different possibilities are explained in more detail below.

Properties

The label syntax is used in properties that are set to display text on the user interface. It applies to the following properties:

- Caption Property
- ToolTip Property
- OptionCaption Property
- AdditionalSearchTerms Property
- InstructionalText Property
- PromotedActionCategories Property
- RequestFilterHeading Property

The following example shows the label syntax when it is used as property value for the **Caption** property.

```
Caption = 'Developer translation for %1',  Comment = '%1 is extension name', locked = false, MaxLength=999;
```

Report labels

Report labels are used by RDL and Word report layouts as, for example, the caption for a field, the title for a chart, or the title for the report itself. For a code example on how to use report labels for an RDL layout, see Walkthrough: Designing a Report from Multiple Tables.

Report labels are defined inside the `labels` control of a report object, as shown in the code sample below.

```
labels
{
  LabelName1='LabelText1',Comment='Foo',MaxLength=999,Locked=true;
  LabelName2 = 'LabelText2',Comment='Foo',Locked=false;
}
```

Page labels

Page labels are used to display plain text on a page, such as instructions or informative texts. You can find several examples of page labels in the Rapidstart Services Wizard in page `"Config. Wizard"`.

Page labels are defined by a `label(Name)` control inside the `area(Content)` part of a page. The following code shows how to define a page label.

```
label(BeforeSetupCloseMessage)
{
    ApplicationArea = Basic, Suite;
    Caption = 'If you still need to change setup data, do not change the profile.'
}
```

Label data type

The Label Data Type denotes a string variable used to define error messages, questions, captions, tokens, or other text constants displayed to the user.

The following code sample illustrates how to use the **Label** data type.

```
var
a:Label'LabelText',Comment='Foo',MaxLength=999,Locked=true;
```

The **Label** variable names should have an approved suffix. For more information, see CodeCop Rule AA0074.

See Also

Working with labels
Working with Translation Files
Label Data Type
Report Layouts

Table Object

2/17/2021 • 2 minutes to read • Edit Online

Tables are the core objects used to store data in Dynamics 365 Business Central. No matter how data is registered in the product - from a web service to a finger swipe on the phone app, the results of that transaction will be recorded in a table.

The structure of a table has four sections:

- The first block contains metadata for the overall table, such as the table type.
- The fields section describes the data elements that make up the table, such as their name and the type of data they can store.
- The keys section contains the definitions of the keys that the table needs to support.
- The final section details the triggers and code that can run on the table.

IMPORTANT

Only tables with the Extensible Property set to **true** can be extended.

NOTE

Extension objects can have a name with a maximum length of 30 characters.

IMPORTANT

System and virtual tables can't be extended. System tables are created in the ID range of 2.000.000.000 and above. For more information about object ranges, see Object Ranges.

Snippet support

Typing the shortcut `ttable` will create the basic layout for a table object when using the AL Language extension in Visual Studio Code.

TIP

Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Table example

This table stores address information and it has four fields; `Address`, `Locality`, `Town/City`, and `County`.

```
table 50104 Address
{
    Caption = 'Sample table';
    DataPerCompany = true;

    fields
    {
        field(1; Address; Text[50])
        {
            Description = 'Address retrieved by Service';
        }
        field(2; Locality; Text[30])
        {
            Description = 'Locality retrieved by Service';
        }
        field(3; "Town/City"; Text[30])
        {
            Description = 'Town/City retrieved by Service';
        }
        field(4; County; Text[30])
        {
            Description = 'County retrieved by Service';

            trigger OnValidate();
            begin
                ValidateCounty(County);
            end;

        }
    }
    keys
    {
        key(PrimaryKey; Address)
        {
            Clustered = TRUE;
        }
    }

    var
        Msg: Label 'Hello from my method';

    trigger OnInsert();
    begin

    end;

    procedure MyMethod();
    begin
        Message(Msg);
    end;
}
```

System fields

The Dynamics 365 Business Central platform will automatically add several system fields to tables. For more information, see System Fields.

See Also

AL Development Environment
Table Overview
Table Extension Object
SqlTimestamp Property

Table Keys
Table, Table Fields, and Table Extension Properties

Table Extension Object

2/17/2021 • 2 minutes to read • Edit Online

The table extension object allows you to add additional fields or to change some properties on a table provided by the Dynamics 365 Business Central service. In this way, you can add data to the same table and treat it as a single table. For example, you may want to create a table extension for a retail winter sports store. In your solution you want to have `ShoeSize` as an additional field on the customer table. Adding this as an extension allows you to write code for the customer record and also include values for the `ShoeSize`.

Along with defining other fields, the table extension is where you write trigger code for your additional fields.

When developing a solution for Dynamics 365 Business Central , you will follow the code layout for a table extension as shown in the example below.

IMPORTANT

Only tables with the Extensible Property set to **true** can be extended.

NOTE

Extension objects can have a name with a maximum length of 30 characters.

IMPORTANT

System and virtual tables cannot be extended. System tables are created in the ID range of 2.000.000.000 and above. For more information about object ranges, see Object Ranges.

IMPORTANT

Extending tables from Dynamics 365 for Sales is currently not supported.

Snippet support

Typing the shortcut `ttableext` will create the basic layout for a table extension object when using the AL Language extension in Visual Studio Code.

Properties

Using a table extension allows you to overwrite some properties on fields in the base table. For a list of Table properties, see Table and Table Extension Properties.

Table extension syntax

```
tableextension Id MyExtension extends MyTargetTable
{
    fields
    {
        // Add changes to table fields here
    }

    var
        myInt: Integer;
}
```

Table extension example

This table extension object extends the Customer table object by adding a field `ShoeSize`, with ID 50116 and the data type `Integer`. It also contains a procedure to check if the `ShoeSize` field is filled in.

```
tableextension 50115 RetailWinterSportsStore extends Customer
{
    fields
    {
        field(50116;ShoeSize;Integer)
        {
            trigger OnValidate();
            begin
                if (rec.ShoeSize < 0) then
                begin
                   message('Shoe size not valid: %1', rec.ShoeSize);
                end;
            end;
        }
    }

    procedure HasShoeSize() : Boolean;
    begin
        exit(ShoeSize <> 0);
    end;

    trigger OnBeforeInsert();
    begin
        if not HasShoeSize then
            ShoeSize := Random(42);
    end;
}
```

See Also

AL Development Environment
Table Overview
Table Object
Table, Table Fields, and Table Extension Properties
Table Keys

Table Keys

2/17/2021 • 7 minutes to read • Edit Online

The database management system, which is SQL Server, keeps track of data in a table using a primary key. The primary key is composed of up to 16 fields in a record. The combination of values in fields in the primary key makes it possible to uniquely identify each record.

A table definition in AL can contain a list of keys. A key is a sequence of one or more field IDs from the table. Up to 40 keys can be associated to a table.

Primary keys

The first key in the list of keys in a table definition is the primary key.

The primary key determines the logical order in which records are stored, regardless of their physical placement.

Logically, records are stored sequentially in ascending order and are sorted by the primary key. Before adding a new record to a table, SQL Server checks if the information in the record's primary key fields is unique and only then inserts the record into the correct logical position. Records are sorted dynamically so the database is always structurally correct. This allows for fast data manipulation and retrieval.

The primary key is always active, and SQL Server keeps the table sorted in primary key order and rejects records with duplicate values in primary key fields. Therefore, the values in the primary key must always be unique. Note that it is not the value in each field in the primary key that must be unique, but it is the combination of values in all fields that make up the primary key.

> **NOTE**
>
> In the development environment, it is technically possible to create a primary key based on up to 20 fields. However, because of SQL Server limitations, only the first 16 are used.

Secondary keys

Any keys defined after the primary key in the list of keys in a table definition are called *secondary keys*.

A secondary key is implemented on SQL Server using an additional structure that is called an *index*. This is like an index that is used in textbooks. A textbook index alphabetically lists important terms at the end of a book. Next to each term are page numbers. You can quickly search the index to find a list of page numbers (addresses), and you can locate the term by searching the specified pages. The index is an exact indicator that shows where each term occurs in the textbook.

When you define a secondary key and mark it as enabled, an index is automatically maintained on SQL Server. The index reflects the sorting order that is defined by the key. Several secondary keys can be active at the same time.

A secondary key can be changed to be disabled, which does not occupy database space, and does not use time during updates to maintain its index. Disabled keys can be re-enabled, although this can be time-consuming because SQL Server must scan the whole table to rebuild the index.

The fields that make up the secondary keys do not always contain unique data, and SQL Server does not reject records with duplicate data in secondary key fields. If two or more records contain identical information in the

secondary key, then SQL Server uses the primary key for the table to resolve this conflict.

Unique secondary keys

A key definition includes the Unique property that you can use to create a unique constraint on the table in SQL Server. A unique key ensures that records in a table do not have identical field values. With a unique key, when table is validated, the key value is checked for uniqueness. If the table includes records with duplicate values, the validation fails. Another benefit of unique indexes is providing information to the query optimizer that helps produce more efficient execution plans.

Like primary keys, you can create unique secondary keys that are comprised of multiple fields. In this case, it's the combination of the values in the secondary key that must be unique. For example, if you have a **Customer** table, you could create a unique key for the **Name**, **Address**, and **City** fields to make sure that there are no customers that have the same combination of values for these fields.

Unlike primary keys, it is possible to define multiple unique secondary keys on a table.

> **NOTE**
> The `Unique` property is not supported in table extension objects.

System keys

There is always a unique secondary key on the **SystemId** field.

Clustered and non-clustered keys

A key definition includes the Clustered property that you use to create a clustered index. A clustered index determines the physical order in which records are stored in the table. Based on the key value, records are sorted in ascending order. Using a clustered key can speed up the retrieval of records.

There can be only one clustered index per table. By default the primary is configured as a clustered key.

> **NOTE**
> The `Clustered` property is not supported in table extension objects.

Sort orders and secondary keys

The following example shows how the primary key influences the sort order when a secondary key is active. The Customer table includes four entries (records), and the records in the Customer table have two fields: Customer Number and Customer Name.

The following is the key list for the Customer table.

KEY	KEY TYPE	DEFINITION
1	Primary	Customer Number
2	Secondary	Customer Name

When you sort by the primary key, the Customer table resembles the following table.

CUSTOMER NUMBER	CUSTOMER NAME

CUSTOMER NUMBER	CUSTOMER NAME
001	Customer C
002	Customer A
003	Customer B
004	Customer C

If you select the secondary key for sorting, then the order is based on the contents of the Customer Name field. Because the contents of these fields are not unique, the records must be sub-sorted according to the primary key.

CUSTOMER NAME	CUSTOMER NUMBER
Customer A	002
Customer B	003
Customer C	001
Customer C	004

> **NOTE**
> The two records that have the same Customer Name value are sorted by Customer Number.

How keys affect performance

Searching for specific data is easier if several keys have been defined and maintained for the table that holds the desired data. The indexes for each key provide specific views that enable quick, flexible searches. There are advantages and disadvantages to using many keys, as demonstrated in the following table.

IF YOU	PERFORMANCE IMPROVES WHEN YOU	PERFORMANCE SLOWS WHEN YOU
Increase the number of secondary keys that are marked as active.	Retrieve data in several different sorting sequences because the data is already sorted.	Enter data because indexes for each secondary key must be maintained.
Decide to use only a few keys.	Enter data because a minimal number of indexes are maintained.	Retrieve data. You may have to define or reactivate the secondary keys to get the appropriate sorting. Depending on the size of the database, this can take some time, because the index must be rebuilt.

The decision whether to use a few or many keys is not easy. The choice of appropriate keys and the number of active keys to use should be the best compromise between maximizing the speed of data retrieval and maximizing the speed of data updates (operations that insert, delete, or modify data). In general, it may be worthwhile to deactivate complex keys if they are rarely used.

The overall speed depends on the following factors:

- Size of the database.
- Number of active keys.
- Complexity of the keys.
- Number of records in your tables.
- Speed of your computer and its hard disk.

Defining new keys

You define keys in AL code of a table object. To define keys, add the `keys` keyword after the `fields` definition, and then add a `key` keyword for each key:

```
keys
{
    key(Name; Fields)
    {

    }
    key(Name; Fields)
    {

    }
}
```

Replace `Name` with descriptive text that you want to use to identify the key. Replace `Field` with the name of a field that you want to use as the key. If you want to include multiple fields in a single key, separate each field with a comma.

The first `key` keyword defines the primary key. Subsequent `key` keywords define secondary keys.

Key properties

There are several properties that configure the behavior of a key, such as the Enabled, Clustered, and Unique properties:

```
keys
{
    key(PrimaryKey; ID)
    {
        Clustered = true;
    }
    key(CustomerInfo; Name,Address,City)
    {
        Unique = true;
    }
   key(Currency; Currency Code)
    {
        Enabled = false;
    }
}
```

For a more information about the different key properties, see Key Properties.

Restrictions on key modifications

When developing a new version of an extension, be aware of the following restrictions to avoid schema synchronization errors that prevent you from publishing the new version:

- Do not delete primary keys.
- Do not add or remove primary key fields, nor change their order.
- Do not change properties of existing primary keys.
- Do not add additional unique keys.
- Do not add additional clustered keys.
- Do not add keys that are fields of the base table.

See Also

Key Properties Tables Overview
Table Object
Table Extension Object
SystemId Field

Page Object

2/17/2021 • 2 minutes to read • Edit Online

Pages are the main way to display and organize visual data in Dynamics 365 Business Central. They are the primary object that a user will interact with and have a different behavior based on the type that you choose. Pages are designed independently of the device they are to be rendered on, and in this way the same page can be reused across phone, tablet, and web clients.

The structure of a page is hierarchical and breaks down in to three sections. The first block contains metadata for the overall page; the type of the page and the source table it is showing data from. The next section; the layout, describes the visual parts on the page. The final section details the actions that are published on the page.

When developing a solution for Dynamics 365 Business Central, you will follow the code layout for a page as shown in the page example below, but for more details on the individual controls and properties that are available, see Page Property Overview.

If you want to, for example, add functionality to a page that already exists in Business Central, you can create a page extension object that changes an existing page object. For more information, see Page Extension Object. Depending on how much you want to change on an existing page, you can also create a page customization object, which offers modifications on actions and layout. For more information, see Page Customization Object.

> **IMPORTANT**
> Only pages with the Extensible Property set to **true** can be extended.

> **NOTE**
> Extension objects can have a name with a maximum length of 30 characters.

Snippet support

Typing the shortcut `tpage` will create the basic layout for a page object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Views

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. For more information, see Views.

Page example

```
page 50101 SimpleCustomerCard
{
    PageType = Card;
    SourceTable = Customer;
    ContextSensitiveHelpPage = 'my-feature';

    layout
    {
        area(content)
        {
            group(General)
            {
                field("No."; "No.")
                {
                    ApplicationArea = All;
                    CaptionML = ENU = 'Hello';

                    trigger OnValidate()
                    begin
                        if "No." < '' then
                            Message('Number too small')
                    end;
                }

                field(Name; Name)
                {
                    ApplicationArea = All;
                }
                field(Address; Address)
                {
                    ApplicationArea = All;
                }
            }
        }
    }
    actions
    {
        area(Navigation)
        {
            action(NewAction)
            {
                ApplicationArea = All;
                RunObject = codeunit "Document Totals";
            }
        }
    }
}
```

See Also

AL Development Environment
Views
Adding Help Links from Pages, Reports, and XMLports
Page Extension Object
Page, Page Fields, and Page Extension Properties
Page Properties
Developing Extensions
Configure Context-Sensitive Help

Page Extension Object

2/17/2021 • 4 minutes to read • Edit Online

The page extension object extends a Dynamics 365 Business Central page object and adds or overrides the functionality.

The structure of a page is hierarchical and breaks down into three sections. The first block contains metadata for the overall page; the type of the page and the source table it is showing data from. The next section; the layout, describes the visual parts on the page. The final section details the actions that are published on the page.

For more information about the Page and Page Extension objects, see Pages Overview.

IMPORTANT

Only pages with the Extensible Property set to **true** can be extended.

NOTE

Extension objects can have a name with a maximum length of 30 characters.

IMPORTANT

The API page type should not be extended by creating a page extension object. Instead, create a new API by adding a page object.

NOTE

Modifying actions in Cue groups on page extensions is not supported.

Snippet support

Typing the shortcut `tpageext` will create the basic layout for a page extension object when using the AL Language extension in Visual Studio Code.

TIP

Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Views

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. For more information, see Views.

Using keywords to place actions and controls

You can use the following keywords in the `layout` section to place and move fields and groups on the page extension. Similarly, in the `actions` section, you use these keywords to place actions in the ribbon.

KEYWORDS	SYNTAX	APPLIES TO
`addfirst`	`addfirst(Anchor)`	**Anchor**: areas and groups
`addlast`	`addlast(Anchor)`	**Anchor**: areas and groups
`addafter`	`addafter(Anchor)`	**Anchor**: controls, actions, and groups
`addbefore`	`addbefore(Anchor)`	**Anchor**: controls, actions, and groups
`movefirst`	`movefirst(Anchor; Target1, Target2)`	**Anchor**: area, group **Target**: list of actions or list of controls
`movelast`	`movelast(Anchor; Target1, Target2)`	**Anchor**: area, group **Target**: list of actions or list of controls
`moveafter`	`moveafter(Anchor; Target1, Target2)`	**Anchor**: controls, actions, and groups **Target**: list of actions or list of controls
`movebefore`	`movebefore(Anchor; Target1, Target2)`	**Anchor**: controls, actions, and groups **Target**: list of actions or list of controls
`modify`	`modify(Target)`	**Target**: controls, actions, and groups

Example

To modify the existing fields and groups on a page, you use the `modify` keyword. See the code snippet below for `addlast`, `modify` and `action` syntax. In the following example, the `actions` section creates a new group in the ribbon and places it last in the `Creation` group.

```
pageextension 70000020 CustomerCardExtension extends "Customer Card"
{
    layout
    {
        // Adding a new control field 'ShoeSize' in the group 'General'
        addlast(General)
        {
            field("Shoe Size"; ShoeSize)
            {
                Caption = 'Shoe size';

                trigger OnValidate();
                begin
                    if ShoeSize < 10 then
                        Error('Feet too small');
                end;
            }
        }

        // Modifying the caption of the field 'Address 2'
        modify("Address 2")
        {
            Caption = 'New Address 2';
        }

        // Moving the two fields 'CreditLimit' and 'CalcCreditLimitLCYExpendedPct'
        // to be the first ones in the 'Balance' group.
        movefirst(Balance; CreditLimit, CalcCreditLimitLCYExpendedPct)
    }
    actions
    {
        // Adding a new action group 'MyNewActionGroup' in the 'Creation' area
        addlast(Creation)
        {
            group(MyNewActionGroup)
            {
                action(MyNewAction)
                {
                    Caption = 'My New Action';

                    trigger OnAction();
                    begin
                        Message('My message');
                    end;
                }
            }
        }
    }
}

tableextension 70000020 CustomerTableExtension extends Customer
{
    fields
    {
        // Adding a new table field in the 'Customer' table
        field(50100; ShoeSize; Integer) { }
    }
}
```

Page extension examples

In the following example, we use a table extension to extend the Customer table with a new field named `ShoeSize` of the datatype Integer. Then we create a page extension object that extends the Customer Card page object by adding a field control `ShoeSize` to the `General` group on the page. The field control is added as the

last control in the group using the `addlast` method. The example also illustrates how to add a display-only control to the page. In the actions area, you can see what the syntax looks like for actions that execute triggers and actions that run objects.

```
tableextension 50115 RetailWinterSportsStore extends Customer
{
    fields
    {
        field(50116;ShoeSize;Integer)
        {
            Caption = 'ShoeSize';

            trigger OnValidate();
            begin
                if (rec.ShoeSize < 0) then
                begin
                   message('Shoe size not valid: %1', rec.ShoeSize);
                end;
            end;
        }
    }

    procedure HasShoeSize() : Boolean;
    begin
        exit(ShoeSize <> 0);
    end;

    trigger OnBeforeInsert();
    begin
        if not HasShoeSize then
            ShoeSize := Random(42);
    end;
}

pageextension 50110 CustomerCardExtension extends "Customer Card"
{
    layout
    {
        addlast(General)
        {
            // control with underlying datasource
            field("Shoe Size"; ShoeSize)
            {
                ApplicationArea = All;

                trigger OnValidate();
                begin
                    if ShoeSize < 10 then
                        Error('Feet too small');
                end;
            }

            // display-only control (without underlying datasource)
            field(ShoesInStock; 10)
            {
                ApplicationArea = All;
                Caption = 'Shoes in stock';

            }
        }

        modify("Address 2")
        {
            Caption = 'New Address 2';
        }
    }
```

```
    actions
    {
        addlast(Creation)
        {
            group(MyActionGroup)
            {
                Action(MyAction1)
                {
                    ApplicationArea = All;
                    Caption = 'Hello!';

                    trigger OnAction();
                    begin
                        Message('My message');
                    end;
                }

                Action(MyAction2)
                {
                    ApplicationArea = All;

                    // Run page to test how actions work
                    RunObject = page "Absence Registration";
                }
            }
        }
    }
}
```

You can reference Report and XMLPort objects and use these objects in the **RunObject** property, as well as, declare variables of the types **Report** and **XMLPort** and call AL methods on them. This page extension object extends the Customer List page object by adding two actions; the first action calls the **Customer - List** report, the second action calls the **Export Contact** XMLPort.

```
pageextension 50114 AddCustomerReport extends "Customer List"
{
    actions
    {
        AddLast("&Customer")
        {
            action("Customer List Report")
            {
                trigger OnAction();
                var
                    rep : Report "Customer - List";
                begin
                    rep.Run;
                end;
            }

            action("Export Contact List")
            {
                trigger OnAction();
                var
                    xml : XmlPort "Export Contact";
                begin
                    xml.Run;
                end;
            }
        }
    }
}
```

See Also

Page Object
Views
Page, Page Fields, and Page Extension Properties
Extending Pages Previously Based on the Date Virtual Table Developing Extensions
AL Development Environment

Page Customization Object

2/17/2021 • 2 minutes to read • Edit Online

The page customization object in Dynamics 365 Business Central allows you to add changes to the layout and actions on page that are accessible for a profile. See Using keywords to place actions and controls for how to place actions and controls on a page customization object.

The page customization object has more restrictions than the page extension object; when you define a new page customization object, you cannot add variables, procedures, or triggers.

> **NOTE**
> A single page customization can be used with multiple profiles within the same extension. Page customizations only apply to the RoleCenters they are specified for. In order to view or changes the RoleCenters in the client, go to **My Settings** > **Role Center**.

> **NOTE**
> Extension objects can have a name with a maximum length of 30 characters.

> **NOTE**
> Modifying actions in Cue groups on page extensions is not supported.

> **NOTE**
> `showMyCode` does not apply to page customizations. Page customizations defined in an extension with `showMyCode` set to `false` can still be copied using Designer.

Snippet support

Typing the shortcut `tpagecust` will create the basic layout for a page customization object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Views

Views in Dynamics 365 Business Central are used on list pages to define a different view of the data on a given page. Views can be defined for Pages, Page Extensions, and Page Customization. For more information, see Views.

Page customization example

The following page customization example `MyCustomization` makes changes to **Customer List**. By using the `moveafter` method, `Blanket Orders` is moved after the `Orders` action item. And the `modify` method is used to hide the `NewSalesBlanketOrder` action item.

```
profile TheBoss
{
    Description = 'The Boss';
    RoleCenter = "Business Manager Role Center";
    Customizations = MyCustomization;
    Caption = 'Boss';
}

pagecustomization MyCustomization customizes "Customer List"
{
    actions
    {
        moveafter(Orders; "Blanket Orders")

        modify(NewSalesBlanketOrder)
        {
            Visible = false;
        }

    }
}
```

You can use the same page customization on another profile within the same extension package by referencing its name from the profile definition, for example:

```
profile TheSalesman
{
    ProfileDescription = 'The Boss';
    RoleCenter = "Sales Manager Role Center";
    Customizations = MyCustomization;
    Caption = 'Salesman';
}
```

See Also

Developing Extensions
AL Development Environment
Page Object
Page Extension Object
Views
Page, Page Fields, and Page Extension Properties

Report Object

2/17/2021 • 3 minutes to read • Edit Online

Reports are used to print or display information from a database. You can use a report to structure and summarize information, and to print documents, such as sales quotes and invoices.

Creating a report consists of two primary tasks; the first task is to create the underlying data model and the next is to define the visual layout that displays the data. The report object defines the underlying data model and specifies which database tables and fields to pull data from. When the report is run, that data is displayed in a specified layout; the visual layout, which determines the content and format of a report when it is viewed and printed.

For more information about defining database tables and fields, see Defining a Report Dataset. For more information about the Report data type, see Report Data Type.

You build the layout of a report by arranging data items and columns, and specifying the general format, such as text font and size. There are two types of report layouts; client report definition, also called RDL layouts and Word layouts. RDL layouts are defined in Visual Studio Report Designer or Microsoft SQL Server Reporting Services Report Builder. Word layouts are created using Word. Word layouts are based on a Word document that includes a custom XML part representing the report dataset.

> **NOTE**
> Extension objects can have a name with a maximum length of 30 characters.

Snippet support

Typing the shortcut `treport` will create the basic layout for a report object when using the AL Language extension in Visual Studio Code.

> **TIP**
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Report example

The following example is a report that prints the list of customers. The report object defines a dataset of columns from the Customer table. For more information on creating a Word Layout report, see Creating a Report.

```
report 50103 "Customer List"
{
  CaptionML=ENU='Customer List';
  DefaultLayout = RDLC; // if Word use WordLayout property
  RDLCLayout = 'MyRDLReport.rdl';

  dataset
  {
    dataitem(Customer;Customer)
    {
      RequestFilterFields="No.","Search Name","Customer Posting Group";
```

```
        column(COMPANYNAME;COMPANYNAME)
        {
        }
        column(CurrReport_PAGENO;Customer."no.")
        {
        }
        column(Customer_TABLECAPTION_CustFilter;TABLECAPTION + ': ' + CustFilter)
        {
        }
        column(CustFilter;CustFilter)
        {
        }
        column(Customer_No;"No.")
        {
        }
        column(Customer_Customer_Posting_Group;"Customer Posting Group")
        {
        }
        column(Customer_Customer_Disc_Group;"Customer Disc. Group")
        {
        }
        column(Customer_Invoice_Disc_Code;"Invoice Disc. Code")
        {
        }
        column(Customer_Customer_Price_Group;"Customer Price Group")
        {
        }
        column(Customer_Fin_Charge_Terms_Code;"Fin. Charge Terms Code")
        {
        }
        column(Customer_Payment_Terms_Code;"Payment Terms Code")
        {
        }
        column(Customer_Salesperson_Code;"Salesperson Code")
        {
        }
        column(Customer_Currency_Code;"Currency Code")
        {
        }
        column(Customer_Credit_Limit_LCY;"Credit Limit (LCY)")
        {
          DecimalPlaces=0:0;
        }
        column(Customer_Balance_LCY;"Balance (LCY)")
        {
        }
        column(CustAddr_1;CustAddr[1])
        {
        }
        column(CustAddr_2;CustAddr[2])
        {
        }
        column(CustAddr_3;CustAddr[3])
        {
        }
        column(CustAddr_4;CustAddr[4])
        {
        }
        column(CustAddr_5;CustAddr[5])
        {
        }
        column(Customer_Contact;Contact)
        {
        }
        column(Customer_Phone_No;"Phone No.")
        {
        }
        column(CustAddr_6;CustAddr[6])
        {
```

```
      {
      }
      column(CustAddr_7;CustAddr[7])
      {
      }
      column(Customer_ListCaption;Customer_ListCaptionLbl)
      {
      }
      column(CurrReport_PAGENOCaption;CurrReport_PAGENOCaptionLbl)
      {
      }
      column(Customer_NoCaption;FIELDCAPTION("No."))
      {
      }
      column(Customer_Customer_Posting_GroupCaption;Customer_Customer_Posting_GroupCaptionLbl)
      {
      }
      column(Customer_Customer_Disc_GroupCaption;Customer_Customer_Disc_GroupCaptionLbl)
      {
      }
      column(Customer_Invoice_Disc_CodeCaption;Customer_Invoice_Disc_CodeCaptionLbl)
      {
      }
      column(Customer_Customer_Price_GroupCaption;Customer_Customer_Price_GroupCaptionLbl)
      {
      }
      column(Customer_Fin_Charge_Terms_CodeCaption;FIELDCAPTION("Fin. Charge Terms Code"))
      {
      }
      column(Customer_Payment_Terms_CodeCaption;Customer_Payment_Terms_CodeCaptionLbl)
      {
      }
      column(Customer_Salesperson_CodeCaption;FIELDCAPTION("Salesperson Code"))
      {
      }
      column(Customer_Currency_CodeCaption;Customer_Currency_CodeCaptionLbl)
      {
      }
      column(Customer_Credit_Limit_LCYCaption;FIELDCAPTION("Credit Limit (LCY)"))
      {
      }
      column(Customer_Balance_LCYCaption;FIELDCAPTION("Balance (LCY)"))
      {
      }
      column(Customer_ContactCaption;FIELDCAPTION(Contact))
      {
      }
      column(Customer_Phone_NoCaption;FIELDCAPTION("Phone No."))
      {
      }
      column(Total_LCY_Caption;Total_LCY_CaptionLbl)
      {
      }

      trigger OnAfterGetRecord();
      begin
        CalcFields("Balance (LCY)");
        FormatAddr.FormatAddr(
          CustAddr,Name,"Name 2",'',Address,"Address 2",
          City,"Post Code",County,"Country/Region Code");
      end;

    }
  }

  requestpage
  {
    SaveValues=true;
    ContextSensitiveHelpPage = 'my-feature';
    layout
```

```
    layout
    {
    }

    actions
    {
    }
  }

  labels
  {
      LabelName = 'LabelText', Comment = 'Foo', MaxLength = 999, Locked = true;
  }

  trigger OnPreReport();
  var
    CaptionManagement : Codeunit 42;
  begin
    CustFilter := CaptionManagement.GetRecordFiltersWithCaptions(Customer);
  end;

  var
    FormatAddr : Codeunit 365;
    CustFilter : Text;
    CustAddr : ARRAY [8] OF Text[50];
    Customer_ListCaptionLbl : Label 'Customer - List';
    CurrReport_PAGENOCaptionLbl : Label 'Page';
    Customer_Customer_Posting_GroupCaptionLbl : Label 'Customer Posting Group';
    Customer_Customer_Disc_GroupCaptionLbl : Label 'Cust./Item Disc. Gr.';
    Customer_Invoice_Disc_CodeCaptionLbl : Label 'Invoice Disc. Code';
    Customer_Customer_Price_GroupCaptionLbl : Label 'Price Group Code';
    Customer_Payment_Terms_CodeCaptionLbl : Label 'Payment Terms Code';
    Customer_Currency_CodeCaptionLbl : Label 'Currency Code';
    Total_LCY_CaptionLbl : Label 'Total (LCY)';
}
```

See Also

Request Pages
Creating an RDL Layout Report
Creating a Word Layout Report
Adding Help Links from Pages, Reports, and XMLports
Page Extension Object
Page Properties
Developing Extensions
AL Development Environment

Profile Object

2/17/2021 • 2 minutes to read • Edit Online

The profile object in Dynamics 365 Business Central allows you to build an individual experience for each user profile. The Profile object performs a validation to check whether the specified role center page exists, and page customization objects exists, when you define a new profile object. On a page customization you can add changes to the page layout, and actions; but you cannot add variables, procedures, or triggers.

> **NOTE**
>
> Page customizations only apply to the RoleCenter they are specified for. In order to see them, in Dynamics 365 Business Central under **My Settings**, **Role Center** change to the specific RoleCenter that a page customization is defined for.

> **NOTE**
>
> Extension objects can have a name with a maximum length of 30 characters.

> **NOTE**
>
> `showMyCode` does not apply to profiles. Profiles defined in an extension with `showMyCode` set to `false` can still be copied using Designer.

Snippet support

Typing the shortcut `tprofile` will create the basic layout for a profile object when using the AL Language extension in Visual Studio Code.

> **TIP**
>
> Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Profile example

The following profile object example creates a profile for the `MyRoleCenter` Role Center, which is available in the **Role Explorer** in the UI and available to end-users. The profile also depends on the customization `MyCustomization` and modifies the layout of the **Customer List** to make the `Name` field invisible using the `modify` method. For more information, see Profile Properties.

```
profile MyProfile
{
    Description = 'Some internal comment that only the Dev can see';
    Caption = 'My User-friendly Name';
    ProfileDescription = 'A detailed description of who is this profile for, why/how to use it (etc)';
    RoleCenter = MyRoleCenter;
    Enabled = true;
    Promoted = true;
    Customizations = MyCustomization;
}

pagecustomization MyCustomization customizes "Customer List"
{
    layout
    {
        modify(Name)
        {
            Visible = false;
        }
    }
}
```

See Also

AL Development Environment
Developing Extensions
Pages Overview
Page Customization Object

Codeunit Object

2/17/2021 • 2 minutes to read • Edit Online

A codeunit is a container for AL code that you can use in many application objects. You typically implement business logic in codeunits and call the codeunit from the object that needs to perform that specific logic.

Snippet support

Typing the shortcut `tcodeunit` will create the basic layout for a codeunit object when using the AL Language extension in Visual Studio Code.

Codeunit example

This codeunit example checks whether a given customer has registered a shoe size. If not, the customer is assigned a shoe size of 42.

The codeunit can be used both as a direct call to `codeunit.run(customer)` or as a call to the procedure inside the codeunit `createcustomer.CheckSize(customer)`.

```
codeunit 50113 CreateCustomer
{
    TableNo = Customer;
    trigger OnRun();
    begin
        CheckSize(Rec);
    end;
    procedure CheckSize(var Cust : Record Customer)
    begin
        if not Cust.HasShoeSize() then
            Cust.ShoeSize := 42;
    end;
}
```

See Also

Developing Extensions
Table Extension Object
Page Extension Object
AL Development Environment
XML Comments in Code

Query Object

2/17/2021 • 3 minutes to read • Edit Online

Business Central query objects enable you to retrieve records from one or more tables and then combine the data into rows and columns in a single dataset. Query objects can also perform calculations on data, such finding the sum or average of all values in a column of the dataset.

There are two types of query objects: normal and API. This article describes normal query objects, which can be used to display data in the user interface. API query objects are used to generate web service endpoints and cannot be displayed in the user interface. For information about creating a query of the type API, see API Query Type.

Creating a query object

A query object is comprised mainly of two different types of elements: dataitems and columns. A dataitem specifies the table to retrieve records from. A column specifies a field of the table to include in the resulting dataset of a query. The basic steps to create a query object are as follows:

1. Add the `query` keyword, followed by the `elements` control.
2. Build the dataset by adding `dataitem` controls and `column` controls within the `elements` control.

   The hierarchy of the `dataitem` and `column` controls is important because it will determine the sequence in which data items are linked, which in turn will control the results. Working from top-to-bottom, you start by adding the `dataitem` control for first table that you want in the dataset, then add `column` controls for each table field that you want to include in the dataset. For the next table, you add another `dataitem` control that is embedded within the first `dataitem` control, then add `column` controls as needed. You continue this pattern for additional tables and fields.
3. When you have specified the dataitem and column elements, create links and joins between the `dataitem` elements.

   Dataitem links and joins determine which records to include in the dataset based on the values of a common field between dataitems. You set a link between one or more fields of the dataitem tables with the DataItemLink Property and you define the type of the link using the SQLJoinType Property. Both properties must be set on the lower dataitem of the query object. For more information, see Linking and Joining Data Items.

The following shows the basic structure of a query object.

```
query ID Name
{

    elements
    {
        dataitem(DataItem1; Table1)
        {
            column(Column1; Field1)
            {
            }
            column(Column2; Field2)
            {
            }
            dataitem(DataItem2; Table2)
            {
                // Sets a link between FieldY of Table2 and FieldX of Table1.
                DataItemLink = FieldY = DataItem1.FieldX;
                //The dataset contains records from Table1 and Table2 where a match is found between FieldY
and FieldX.
                SqlJoinType = InnerJoin;

                column(Column1; Field1)
                {
                }
                dataitem(DataItem3; Table3)
                {
                    DataItemLink = FieldZ = DataItem2.FieldY;
                    SqlJoinType = InnerJoin;
                    column(Column1; Field1)
                    {
                    }
                }
            }
        }
    }
}
```

NOTE

Extension objects can have a name with a maximum length of 30 characters.

Snippet support

Typing the shortcut `tquery` will create the basic layout for a Query object when using the AL Language extension in Visual Studio Code.

TIP

Use **Ctrl+Space** to trigger IntelliSense and get assistance on code completion, parameter info, quick info, and member lists.

Query example

The following example shows a query that displays a list of customers with sales and profit figures. The query primarily retrieves fields from the **Customer** table, but also displays fields from the **Salesperson Purchaser** and **Country Region** tables.

The query also uses the DataItemLink property to create a link between the **Customer** table, **Salesperson**

Code field and the **Salesperson Purchaser** table, **Code** fields and a link between the **Customer** table, **Country/Region Code** field and the **Country/Region** table, **Code** field.

```
query 50102 "Top Customer Overview"
{
    QueryType = Normal;
    Caption = 'Top Customer Overview';

    elements
    {
        dataitem(Customer; Customer)
        {
            column(Name; Name)
            {
            }
            column(No; "No.")
            {
            }
            column(Sales_LCY; "Sales (LCY)")
            {
            }
            column(Profit_LCY; "Profit (LCY)")
            {
            }
            column(Country_Region_Code; "Country/Region Code")
            {
            }
            column(City; City)
            {
            }
            column(Global_Dimension_1_Code; "Global Dimension 1 Code")
            {
            }
            column(Global_Dimension_2_Code; "Global Dimension 2 Code")
            {
            }
            column(Salesperson_Code; "Salesperson Code")
            {
            }
            dataitem(Salesperson_Purchaser; "Salesperson/Purchaser")
            {
                DataItemLink = Code = Customer."Salesperson Code";
                column(SalesPersonName; Name)
                {
                }
                dataitem(Country_Region; "Country/Region")
                {
                    DataItemLink = Code = Customer."Country/Region Code";
                    column(CountryRegionName; Name)
                    {
                    }
                }
            }
        }
    }
}
```

IMPORTANT

You cannot run a query that gets data from both the application database and the business data database. This also applies to single-tenant deployments so that you do not have to rewrite queries if you decide to export the application. For a description of which tables are considered part of the application database, see Separating Application Data from Business Data.

See Also

Linking and Joining Data Items
Aggregating Data in Query Objects
Query Objects and Performance
Query Properties
Developing Extensions
AL Development Environment
API Query Type

XMLport Object

2/17/2021 • 2 minutes to read • Edit Online

XMLports are used to export and import data between an external source and Dynamics 365 Business Central. Sharing data between different computer systems is seamless when it is shared in an XML format. Working with XML files can be tedious so the details of how the XML file is handled are encapsulated in XMLports.

To use an XMLport to import or export data, you first create an XMLport object. Once created, you can run the XMLport from a page or codeunit object.

You can design XMLports to include a request page, which is a dialog box that enables the user to set a filter on the data, sort the data, or choose whether to export or import the data. For more information about request pages, see Request Pages.

XMLport example

The following example shows a page extension of the **Permission Sets** page that adds an action to the specified page calling the XMLport **ExportPermissionSet**. The XMLport exports the permission set data to an XML file.

```
pageextension 50111 PermissionSetExporter extends "Permission Sets"
{
    actions
    {
        addafter(Permissions)
        {
            action(ExportPermissionSet)
            {
                Promoted = true;
                PromotedCategory = New;
                trigger OnAction();
                begin
                    Xmlport.Run(50112, false, false);
                end;
            }
        }
    }
}

xmlport 50112 ExportPermissionSet
{
    Format = xml;

    schema
    {
        textelement(PermissionSets)
        {
            tableElement(PSet; "Aggregate Permission Set")
            {
                SourceTableView = WHERE ("App Name" = FILTER (<> ''));
                XmlName = 'PermissionSet';
                fieldattribute(RoleID; pset."Role ID") { }
                fieldattribute(RoleName; pset.Name) { }
                tableelement(P; "Tenant Permission")
                {
                    XmlName = 'Permission';
                    LinkTable = pset;
                    LinkFields = "Role ID" = FIELD ("Role ID");
```

```
                    textelement(ObjectType)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Object Type";
                            ObjectType := format(int);
                        end;
                    }
                    textelement(ObjectID)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Object ID";
                            ObjectID := format(int);
                        end;
                    }
                    textelement(ReadPermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Read Permission";
                            ReadPermission := format(int);
                        end;
                    }
                    textelement(InsertPermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Insert Permission";
                            InsertPermission := format(int);
                        end;
                    }
                    textelement(ModifyPermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Modify Permission";
                            ModifyPermission := format(int);
                        end;
                    }
                    textelement(DeletePermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Delete Permission";
                            DeletePermission := format(int);
                        end;
                    }
                    textelement(ExecutePermission)
                    {
                        trigger onbeforePassvariable();
                        var
                            int: Integer;
                        begin
                            int := p."Execute Permission";
                            ExecutePermission := format(int);
                        end;
```

```
                    }
                    textelement(SecurityFilter)
                    {
                        trigger onbeforePassvariable();
                        begin
                            SecurityFilter := format(p."Security Filter");
                        end;
                    }
                }
            }
        }
    }
}
```

See Also

Developing Extensions
AL Development Environment
XMLport Overview
Using Namespaces with XMLports
Page Extension Object
Report Object

Control Add-In Object

2/17/2021 • 4 minutes to read • Edit Online

The control add-in object allows you to add custom functionality to Dynamics 365 Business Central. A control add-in is a custom control, or visual element, for displaying and modifying data within an iframe or a page. For example, a control add-in can display the content of a webpage, visualize data as a chart or on a map, or host a custom web application. Control add-ins can exchange data with the Dynamics 365 server on various data types and respond to user interaction to raise events that execute additional AL code.

Control add-in properties

In the control add-in definition, you must set the `Scripts` property to include scripts in the control add-in. The scripts could be local files in the package or references to external files using the HTTP or the HTTPS protocol. With the `StartupScript` property, you can call a special script that runs when the page you've implemented the control add-in on, is loaded. These settings initialize the control add-in. With the `Images` and `StyleSheet` properties, you can specify additional styling to the control add-in. For more information about some of the control add-in properties, see:

- Images
- Scripts
- StartupScript
- StyleSheets
- RecreateScript
- RefreshScript

Sizing of the control add-in

Control add-ins can either have fixed dimensions or dynamically adapt to the available space on the screen. By controlling the sizing of an add-in, you ensure the add-in and the surrounding content on the page remain optimal on smaller display targets such as the phone or when users resize the browser. The following properties are available for you to specify how the sizing of the control add-in should behave.

- HorizontalShrink
- HorizontalStretch
- MinimumHeight
- MinimumWidth
- MaximumHeight
- MaximumWidth
- RequestedHeight
- RequestedWidth
- VerticalShrink
- VerticalStretch

Control add-in considerations and limitations

Designing control add-ins that provide the best possible experience can require some additional planning, design, and implementation. The following considerations and limitations will help you design add-ins that look and feel seamlessly integrated with both Dynamics 365 Business Central online and on-premises.

- Respond to touch events so that mobile users or users on devices supporting touch input can also use the add-in.
- Design content that is responsive and can flow, resize, or reorganize naturally based on the available space.
- Consider the accessibility needs of users, for example by implementing keyboard access and support for screen readers.
- Use the Style guidelines to apply a choice of colors, typefaces, and font sizes that match that of Dynamics 365 Business Central. For more information, see Control Add-in Style Guide.
- Provide language translation and other localizations that match the current user language in Dynamics 365 Business Central.
- In extensions for Business Central online, don't reference font files in stylesheets, because the fonts won't display in client. Instead, do one of the following:
  - Reference the font files from some other source such as a public or private CDN.
  - Base64 encode the fonts and include the encoded fonts in the CSS file.

Control add-in syntax example

The following control add-in syntax shows how to implement small customizations of the layout and functionality of a page.

```
// The controladdin type declares the new add-in.
controladdin SampleAddIn
{
    // The Scripts property can reference both external and local scripts.
    Scripts = 'https://cdnjs.cloudflare.com/ajax/libs/knockout/3.4.2/knockout-debug.js',
              'main.js';
    // The StartupScript is a special script that the web client calls once the page is loaded.
    StartupScript = 'startup.js';

    // Specifies the StyleSheets that are included in the control add-in.
    StyleSheets = 'skin.css';

    // Specifies the Images that are included in the control add-in.
    Images = 'image.png';

    // The procedure declarations specify what JavaScript methods could be called from AL.
    // In main.js code, there should be a global function CallJavaScript(i,s,d,c)
{Microsoft.Dynamics.NAV.InvokeExtensibilityMethod('CallBack', [i, s, d, c]);}
    procedure CallJavaScript(i: integer; s: text; d: decimal; c: char);

    // The event declarations specify what callbacks could be raised from JavaScript by using the webclient
API:
    // Microsoft.Dynamics.NAV.InvokeExtensibilityMethod('CallBack', [42, 'some text', 5.8, 'c'])
    event Callback(i: integer; s: text; d: decimal; c: char);
}
```

The `controladdin` object is then invoked as a `usercontrol` on a page called **PageWithAddIn**.

```
page 50130 PageWithAddIn
{
    layout
    {
        area(Content)
        {
             // The control add-in can be placed on the page using usercontrol keyword.
            usercontrol(ControlName; SampleAddIn)
            {

                ApplicationArea = All;

                 // The control add-in events can be handled by defining a trigger with a corresponding
name.
                trigger Callback(i: integer; s: text; d: decimal; c: char)
                begin
                    Message('Got from js: %1, %2, %3, %4', i, s, d, c);
                end;
            }
        }
    }

    actions
    {
        area(Creation)
        {
            action(CallJavaScript)
            {
                ApplicationArea = All;

                trigger OnAction();
                begin

                    // The control add-in methods can be invoked via a reference to the usercontrol.
                    CurrPage.ControlName.CallJavaScript(5, 'text', 6.3, 'c');
                end;
            }
        }
    }
}
```

Loading static resources using AJAX requests

You can design a control add-in to load static resources from the add-in package by using AJAX requests. For example, the control add-in could load HTML content and inject it into add-in's HTML structure. In this case, you must use the `withCrendentials` property set to `true` in the AJAX request. Otherwise, the request won't contain the necessary context and important cookies required by the Business Central service, and it may fail in production. This concept is illustrated in the following examples.

Wrong:

```
$.get(url).done(function(response) { } );
```

Correct:

```
$.ajax({
     url: url,
     xhrFields: {
          withCredentials: true
    }
)).done(function(data) {
    $("#controlAddIn).text(data);
});
```

See Also

AL Development Environment
Developing Extensions
Asynchronous Considerations for Control Add-ins
InvokeExtensibility Method
GetImageResource Method
GetEnvironment Method
Pages Overview
Page Extension Object
Page Customization Object

Data Types and Methods in AL

2/17/2021 • 9 minutes to read • Edit Online

The following data types are available as part of the AL Language. Each data type has various methods that support it. For more information about a data type and its methods, select a link in the table.

TYPE	DESCRIPTION
BigInteger	Stores very large whole numbers that range from -9,223,372,036,854,775,807 to 9,223,372,036,854,775,807.
BigText	Handles large text documents.
Blob	Is a complex data type. Variables of this data type differ from normal numeric and string variables in that BLOBs have a variable length. The maximum size of a BLOB(binary large object) is 2 GB.
Boolean	Indicates true or false.
Byte	Stores a single, 8-bit character as a value in the range 0 to 255. You can easily convert this data type from a number to a character and vice versa. This means you can use mathematical operators on Byte variables.
Char	Stores a single, 16-bit character as a value in the range 0 to 65535. You can convert this data type from a number to a character and vice versa. This means you can use mathematical operators on Char variables.
Code	Denotes a special type of string that is converted to uppercase and removes any trailing or leading spaces.
Codeunit	Is a container for AL code that you can use from other application objects.
CompanyProperty	Provides language support for company properties.
Database	Provides access to common database functionality.
Date	Denotes a date ranging from January 1, 1753 to December 31, 9999.
DateFormula	Represents a date formula that has the same capabilities as an ordinary input string for the CALCDATE Method (Date). The DateFormula data type is used to provide multilanguage capabilities to the CALCDATE Method (Date).
DateTime	Denotes a date and time ranging from January 1, 1753, 00:00:00.000 to December 31, 9999, 23:59:59.999. An undefined or blank DateTime is specified by 0DT.

TYPE	DESCRIPTION
Debugger	Enables communication with a debugger.
Decimal	Denotes decimal numbers ranging from -999,999,999,999,999.99 to +999,999,999,999,999.99.
Dialog	Represents a dialog window.
Dictionary	Represents an unordered collection of keys and values. The Dictionary data type is optimized for fast lookup of values.
DotNet	Represents an unspecified .NET type.
Duration	Represents the difference between two DateTimes. This value can be negative. It is stored as a 64-bit integer. The integer value is the number of milliseconds during the duration.
Enum	Represents the text content of an element or attribute.
ErrorInfo	Provides a structure for grouping information about an error.
FieldRef	Identifies a field in a table and gives you access to this field.
File	Represents a file.
FilterPageBuilder	Stores filter configurations for a filter page. A filter page is a dynamic page type that contains one or more filter controls that enables users to set filters on fields of the underlying tables.
Guid	Represents a 16 byte binary data type. This data type is used for the global identification of objects, programs, records, and so on. The important property of a GUID is that each value is globally unique. The value is generated by an algorithm, developed by Microsoft, which assures this uniqueness.
HttpClient	Provides a data type for sending HTTP requests and receiving HTTP responses from a resource identified by a URI.
HttpContent	Represents an HTTP entity body and content headers.
HttpHeaders	Is a collection of headers and their values.
HttpRequestMessage	Represents an HTTP request message.
HttpResponseMessage	Represents a HTTP response message including the status code and data.
InStream	Is a generic stream object that you can use to read from or write to files and BLOBs. You can define the internal structure of a stream as a flat stream of bytes. You can assign one stream to another. Reading from and writing to a stream occurs sequentially.

TYPE	DESCRIPTION
Integer	Stores whole numbers with values that range from -2,147,483,647 to 2,147,483,647.
IsolatedStorage	Provides data isolation for extensions.
Any	This data type can be substituted by any other data type.
JsonArray	Is a container for any well-formed JSON array. A default JsonArray contains an empty JSON array.
JsonObject	Is a container for any well-formed JSON object. A default JsonObject contains an empty JSON object.
JsonToken	Is a container for any well-formed JSON data. A default JsonToken object contains the JSON value of NULL.
JsonValue	Is a container for any well-formed fundamental JSON value. A default JsonValue is set to the JSON value of NULL.
KeyRef	Identifies a key in a table and the fields in this key.
Label	Denotes a string constant that can be optionally translated into multiple languages.
List	Represents a strongly typed list of ordered objects that can be accessed by index. Contrary to the Array data type, a List is unbounded, such that its dimension does not need to be specified upon declaration.
Media	Encapsulates media files, such as image .jpg and .png files, in application database tables. The Media data type can be used as a table field data type, but cannot be used as a variable or parameter. The Media data type enables you to import a media file to the application database and reference the file from records, making it possible to display the media file in the client user interface. You can also export media from the database to files and streams.
MediaSet	Encapsulates media, such as images, in application database tables.
ModuleDependencyInfo	Provides information about a dependent module.
ModuleInfo	Represents information about an application consumable from AL.
NavApp	Provides information about a NavApp.
None	Is used implicitly when a method does not return a value.
Notification	Provides a programmatic way to send non-intrusive information to the user interface (UI) in the Business Central Web client.

TYPE	DESCRIPTION
NumberSequence	Is a complex data type for creating and managing number sequences in the database.
SessionInformation	Is a complex data type for exposing Session information into AL.
Option	Denotes an option value. In the code snippet below, you can see how the Option data type is declared.
OutStream	Is a generic stream object that you can use to write to files and BLOBs.
Page	Contains a number of simpler elements called controls. Controls are used to display information to the user or receive information from the user.
ProductName	An application can have a full name, marketing name, and short name. The PRODUCTNAME functions enable you to retrieve these name variations.
Query	Enables you to retrieve data from multiple tables and combine the data in single dataset.
RecordId	Contains the table number and the primary key of a table.
RecordRef	References a record in a table.
Report	Is used to display, print, or process information from a database.
RequestPage	Is a page that is run before the report starts to execute. Request pages enable end-users to specify options and filters for a report.
Session	Represents a Microsoft Dynamics Business Central session.
SessionSettings	Is a complex data type for passing user personalization settings for a client session as an object. The object contains properties that correspond to the fields in the system table **2000000073 User Personalization**, including: App ID, Company, Language ID, Locale ID, Profile ID, Scope, and Time Zone. You can use the AL methods of the SessionSettings data type to get, set, and send the user personalization settings for the current client session.
String	Denotes a sequence of characters. It can be represented by a string literal, a text value or a code value.
System	Is a complex data type.
Record	Is a complex data type.
TaskScheduler	Is a complex data type for creating and managing tasks in the task scheduler, which runs codeunits at scheduled times.

TYPE	DESCRIPTION
TestAction	Represents a test action on a page.
TestField	Represents a testable field on a page.
TestFilter	Represents a test filter on a page.
TestFilterField	Represents the type of a field filter in a test filter on a page or on a request page.
TestPage	Represents a variable type that can be used to test Page Application Objects.
TestPart	Represents a variable type that can be used to test Page Application Objects of type Part.
TestRequestPage	Stores test request pages. A test request page part is a logical representation of a request page on a report. A test request page does not display a user interface (UI). The subtype of a test request page is the report whose request page you want to test.
Text	Denotes a text string.
TextConst	Denotes a multi-language string constant.
TextBuilder	Represents a lighweight wrapper for the .Net implementation of StringBuilder.
Time	Denotes a time ranging from 00:00:00.000 to 23:59:59.999. An undefined or blank time is specified by 0T.
Variant	Represents an AL variable object. The AL variant data type can contain many AL data types.
Version	Represents a version matching the format: Major.Minor.Build.Revision .
WebServiceActionContext	Represents an AL WebServiceActionContext.
XmlAttribute	Represents an XML attribute.
XmlAttributeCollection	Represents a collection of XML attributes.
XmlCData	Represents a CData section.
XmlComment	Represents an XML comment.
XmlDeclaration	Represents an XML declaration.
XmlDocument	Represents an XML document.
XmlDocumentType	Represents an XML document type.

TYPE	DESCRIPTION
XmlElement	Represents an XML element.
XmlNamespaceManager	Represents a namespace manager that can be used to resolve, add and remove namespaces to a collection. It also provides scope management for these namespaces.
XmlNameTable	Represents a table of atomized string objects.
XmlNode	Represents a XML node which can either be for instance an XML attribute, an XML element or a XML document.
XmlNodeList	Represents a collection of XML nodes.
Xmlport	XmlPorts are used to export or import data between an external source and a Microsoft Dynamics Business Central database.
XmlProcessingInstruction	Represents a processing instruction, which XML defines to keep processor-specific information in the text of the document.
XmlReadOptions	Represents the options configuring how XML is loaded from a data source.
XmlText	Represents the text content of an element or attribute.
XmlWriteOptions	Represents the options configuring how XML is saved.
Action	Represents the action that the user took on the page.
ClientType	Represents the type of the client executing the operation.
CommitBehavior	Specifies whether commit is allowed within the scope of the method.
DataClassification	Sets the classification of the data in the table or field.
DataScope	Identifies the scope of stored data in the isolated storage.
DefaultLayout	The default layout to be used by a report.
ErrorType	Represents the type of error.
ExecutionContext	Represents the context in which a session is running. In certain scenarios, for example during upgrade, the system will run a session in a special context for a limited time.
ExecutionMode	The execution mode of the current session.
FieldClass	Represents the type of a field class.
FieldType	Represents the type of a table field.

TYPE	DESCRIPTION
NotificationScope	Specifies the context in which the notification appears in the client.
ObjectType	The different types of objects.
PageBackgroundTaskErrorLevel	Specifies how an error in the page background task appears in the client.
ReportFormat	Specifies the format of the report.
SecurityFilter	Specifies how security filters are applied to the record.
TableConnectionType	Use variables of this data type to specify the type of connection to an external database.
TelemetryScope	Represents the emission scope of the telemetry signal.
TestPermissions	Specifies a value that can be used to determine which permission sets are used on tests that are run by test codunits or test functions.
TextEncoding	Represents a file encoding.
TransactionModel	Represents a test transaction model.
TransactionType	Represents a transaction type.
Verbosity	Represents the security level of events.
WebServiceActionResultCode	Represents a web service action status code.

See Also

Getting Started with AL
Developing Extensions

Array Methods

2/17/2021 • 2 minutes to read • Edit Online

An array is a data structure that contains a number of variables which are accessed through computed indices. The variables contained in an array, also called the elements of the array, are all of the same type, and this type is called the element type of the array.

An array has a rank which determines the number of indices associated with each array element. The rank of an array is also referred to as the dimensions of the array. An array with a rank of one is called a single-dimensional array. An array with a rank greater than one is called a multi-dimensional array. Specific sized multi-dimensional arrays are often referred to as two-dimensional arrays, three-dimensional arrays, and so on. Each dimension of an array has an associated length which is an integral number greater than or equal to zero. The maximum number of dimensions is 10 and the total number of elements in all dimensions is 1,000,000.

The length of a dimension determines the valid range of indices for that dimension. For a dimension of length N, indices can range from **1 to N** inclusive. The total number of elements in an array is the product of the lengths of each dimension in the array. If one or more of the dimensions of an array have a length of zero, the array is considered to be empty.

Syntax

The syntax for declaring an array of a specific type is the following:

```
Array [Dimension] of Type;
```

The `Dimension` is a comma-delimited list of integer literals greater than 0, where each integer defines the number of elements in that dimension.

The `Type` is the element type of the array.

Code example

The following code sample shows the declaration of an array with a simple element type.

```
ArrayOfInteger: Array [10] of Integer;
```

The following code sample shows the declaration of an array with an element type of a fixed length.

```
ArrayOfCode: Array [10] of Code[20];
ArrayOfText: Array [10] of Text[20];
```

The following code sample shows the declaration of an array with a complex element type.

```
ArrayOfCodeunits: Array [10] of Codeunit 10;
ArrayOfQueryes: Array [10] of Query "My Query";
ArrayOfTemporaryRecords: Array [10] of Record 10 Temporary;
ArrayOfDotNetVariables: Array [10] of DotNet String;
```

Methods

The following AL methods for arrays are available:

ArrayLen Method
CompressArray Method
CopyArray Method

See Also

AL Method Reference

Essential AL Methods

2/17/2021 • 2 minutes to read • Edit Online

Although there are hundreds of methods in AL, there are several methods that you will use more often than the others. This does not mean that the rest of the methods are obsolete or that you will never use them. However, it does mean that if you are very familiar with this small set of essential methods, you will be able to accomplish many tasks when you are programming in AL. When you want to add more specialized functionality to your applications, you can learn about more of the methods.

The topics in this section describe the most common AL methods. For more details about all of the AL methods, see AL Method Reference.

- Get, Find, and Next Methods
- SetCurrentKey, SetRange, SetFilter, GetRangeMin, and GetRangeMax Methods
- Insert, Modify, ModifyAll, Delete, and DeleteAll Methods
- LockTable Method
- Field Calculation Methods
- Progress Windows, Message, Error, and Confirm Methods
- StrMenu Method

See Also

AL Method Reference

Get, Find, and Next Methods

2/17/2021 • 3 minutes to read • Edit Online

The following methods are used to search for records:

- `Get`
- `Find`
- `Next`

These methods are some of the most frequently used AL methods. When you search for records, you must know the difference between Get and Find and to know how to use Find and Next in conjunction.

> **TIP**
> When using these methods, consider using the partial records methods to improve performance, especially when looping through several records or when table extensions are defined on the table. For more information, see Using Partial Records.

Get method

The Get Method (Record) retrieves one record based on values of the primary key fields.

Get has the following syntax.

```
[Ok :=] Record.Get([Value],...)
```

For example, if the **No.** field is the primary key of the **Customer** table and if you have created a record variable called **CustomerRec** that has a subtype of Customer, then you can use Get in the following way.

```
CustomerRec.Get('4711');
```

The result is that the record of customer 4711 is retrieved.

Get produces a run-time error if it fails and the return value is not checked by the code. In the previous example, the actual code that you write should resemble the following.

```
if CustomerRec.GET('4711') then
.... // Do some processing.
else
.... // Do some error processing.
```

Get searches for a record, regardless of the current filters, and it does not change any filters. Get always searches through all the records in a table.

GetBySystemId method

APPLIES TO: Business Central 2019 release wave 2 and later

The GetBySystemId(Guid) retrieves a record based on the value of its **SystemId** field.

GetBySystemId has the following syntax:

```
RecordExists :=   Record.GetBySystemId(SystemId: Guid)
```

The following example gets the record that has the SystemId `5286305A-08A3-E911-8180-001DD8B7338E`:

```
var
    Customer: Record Customer;
    Text000: Label 'Customer was found.';
begin
    If Customer.GetBySystemId('{5286305A-08A3-E911-8180-001DD8B7338E}') then
    Message(Text000);
end;
```

Similar to the Get method, GetBySystemId searches for a record, regardless of the current filters, and it does not change any filters. Get always searches through all the records in a table.

Find methods

The Find Method (Record) locates a record in a table that is based on the values stored in the keys.

Find has the following syntax.

```
Ok := Record.Find([Which])
```

The *Which* parameter specifies how to perform the search. You can search for values that are greater than, less than, or equal to the key value, or for the first or last record in a table.

The important differences between Get and Find are as follows:

- Find uses the current filters.
- Find can look for records where the key value is equal to, greater than, or smaller than the search string.
- Find can find the first or the last record, depending on the sort order defined by the current key.

When you are developing applications in a relational database, there are often one-to-many relationships defined between tables. An example could be the relationship between an **Item** table, which registers items, and a **Sales Line** table, which registers the detailed lines from sales orders. One record in the **Sales Line** table can only be related to one item, but each item can be related to any number of sales line records. You would not want an item record to be deleted as long as there are still open sales orders that include the item. You can use Find to check for open sales orders.

The OnDelete trigger of the **Item** table includes the following code that illustrates using Find.

```
SalesOrderLine.SetCurrentKey(Type,"No.");
SalesOrderLine.SetRange(Type,SalesOrderLine.Type::Item);
SalesOrderLine.SetRange("No.","No.");
if SalesOrderLine.Find('-') then
    Error(Text001,TableCaption,"No.",SalesOrderLine."Document Type");
```

If you want to find the first record in a table or set, then use the FindFirst Method (Record). If you want to find the last record in a table or set, then use the FindLast Method (Record).

Next method

The Next Method (Record) is often used with FIND to step through the records of a table.

Next has the following syntax.

```
Steps := Record.Next([Steps])
```

In the following example, Find is used to go to the first record of the table. Next is used to step through every record, until there are no more. When there are no more records, Next returns 0 (zero).

```
if (Rec.FindSet) then
repeat
  // process record
until (Rec.Next = 0);
```

See Also

AL Methods
SystemId Field

Creating Handler Methods

2/17/2021 • 3 minutes to read • Edit Online

You can create test codeunits, test methods, and test pages to test your application. We recommend that you create tests that can be automated. To create automated tests, you must write code to handle all UI interactions so that the tests do not require user interaction when they are running. To do this, you create special handler methods.

You can use the following handler methods:

METHOD TYPE	PURPOSE	SIGNATURE	
MessageHandler	Handles **Message** statements.	< *Function name*> (< *Message*> : Text[1024])	
ConfirmHandler	Handles **Confirm** statements.	< *Function name*> (< *Question*> : Text[1024]; var < *Reply*> : Boolean)	
StrMenuHandler	Handles **StrMenu** statements.	< *Function name*> (< *Options* : Text[1024]; var < *Choice*> : Integer; < *Instruction*> : Text[1024])	
PageHandler	Handles specific pages that are not run modally.	< *Function name*> (var < *Page*> : Page < *page id*>) < *Function name*> (var < *Page*> : TestPage < *testpage id*>)	
ModalPageHandler	Handles specific pages that are run modally.	< *Function name*> (var < *Page*> : Page < *page id*>; var < *Response*> : Action) < *Function name*> (var < *Page*> : Page < *testpage id*>)	
ReportHandler	Handles specific reports. If you create a **ReportHandler** method, then that method replaces all code for running the report, including the request page, and a **RequestPageHandler** is not called. Only create a **RequestPageHandler** if you are not using a ReportHandler.	< *Function name*> (var < *Report*> : Report < *report id*>)	

METHOD TYPE	PURPOSE	SIGNATURE	
FilterPageHandler	Handles a specific filter page. The FilterPageHandler tests the UI that is generated by a FilterPageBuilder Data Type.	<*Function name*>(var <*Record1*> : RecordRef)[, var <*Record2*> : RecordRef] [, ...]): Boolean	
RequestPageHandler	Handles the request page of a specific report.	<*Function name*>(var <*RequestPage* > : TestRequestPage)	
HyperlinkHandler	Handles specific hyperlinks.	<*Function name*> (<*Hyperlink*> : Text[1024])	
SendNotificationHandler	Handles **Send** statements.	<*Function name*> (< *TheNotification*>: Notification): Boolean	
RecallNotificationHandler	Handles **Recall**	<*Function name*> (< *TheNotification*>: Notification): Boolean	
SessionSettingsHandler	Handles **RequestSessionUpdate** statements.	<*Function name*>(var <*SessionSettings*>: SessionSettings): Boolean	

How to create a handler method

To create a handler method, you set one of the handler attributes on a method. You must use the method signature specified for the handler attribute that you are using, as illustrated in this code example.

```
[MessageHandler]
procedure MessageHandler(Message: Text[1024])
begin
    Assert.IsTrue(StrPos(Message, MSG_HAS_BEEN_CREATED) > 0, Message);
end;
```

The parameters of the methods that are being handled are passed as parameters to the handler methods. For example, when **Message** is called in a test method, the parameter of the **Message** method is passed as the parameter of the **MessageHandler** method. For page and report handlers, the page, report, or request page is passed as the parameter of the **PageHandler**, **ModalPageHandler**, **ReportHandler**, or **RequestPageHandler**.

You can call handler methods from methods that have the Test Attribute and then specify the handler methods that it will use in the HandlerFunctions Attribute. The code inside the test method should simulate that the UI was actually raised and some values entered or some actions were taken. You can specify more than one handler method by separating the handler method names with a comma.

> **NOTE**
> Every handler method that you enter in the HandlerFunctions Attribute of a test method must be called at least one time in the test method. If you run a test method that has a handler method listed that is not called, then the test fails.

The following example shows a test method that uses the HandlerFunctions Attribute to call the

MessageHandler method.

```
[Test]
[HandlerFunctions('MessageHandler')]
procedure ApproveRequestForPurchCreditMemo()
var
    PurchHeader: Record "Purchase Header";
begin
    ApproveRequestForPurchDocument(PurchHeader."Document Type"::"Credit Memo");
end;
```

See Also

Testing the Application
AL Methods

Handling Errors using Try Methods

2/17/2021 • 2 minutes to read • Edit Online

Try methods in AL enable you to handle errors that occur in the application during code execution. For example, with try methods, you can provide more user-friendly error messages to the end user than those thrown by the system.

> **NOTE**
> Try Methods are available from runtime version 2.0.

Behavior and usage

The main purpose of try methods is to catch errors/exceptions that are thrown by Business Central or exceptions that are thrown during .NET Framework interoperability operations. Try methods catch errors similar to a conditional Codeunit.Run method call, except try method calls do not require that write transactions are committed to the database, and changes to the database that are made with a try method are not rolled back.

Database write transactions in try methods

Because changes made to the database by a try method are not rolled back, you should not include database write transactions within a try method. By default, the Business Central Server configuration prevents you from doing this. If a try method contains a database write transaction, a runtime error occurs.

Handling errors with a return value

A method that is designated as a try method has a Boolean return value (**true** or **false**), and has the construction `OK:= MyTrymethod`. A try method cannot have a user-defined return value.

- If a try method call does not use the return value, the try method operates like an ordinary method and errors are exposed as usual.
- If a try method call uses the return value in an `OK:=` statement or a conditional statement such as `if-then`, errors are caught. The try method returns `true` if no error occurs; `false` if an error occurs.

> **NOTE**
> The return value is not accessible within the try method itself.

Getting details about errors

You can use the GetLastErrorText method to obtain errors that are generated by Business Central. To get details of exceptions that are generated by .NET Framework objects, you can use the GetLastErrorObject method to inspect the `Exception.InnerException` property.

Creating a try method

To create a try method, add a method in the AL code of an object such as a codeunit as usual, and then set the TryFunction Attribute property to **true**.

Example 1

The following simple example illustrates how the try method works. First, create a codeunit that has a local method `MyTrymethod`. Add the following code on the `OnRun` trigger and `MyTrymethod` method.

```
trigger OnRun()
begin
    MyTrymethod;
    message('Everything went well');
end;
```

```
local procedure MyTryMethod()
begin
    error('An error occurred during the operation');
end;
```

When you run this codeunit, the execution of the `OnRun` trigger stops. The error message `An error occurred during the operation` is thrown in the UI.

Now, set the TryFunction Attribute of the `MyTrymethod` method. Then, add code to the `OnRun` trigger to handle the return value of the try method:

```
[TryFunction]
local procedure MyTryMethod()
begin
    error('An error occurred during the operation');
end;

trigger OnRun()
begin
    if MyTryMethod then
        message('Everything went well')
    else
        message('Something went wrong')
end;
```

When you run the codeunit, instead of stopping the execution of the `OnRun` trigger when the error occurs, the error is caught and the message `Something went wrong` is returned.

See Also

AL Simple Statements

Progress Windows, Message, Error, and Confirm Methods

2/17/2021 • 4 minutes to read • Edit Online

You can use several specialized methods to display messages and gather input. We recommend that you use pages to ensure that your application has a consistent user interface. However, there are situations where you may want to use the dialog methods instead of pages. The most important uses of the dialog methods are as follows:

- To display a window that indicates the progress of some processing that may take a long time.
- To stop the running program to display an error Message.
- To let the user confirm a choice before the program continues running.

You can also use the StrMenu method to create pages that present options to the user. It is much faster to use this method than to design a page which only presents a limited set of options to the user. For more information about the StrMenu method, see StrMenu Method.

Best practices for user messages

We recommend the following guidelines for writing messages for end users:

- Write messages correctly according to the grammatical rules for your language.
- Do not use backslashes to indicate line breaks in a message. Line formatting is completed automatically. The only exception is in the Open Method (Dialog). You must use backslashes for the message to be aligned correctly.
- Use the FieldCaption Method (Record) and TableCaption Method (Record) whenever possible to return names of fields and tables as strings so that the user can always recognize a term that indicates a field or table name. The only exception to this is in Open Method (Dialog). In this method, you can use the field name directly. Otherwise, it can be difficult to align correctly. If you refer to a field name without using the FieldCaption method, then type the field name without any single or double quotation marks.
- Try to write all messages on only one line. If you want to use more than one line, then start each new line after a period instead of in the middle of a sentence.
- Do not enter the text directly in the AL code. Instead, enter it as a label so that the message can be translated.

Creating a window to indicate progress

If you have an application that performs some processing that can take a long time to complete, then you should consider displaying a window that informs the user of the progress that is being made. It is always a good idea to inform the user that processes are still running.

A **Cancel** button is automatically added to every dialog window and gives the user the opportunity to stop the processing.

In some applications, you may want to create a window in which each field is updated when the program is running. For example, the fields in the window display the count of the number of postings made. In another application, you may want to display information about the record that is currently being processed. For

example, the field in the window displays the number of the account that is currently being processed.

To create this kind of progress window, you use the Dialog data type.

Message method

The Message Method (Dialog) displays a message in a window that remains open until the user chooses the **OK** button.

The Message method has the following syntax.

```
Message(String [, Value1, ...]);
```

The Message method runs asynchronously, which means that the Message is not run until the method from which it was called ends or another method requests user input. The method is useful for notifying the user that some processing has been successfully completed.

For an example of the Message method, see codeunit 83 in the CRONUS International Ltd. demonstration database. The code in the OnRun trigger converts a quote into a sales order and then displays a Message. The Message is generated by the following code.

```
var
    Text001 : Label 'Quote %1 has been changed to order %2';

Message(Text001,"No.",SalesHeader2."No.");
```

> **NOTE**
> Unlike the progress window, the Message method does not require that you first declare a variable of type Dialog. The Message method creates a window of its own.

Error method

The Error Method (Dialog) is very similar to the Message method except that when the user has acknowledged the Message from an Error method, execution ends. The Error method is also similar to the FieldError method. For more information, see CalcFields, CalcSums, FieldError, FieldName, Init, TestField, and Validate Methods.

The Error method has the following syntax.

```
Error(String [, Value1, ...]);
```

Confirm method

The Confirm Method (Dialog) is used just like the Message method to display a Message. However, unlike the Message method, the Confirm method has a required return value.

The Confirm method has the following syntax.

```
Ok := Dialog.Confirm(String [, Default] [, Value1] ,...);
```

The following example shows how to use the Confirm method.

```
if Confirm('Do you want to post the journal lines and print report %1?', False, ReportID) then
    Message('Posting')
else begin
    Message('No Posting');
    exit;
end;
```

The False parameter in the Confirm statement means that No is the default.

See Also

Dialog Data Type

Creating Test Codeunits and Test Methods

2/17/2021 • 2 minutes to read • Edit Online

In Dynamics 365 Business Central, you can create test codeunits and create test methods in the test codeunits.

Test codeunits are codeunits that have the SubType Property set to **Test**. You write tests as Al code in the methods inside of the test codeunits. There are three types of methods that you can add in a test codeunit: test, handler, and normal. Each method type is used for a specific purpose and behaves differently. When a test codeunit runs, it runs the **OnRun** trigger, and then runs each test method in the codeunit.

By default, each test method runs in a separate database transaction, but you can use the TransactionModel Property on test methods and the TestIsolation Property on test runner codeunits to control the transactional behavior.

The results of a test codeunit and of the individual test methods are displayed in a message window, but you can use the OnAfterTestRun Trigger on a test runner codeunit to capture the results. The outcome of a test method is either SUCCESS or FAILURE. If any error is raised by either the code that is being tested or the test code, then the global outcome of the test codeunit is FAILURE and the error is included in the results log file.

The difference between a normal codeunit and a test codeunit is their execution at runtime. When a normal codeunit is run, if one of its methods fails, then the codeunit is terminated. When a test codeunit is run, even if the outcome of one test method is FAILURE, the next test methods are still run.

The methods in a test codeunit can be one of the following types:

TYPE	DESCRIPTION	
Test method	You use test methods that include AL code that tests the business logic in the application, where each method covers a transaction. You declare the Test Attribute on the method.	
Handler method	You use handler methods to automate tests by handling instances when user interaction is required by the code that is being tested by the test method. In these instances, the handler method is run instead of the requested user interface. The handler method should simulate the user interaction for the test case, such as validating messages, making selections, or entering values. You declare a handler type attribute on the method. For more information, see Creating Handler Methods	
Normal method	You use normal methods to structure the test code by using the same design practices and principles as methods in other codeunits of the application. You declare the Normal Attribute on the method.	

See Also

Testing the Application

Method Attributes

2/17/2021 • 2 minutes to read • Edit Online

An attribute is modifier on a method declaration that specifies information that controls the method's use and behavior. Adding an attribute on a method declaration is also known as *decorating* a method. For example, decorating a method with the `Integration` attribute sets the method to be an event publisher. An attribute can have one or more arguments that set properties for the method instance.

In AL, attributes are placed before the method, and have the following syntax:

```
[Attribute_Name(ArgumentName : data_type, ArgumentName : data_type)]
```

For example, the Integration attribute has two arguments, and the syntax is:

```
[Integration(IncludeSender : Boolean, GlobalVarAccess : Boolean)]
```

Attributes

The following method attributes are available:

- CommitBehavior Attribute
- Business Attribute
- ConfirmHandler Attribute
- EventSubscriber Attribute
- FilterPageHandler Attribute
- HandlerFunctions Attribute
- HyperlinkHandler Attribute
- InDataSet Attribute
- IntegrationEvent Attribute
- InternalEvent Attribute
- MessageHandler Attribute
- ModalPageHandler Attribute
- NonDebuggable Attribute
- Normal Attribute
- Obsolete Attribute
- PageHandler Attribute
- RecallNotificationHandler Attribute
- ReportHandler Attribute
- RequestPageHandler Attribute
- Scope Attribute
- SendNotificationHandler Attribute
- SessionSettingsHandler Attribute
- StrMenuHandler Attribute
- Test Attribute
- TestPermissions Attribute

- TryFunction Attribute

See Also

AL Method Reference

Procedure overload

2/17/2021 • 2 minutes to read • Edit Online

Procedure overload enables developers to create multiple procedures with the same name, but with different signatures, on the same application object. Conceptually, overloaded procedures are used to execute the same task on a different set of arguments. When an overloaded procedure is called, a specific implementation of that procedure, appropriate to the context of the call, will be run.

Reasons for using procedure overload

Overloaded procedures give programmers the flexibility to call a procedure with similar semantics for different types of data. At the same time, overloaded procedures remove the need for abusing the Variant data type for the purpose of processing different types of data in a similar manner and allows the developer to write strongly-typed code and rely on the compiler for validation.

Remarks

Overload resolution is performed by using procedure signatures to find the best match. The signature of a procedure is represented by its name and the type, order, and number of parameters. The return type of a procedure is not part of the procedure's signature.

Example

The following example shows how a **ToString** method can be implemented with and without using procedure overloads.
In the first code snippet, a **ToString** procedure is implemented. This takes a Variant value and inspects the type of the value to delegate to different implementations. If the caller passes a value of a different type than Integer, Date, and Text, an empty string will be returned. This can lead to bugs that will only show up at runtime.

```
codeunit 10 Stringifier
{
    local procedure TextToString(value : Text) : Text;
    begin
        Exit(value);
    end;

    local procedure DateToString(value : Date) : Text;
    begin
        Exit(Format(value));
    end;

    local procedure IntegerToString(value : Integer) : Text;
    begin
        Exit(Format(value));
    end;

    procedure ToString(value: Variant) : Text;
    begin
        if value.IsInteger then
            Exit(IntegerToString(value))
        else if value.IsDate then
                Exit(DateToString(value))
        else if value.IsText then
                Exit(TextToString(value))
        else
            Exit('');
    end;
}
```

In the second code snippet, we overload the ToString procedure for Text, Date and Integer. At this point, it is not possible for a caller to call a ToString method with a different type other than Integer, Date, or Text. This will catch the bug above at compile time.

```
codeunit 10 StringifierWithOverloads
{
    procedure ToString(value : Text) : Text;
    begin
        Exit(value);
    end;

    procedure ToString(value : Date) : Text;
    begin
        Exit(Format(value));
    end;

    procedure ToString(value : Integer) : Text;
    begin
        Exit(Format(value));
    end;
}
```

See Also

AL Method Reference
AL Development Environment

Joker Data Type

2/17/2021 • 2 minutes to read • Edit Online

`Joker` is an *internal* data type that is not exposed to AL developers. Joker can replace any other type and represents a wildcard. In certain cases, `Joker` plays the role of a generic T which is inferred from the type of the left-hand side of the invocation expression, in other cases, it is inferred from another parameter.

The following illustrates examples of how `Joker` is used in AL.

```
procedure SetRange(Field: Joker, [FromValue: Joker], [ToValue: Joker])
```

```
procedure SetFilter(Field: Joker, String: Text, [Value: Joker, ...])
```

```
procedure SetAscending(Field: Joker, Ascending: Boolean)
```

See Also

AL Method Reference
AL Development Environment

Action Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the action that the user took on the page.

Members

MEMBER	DESCRIPTION
None	Represents the result of running a page.
OK	Represents the result of the user closing a page window by performing one of the following actions: - Chooses the **OK** button. - Chooses the **X** button when there was no **Cancel** button on the window. - Presses the Esc key when there is no **Cancel** button on the window.
Cancel	Represents the result of the user closing a page window by performing one of the following actions: - Chooses the **Cancel** button. - Chooses the **X** button when there is a **Cancel** button on the window. - Presses the Esc key when there is a **Cancel** button on the window
LookupOK	Represents the result of the user closing a lookup window by performing one of the following actions: - Chooses the **OK** button. - Chooses an item in the Lookup window.
LookupCancel	Represents the result of the user closing a lookup window by choosing the **Cancel** button.
Yes	Represents the result of the user closing a confirmation window by choosing the **Yes** button.
No	Represents the result of the user closing a confirmation window by performing one of the following actions: - Chooses the No button. - Chooses the X button. - Presses the Esc key.
RunObject	Represents the result of the user selecting an option that ran another object.

MEMBER	DESCRIPTION
RunSystem	Represents the result of the user selecting an option that ran an external program.

See Also

Getting Started with AL
Developing Extensions

ClientType Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the type of the client executing the operation.

Members

MEMBER	DESCRIPTION
Background	A background session.
ChildSession	A child session.
Desktop	A desktop client.
Management	A management client.
NAS	A NAS client.
OData	A NAS client.
Phone	Microsoft Dynamics Business Central Phone client.
SOAP	A SOAP client.
Tablet	Microsoft Dynamics Business Central Tablet client.
Web	Microsoft Dynamics Business Central Web client.
Windows	Microsoft Dynamics Business Central Windows client.
Current	Microsoft Dynamics Business Central Windows client.
Default	The default client.
ODataV4	A ODataV4 client.
Api	An API client.
Teams	Microsoft Teams client.

See Also

Getting Started with AL
Developing Extensions

CommitBehavior Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Specifies whether commit is allowed within the scope of the method.

Members

MEMBER	DESCRIPTION
Ignore	Ignore commits within the scope of this method.
Error	Throw an error when a commit is attempted within the scope of this method.

See Also

Getting Started with AL
Developing Extensions

DataClassification Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the classification of the data in the table or field.

Members

MEMBER	DESCRIPTION
CustomerContent	Content directly provided/created by admins and users.
ToBeClassified	Content that has not yet been given a classification. This is the initial value when table or field is created.
EndUserIdentifiableInformation	(EUII) Data that identifies or could be used to identify the user of a Microsoft service. EUII does not contain Customer content.
AccountData	Customer billing information and payment instrument information, including administrator contact information, such as tenant administrator's name, address, or phone number.
EndUserPseudonymousIdentifiers	(EUPI) An identifier created by Microsoft tied to the user of a Microsoft service. When EUPI is combined with other information, such as a mapping table, it identifies the end user. EUPI does not contain information uploaded or created by the customer (Customer content or EUII).
OrganizationIdentifiableInformation	(OII) Data that can be used to identify a tenant, generally config or usage data. This data is not linkable to a user and does not contain Customer content.
SystemMetadata	Data generated while running the service or program that is not linkable to a user or tenant.

See Also

Getting Started with AL
Developing Extensions

DataScope Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Identifies the scope of stored data in the isolated storage.

Members

MEMBER	DESCRIPTION
Module	Indicates that the record is available in the scope of the app(extension) context.
Company	Indicates that the record is available in the scope of the company within the app context.
User	Indicates that the record is available for a user within the app context.
CompanyAndUser	Indicates that the record is available for a user and specific company within the app context.

See Also

Getting Started with AL
Developing Extensions
Isolated Storage

DefaultLayout Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

The default layout to be used by a report.

Members

MEMBER	DESCRIPTION
None	The default layout is not set.
RDLC	The default layout is RDLC.
Word	The default layout is Word.

See Also

Getting Started with AL
Developing Extensions

ErrorType Option Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 3.0.*

Represents the type of error.

Members

MEMBER	DESCRIPTION
Client	Identifies a client error. The specified message will be shown in the client to the user and sent to telemetry.
Internal	Identifies an internal, the message specified will be sent to telemetry and a generic error will be displayed to the user.

See Also

Getting Started with AL
Developing Extensions

ExecutionContext Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the context in which a session is running. In certain scenarios, for example during upgrade, the system will run a session in a special context for a limited time.

Members

MEMBER	DESCRIPTION
Normal	The normal execution context.
Install	An application is being installed.
Uninstall	An application is being uninstalled.
Upgrade	An application is being upgraded.

See Also

Getting Started with AL
Developing Extensions

ExecutionMode Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

The execution mode of the current session.

Members

MEMBER	DESCRIPTION
Standard	The session is executing in standard mode.
Debug	The session is executing in debug mode.

See Also

Getting Started with AL
Developing Extensions

FieldClass Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the type of a field class.

Members

MEMBER	DESCRIPTION
Normal	A normal field.
FlowField	A flow field.
FlowFilter	A flow filter.

See Also

Getting Started with AL
Developing Extensions

FieldType Option Type

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the type of a table field.

Members

MEMBER	DESCRIPTION
Boolean	Assumes the values true or false. When formatted, a Boolean field is shown as "Yes" or "No". The size of the corresponding SQL data type, TINYINT, is 1 byte.
Integer	Denotes an integer between -2,147,483,647 and 2,147,483,647. The size of the corresponding SQL data type, INTEGER, is 4 bytes.
BigInteger	A 64-bit integer.
Decimal	A decimal number between -10^63 and 10^63. The exponent ranges from -63 to 63. Decimal numbers are held in memory with 18 significant digits. The representation of a decimal number is a Binary Coded Decimal (BCD). The size of the corresponding SQL data type, DECIMAL(38,20), is 17 bytes. We recommend that you construct decimals that operate on numbers in the range of +/- 999,999,999,999,999.99. You can construct larger numbers in some cases, but overflow, truncation or loss of precision can occur.
Option	An option field is defined by using an option string, which is a comma-separated list of strings that represent each valid value of the field. This string is used when a field of type Option is formatted and its value is converted into a string.
Text	Any alphanumeric string. The field must be defined to be between 1 and 250 characters. The number of bytes used by a text field equals (number of characters + 1) * 2. The additional character is used for the string terminating character, which is '0' in Unicode. The size of a Unicode character is 2 bytes. Therefore, you multiply the number of characters by two to get the size.
Code	An alphanumeric string, which is right-justified if the contents are numbers only. If letters or blanks occur among the numbers, the contents are left-aligned. All letters are converted to uppercase upon entry.

MEMBER	DESCRIPTION
DateTime	Represents a point in time as a combined date and time. The datetime is stored in the database as Coordinated Universal Time (UTC) and is always displayed as local time in Dynamics 365 Business Central.
Time	Any time in the range 00:00:00 to 23:59:59.999. A time field contains 1 plus the number of milliseconds since 00:00:00 o'clock, or 0 (zero), an undefined time.
Date	A date value in the range from January 1, 1753 to December 31, 9999. An undefined date is expressed as 0. All dates have a corresponding closing date. The system considers the closing date for a given date as a period that follows the given date but comes before the next normal date; that is, a closing date is sorted immediately after the corresponding normal date but before the next normal date.
DateFormula	Used to verify the date entered by the user.
Duration	Represents the difference between two points in time, in milliseconds. This value can be negative.
Guid	Globally unique identifier (GUID).
RecordId	Unique record identifier.
TableFilter	This data type is used to apply a filter to another table. Currently, this can only be used to apply security filters from the Permission table.
Blob	Binary Large Object. Used to store bitmaps and memos. Notice that the BLOB is not stored in the record, but in the BLOB area of the table.
Media	A complex type that encapsulates media files, such as image .jpg and .png files, in application database tables. The Media data type can be used as a table field data type, but cannot be used as a variable or parameter.
MediaSet	A complex type that encapsulates media, such as images, in application database tables. The MediaSet data type can be used as a table field data type, but cannot be used as variable or parameter.

See Also

Getting Started with AL
Developing Extensions

NotificationScope Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the context in which the notification appears in the client.

Members

MEMBER	DESCRIPTION
GlobalScope	The notifications are not directly related to the user's current task. **Note:** GlobalScope is currently not supported, so do not use this value.
LocalScope	The notification appears in context of the user's current task, on the page the user is currently working on. This is the default value.

See Also

Getting Started with AL
Developing Extensions

ObjectType Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

The different types of objects.

Members

MEMBER	DESCRIPTION
Codeunit	The Codeunit object type
MenuSuite	The Menusuite object type
Page	The Page object type
Query	The Query object type
Report	The Report object type
Table	The Table object type
XmlPort	The XMLPort object type

See Also

Getting Started with AL
Developing Extensions

PageBackgroundTaskErrorLevel Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Specifies how an error in the page background task appears in the client.

Members

MEMBER	DESCRIPTION
Error	Error occuring in a page background task is displayed as an normal error in the client. This is the default value.
Warning	Error occuring in a page background task is displayed as an warning in the client. **Note:** Any error thrown in completion trigger will ignore this value and will be displayed in the client as a normal error.
Ignore	Error occuring in a page background task is not displayed in the client. **Note:** Any error thrown in completion trigger will ignore this value and will be displayed in the client as a normal error.

See Also

Getting Started with AL
Developing Extensions

ReportFormat Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the format of the report.

Members

MEMBER	DESCRIPTION
Excel	Saves the report as an Excel file.
Html	Saves the report in HTML format.
Pdf	Saves the report in PDF format.
Word	Saves the report in Word format.
Xml	Saves the report in XML format.

See Also

Getting Started with AL
Developing Extensions

SecurityFilter Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies how security filters are applied to the record.

Members

MEMBER	DESCRIPTION
Validated	All security filters are applied to this instance of the record and if any code tries to access a record that is outside the range of the security filters, then an error occurs.
Filtered	All security filters are applied to this instance of the record.
Ignored	All security filters are ignored for this instance of the record.
Disallowed	Security filters are not allowed on the record. If any security filters are set, then you receive an error when you run the object that uses this instance of the record.

See Also

Getting Started with AL
Developing Extensions

TableConnectionType Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Use variables of this data type to specify the type of connection to an external database.

Members

MEMBER	DESCRIPTION
CRM	Specifies the table as an integration table for integrating Dynamics 365 Business Central with Dynamics 365 for Sales. The table is typically based on an entity in Dynamics 365 for Sales, such as the Accounts entity.
ExternalSQL	Specifies the table as a table or view in SQL Server that is not in the Dynamics 365 Business Central database.
Exchange	This is for internal use only.
MicrosoftGraph	This is for internal use only.

See Also

Getting Started with AL
Developing Extensions

TelemetryScope Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.4.*

Represents the emission scope of the telemetry signal.

Members

MEMBER	DESCRIPTION
ExtensionPublisher	Emit telemetry to extensions publisher's account.
All	Emit telemetry to extension publisher's and partner's telemetry account .

Examples

```
if not FileManagement.ServerFileExists(ServerFile) then begin
            LogInternalError(SomethingWentWrongErr, DataClassification::SystemMetadata, Verbosity::Error);
```

```
if not XmlDocument.ReadFrom('<?xml version="1.0" encoding="UTF-8"?>' + '<Elster xmlns="' + XmlNameSpace +
'"></Elster>', XmlSubDoc) then
            LogInternalError(XMLDocHasNotBeenCreatedErr, DataClassification::SystemMetadata,
Verbosity::Error);
```

See Also

Getting Started with AL
Developing Extensions

TestPermissions Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies a value that can be used to determine which permission sets are used on tests that are run by test codunits or test functions.

Members

MEMBER	DESCRIPTION
InheritFromTestCodeunit	Is only relevant for test methods; not test codeunits. It specifies that a test method uses the TestPermissions property setting of the test codeunit to which it belongs. If you use this value on a test codunit, the property will resolve to Restrictive at runtime.
Restrictive	Does not perform any operations or have any specific behavior. Instead, you programmatically define what each value does, and the permissions sets it applies at runtime, by adding code in a test runner codeunit.
NonRestrictive	Does not perform any operations or have any specific behavior. Instead, you programmatically define what each value does, and the permissions sets it applies at runtime, by adding code in a test runner codeunit.
Disabled	Does not perform any operations or have any specific behavior. Instead, you programmatically define what each value does, and the permissions sets it applies at runtime, by adding code in a test runner codeunit.

See Also

Getting Started with AL
Developing Extensions

TextEncoding Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a file encoding.

Members

MEMBER	DESCRIPTION
MSDos	MSDos encoding.
UTF8	UTF8 encoding.
UTF16	UTF16 encoding.
Windows	Windows encoding.

See Also

Getting Started with AL
Developing Extensions

TransactionModel Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a test transaction model.

Members

MEMBER	DESCRIPTION
AutoCommit	The transaction automatically commits after the Test method has run.
AutoRollback	The transaction is automatically rolled back after the Test method has run.
None	No write-transaction is open in the test-method code, and writes will fail. The transaction model mirrors the model used by the "real" client. Every call from the TestPage to the "server" has its own transaction.

See Also

Getting Started with AL
Developing Extensions

TransactionType Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a transaction type.

Members

MEMBER	DESCRIPTION
UpdateNoLocks	This is an update transaction. Modifications can occur within the transaction. All read operations are performed with READ UNCOMMITTED locking until the table is either modified by a write operation or locked with the LOCKTable Method (Record). From this point until the end of the transaction, all read operations are performed with UPDLOCK locking. This transaction type improves concurrency for all tables that users access within the transaction by delaying locking as much as it can. However, the disadvantage is that you must know when to lock the tables for the required transaction behavior.
Update	This is an update transaction. Modifications can occur within the transaction. All read operations are performed with REPEATable READ locking until the table is either modified by any write operation or locked with the LOCKTable method. From this point forward, all read operations are performed with UPDLOCK locking. This transaction type provides full transaction isolation from the start of the transaction, regardless of the lock status of tables that users access within the transaction.
Snapshot	This is a read-only transaction. Modifications cannot occur within the transaction. All read operations are performed with REPEATable READ locking. Therefore, shared locks are added on all data and are maintained until the end of the transaction. This prevents other transactions from modifying any rows that have been read by the current transaction.
Browse	This is a read-only transaction. Modifications cannot occur within the transaction. All read operations are performed with READ UNCOMMITTED locking. Therefore, no locks are added and locks that are added by other sessions are not honored. This means that the transaction may read uncommitted data.
Report	Report option maps to one of the basic options. This enables a report to use the most concurrent read-only form of data access for the connected server. When you use Dynamics 365 Business Central database server, it maps to Snapshot and when you use SQL Server, it maps to Browse.

See Also

Verbosity Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the security level of events.

Members

MEMBER	DESCRIPTION
Critical	Identifies an abnormal exit or termination event.
Error	Identifies a severe error event.
Warning	Identifies a warning event such as an allocation failure.
Normal	Identifies a non-error event such as an entry or exit event.
Verbose	Identifies a detailed trace event.

See Also

Getting Started with AL
Developing Extensions

WebServiceActionResultCode Option Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Represents a web service action status code.

Members

MEMBER	DESCRIPTION
None	No status code.
Get	Item read.
Created	Item created.
Updated	Item updated.
Deleted	Item deleted.

See Also

Getting Started with AL
Developing Extensions

BigInteger Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stores very large whole numbers that range from -9,223,372,036,854,775,807 to 9,223,372,036,854,775,807.

Remarks

This data type is a 64-bit integer.

You must add an L to the constant definition to inform AL that the integer must be interpreted and treated as a BigInteger.

If you assign -9,223,372,036,854,775,808 directly to a BigInteger variable, then you get an error when you try to compile the code. However, you can indirectly assign -9,223,372,036,854,775,808 to a BigInteger variable by using the following code.

```
BigIntegerVar := -9223372036854775807L;
BigIntegerVar := BigIntegerVar - 1;
```

If you try to indirectly assign a value that is smaller than -9,223,372,036,854,775,808, or larger than 9,223,372,036,854,775,807, then you get a run-time error.

Example

```
BI := 1L;
BI := 455500000000L;
```

See Also

Getting Started with AL
Developing Extensions

BigText Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Handles large text documents.

The following methods are available on instances of the BigText data type.

METHOD NAME	DESCRIPTION
AddText(String [, Integer])	Adds a text string to a BigText variable.
AddText(BigText [, Integer])	Adds a text string to a BigText variable.
GetSubText(var Text, Integer [, Integer])	Gets part of a BigText variable.
GetSubText(var BigText, Integer [, Integer])	Gets part of a BigText variable.
Length()	Retrieves the length of the text stored in this BigText instance.
Read(InStream)	Streams a BigText object that is stored as a BLOB in a table to a BigText variable.
TextPos(String)	Gets the position at which a specific string first occurs in this BigText instance.
Write(OutStream)	Streams a BigText object to a BLOB field in a table.

Remarks

This data type cannot be shown in a message window or be seen in the Debugger. The maximum length of a BigText variable is 2,147,483,647 characters and this corresponds to 2 GB. You can use the BigText methods to manipulate a BigText variable, for example to extract part of a BigText variable or to add a text string to a BigText variable. The normal string methods cannot be used with a BigText variable.

See Also

Getting Started with AL
Developing Extensions

BigText.AddText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a text string to a BigText variable.

Syntax

```
BigText.AddText(String: String [, Position: Integer])
```

Parameters

BigText Type: BigText An instance of the BigText data type.

String
Type: String
The string that will be added to the BigText variable. If this parameter is empty, then the BigText variable is not modified.

Position
Type: Integer
This is an optional parameter that defines the position in the BigText variable where the string is inserted. If this parameter is omitted, then the string is added at the end of the BigText variable. If this parameter is less than one, then a run-time error occurs. If this parameter is greater than the length of the BigText variable, then the string is added at the end of the BigText variable.

Remarks

Variable can be inserted anywhere in *BigText* or added at the end of the *BigText*.

The first character in a *BigText* variable is position 1.

To delete the content in a *BigText* variable, use the Clear Method. The following code shows the syntax for the method: `Clear(BigText)`

> **NOTE**
> If you use AddText to add multiple BigText strings to what is presented as a single string, you can experience performance problems. The same applies to other repetitive uses of AddText. This is due to the implementation of the BigText data type, which relies on a String object that is immutable. You can avoid this issue by refactoring the code to reduce the number of additions or deletions. Alternatively, you can change your implementation to use the System.Text.StringBuilder class instead. For more information, see Immutability and the StringBuilder Class in the MSDN Library.

Example 1

The following examples show how to use the AddText method. The specified text is inserted into the BigText string at the specified position. In these examples, the initial content of the *BigText* variable is `ABCDEFG`. These examples require that you create the following variable.

```
var
    MyBigText: BigText;
```

The following example inserts the string 'ZZZ' after the character B in the MyBigText variable because 3 is specified for *Position*.

```
// Example 1
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('ZZZ', 3); // Returns the subtext ABZZZCDEFG.
```

Example 2

The following example appends the string 'ZZZ' at the end of the MyBigText variable because the number specified for *Position* is greater than the length of the MyBigText variable.

```
// Example 2
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('ZZZ', 15); // Returns the subtext ABCDEFGZZZ.
```

Example 3

In the following example, the content of the MyBigText variable is unchanged because the specified variable is an empty string.

```
// Example 3
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('', 1); // Returns the subtext ABCDEFG.
```

Example 4

In the following example, the method returns an error because 0 is specified for *Position*.

```
// Example 4
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('ZZZ', 0); // Returns an error.
```

See Also

BigText Data Type
Getting Started with AL
Developing Extensions

BigText.AddText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a text string to a BigText variable.

Syntax

```
BigText.AddText(String: BigText [, Position: Integer])
```

Parameters

BigText Type: BigText An instance of the BigText data type.

String
Type: BigText
The string that will be added to the BigText variable. If this parameter is empty, then the BigText variable is not modified.

Position
Type: Integer
This is an optional parameter that defines the position in the BigText variable where the string is inserted. If this parameter is omitted, then the string is added at the end of the BigText variable. If this parameter is less than one, then a run-time error occurs. If this parameter is greater than the length of the BigText variable, then the string is added at the end of the BigText variable.

Remarks

Variable can be inserted anywhere in *BigText* or added at the end of the *BigText*.

The first character in a *BigText* variable is position 1.

To delete the content in a *BigText* variable, use the Clear Method. The following code shows the syntax for the method: `Clear(BigText)`

> **NOTE**
> If you use AddText to add multiple BigText strings to what is presented as a single string, you can experience performance problems. The same applies to other repetitive uses of AddText. This is due to the implementation of the BigText data type, which relies on a String object that is immutable. You can avoid this issue by refactoring the code to reduce the number of additions or deletions. Alternatively, you can change your implementation to use the System.Text.StringBuilder class instead. For more information, see Immutability and the StringBuilder Class in the MSDN Library.

Example 1

The following examples show how to use the AddText method. The specified text is inserted into the BigText string at the specified position. In these examples, the initial content of the *BigText* variable is `ABCDEFG`. These examples require that you create the following variable.

```
var
    MyBigText: BigText;
```

The following example inserts the string 'ZZZ' after the character B in the MyBigText variable because 3 is specified for *Position*.

```
// Example 1
MyBigText.AddTEXT('ABCDEFG');
MyBigText.AddTEXT('ZZZ', 3); // Returns the subtext ABZZZCDEFG.
```

Example 2

The following example appends the string 'ZZZ' at the end of the MyBigText variable because the number specified for *Position* is greater than the length of the MyBigText variable.

```
// Example 2
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('ZZZ', 15); // Returns the subtext ABCDEFGZZZ.
```

Example 3

In the following example, the content of the MyBigText variable is unchanged because the specified variable is an empty string.

```
// Example 3
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('', 1); // Returns the subtext ABCDEFG.
```

Example 4

In the following example, the method returns an error because 0 is specified for *Position*.

```
// Example 4
MyBigText.AddText('ABCDEFG');
MyBigText.AddText('ZZZ', 0); // Returns an error.
```

See Also

BigText Data Type
Getting Started with AL
Developing Extensions

BigText.GetSubText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets part of a BigText variable.

Syntax

```
[Length := ]  BigText.GetSubText(var Variable: Text, Position: Integer [, Length: Integer])
```

Parameters

BigText Type: BigText An instance of the BigText data type.

Variable
Type: Text
The sub text of the BigText that is retrieved. This is the actual text that is returned.

Position
Type: Integer
The position in the BigText variable that the sub text is to be retrieved from. If this parameter is less than one, then a run-time error occurs. If this parameter is greater than the length of the BigText variable, then an empty string is returned. If the value of this parameter plus the value of the Length parameter is greater than the length of the BigText variable, then the remainder of the BigText variable from the position specified by this parameter is returned.

Length
Type: Integer
The length of the sub text that should be retrieved. This parameter is optional. If this parameter is omitted the function retrieves a sub text that starts at Position and runs to the end of the BigText variable. If this parameter is less than 0, then a run-time error occurs. If the value of the Position parameter plus the value of this parameter is greater than the length of the BigText variable, then the remainder of the BigText variable from the position specified by this parameter is returned.

Return Value

Length Type: Integer The length of the result text.

Remarks

The first character in a BigText variable is position 1.

To delete the content in a BigText variable use the Clear Method. The following code snippet shows the syntax for the clear method. `Clear(BigText)`.

Example 1

The following examples demonstrate how to use the GetSubText method. This example requires that you create

the following global variables and text constant.

```
var
    MyBigText: BigText;
    VarSubText: Text;
    Text000: Label 'VarSubText = %1';
```

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method starts from the third position (the character C) in the MyBigText variable and retrieves two characters. The result is the subtext CD. This is because the number 2 is specified for *Length* .

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 3, 2); // Returns CD.
Message(Text000, VarSubText);
```

Example 2

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method returns an error because zero is specified for *Position*.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 0, 4); // Returns an error.
Message(Text000, VarSubText);
```

Example 3

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method returns an error because a negative number is specified for *Length*.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 5, -2); // Returns an error.
Message(Text000, VarSubText);
```

Example 4

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method returns an empty string because the number specified for *position* is greater than the length of the MyBigText variable.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 15, 4); // Returns an empty string.
Message(Text000, VarSubText);
```

Example 5

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method starts from the forth position (the character D) and retrieves all the characters in the MyBigText string. The result is the subtext DEFG. This is because the number specified for *Length* is greater than the length of the MyBigText variable.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 4, 15); // Returns DEFG.
Message(Text000, VarSubText);
```

See Also

BigText Data Type
Getting Started with AL
Developing Extensions

BigText.GetSubText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets part of a BigText variable.

Syntax

```
[Length := ]  BigText.GetSubText(var Variable: BigText, Position: Integer [, Length: Integer])
```

Parameters

BigText Type: BigText An instance of the BigText data type.

Variable

Type: BigText

The sub text of the BigText that is retrieved. This is the actual text that is returned.

Position

Type: Integer

The position in the BigText variable that the sub text is to be retrieved from. If this parameter is less than one, then a run-time error occurs. If this parameter is greater than the length of the BigText variable, then an empty string is returned. If the value of this parameter plus the value of the Length parameter is greater than the length of the BigText variable, then the remainder of the BigText variable from the position specified by this parameter is returned.

Length

Type: Integer

The length of the sub text that should be retrieved. This parameter is optional. If this parameter is omitted the method retrieves a sub text that starts at Position and runs to the end of the BigText variable. If this parameter is less than 0, then a run-time error occurs. If the value of the Position parameter plus the value of this parameter is greater than the length of the BigText variable, then the remainder of the BigText variable from the position specified by this parameter is returned.

Return Value

Length Type: Integer The length of the result text.

Remarks

The first character in a BigText variable is position 1.

To delete the content in a BigText variable use the Clear Method. The following code snippet shows the syntax for the clear method. `Clear(BigText)`.

Example 1

The following examples demonstrate how to use the GetSubText method. This example requires that you create

the following global variables and text constant.

```
var
    MyBigText: BigText;
    VarSubText: Text;
    Text000: Label 'VarSubText = %1';
```

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method starts from the third position (the character C) in the MyBigText variable and retrieves two characters. The result is the subtext CD. This is because the number 2 is specified for *Length* .

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 3, 2); // Returns CD.
Message(Text000, VarSubText);
```

Example 2

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method returns an error because zero is specified for *Position*.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 0, 4); // Returns an error.
Message(Text000, VarSubText);
```

Example 3

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method returns an error because a negative number is specified for *Length*.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 5, -2); // Returns an error.
Message(Text000, VarSubText);
```

Example 4

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method returns an empty string because the number specified for *position* is greater than the length of the MyBigText variable.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 15, 4); // Returns an empty string.
Message(Text000, VarSubText);
```

Example 5

The following example initializes the content of the MyBigText variable with the text `ABCDEFG`. The method starts from the forth position (the character D) and retrieves all the characters in the MyBigText string. The result is the subtext DEFG. This is because the number specified for *Length* is greater than the length of the MyBigText variable.

```
MyBigText.AddText('ABCDEFG');
MyBigText.GetSubText(VarSubText, 4, 15); // Returns DEFG.
Message(Text000, VarSubText);
```

See Also

BigText Data Type
Getting Started with AL
Developing Extensions

BigText.Length Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the length of the text stored in this BigText instance.

Syntax

```
Length :=   BigText.Length()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

BigText Type: BigText An instance of the BigText data type.

Return Value

Length Type: Integer The length of the text stored in this BigText instance.

Remarks

To delete the content in a BigText variable use the Clear Method. The syntax for the Clear method is shown in the following code snippet: `Clear(BigText)`.

Example

The following example demonstrates how to retrieve the length of a BigText variable.

In this example, the BigText variable is initialized with the text 'ABCDEFG'. The length, which is 7, is stored in the VarLength variable and displayed in a message box.

```
var
    MyBigText: BigText;
    VarLength: Text;
    Text000: Label 'VarLength = %1';
begin
    MyBigText.AddText('ABCDEFG');
    VarLength := MyBigText.Length;
    Message(Text000, VarLength);
end;
```

See Also

BigText Data Type
Getting Started with AL

BigText.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Streams a BigText object that is stored as a BLOB in a table to a BigText variable.

Syntax

```
[Ok := ]  BigText.Read(InStream: InStream)
```

Parameters

BigText Type: BigText An instance of the BigText data type.

InStream
Type: InStream
The InStream object type that you use to stream a BLOB to a BigText variable.

Return Value

Ok Type: Boolean **true** if the read transaction was successful, otherwise **false**.

Remarks

To delete the content in a BigText variable, use the Clear Method.

```
Clear(BigText)
```

Example

This example shows how to stream a BigText that is stored as a BLOB in a table to a BigText variable.

```
var
    Bstr: BigText;
    Istream: InStream;
    EmployeeRec: Record Employee;
begin
    EmployeeRec.Find('-');
    EmployeeRec.CalcFields(Picture);
    EmployeeRec.Picture.CreateInStream(Istream);
    Bstr.Read(Istream);
end;
```

Use the CalcFields Method (Record) to calculate the BlobField. A BlobField is a binary large object (maximum size 2 GB) and must be calculated if you want to use it in AL or display it in the application.

See Also

BigText.TextPos Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the position at which a specific string first occurs in this BigText instance.

Syntax

```
Position :=   BigText.TextPos(String: String)
```

Parameters

BigText Type: BigText An instance of the BigText data type.

String
Type: String
The text string to search for in the BigText variable. If this parameter is empty, then 0 is returned.

Return Value

Position Type: Integer The position at which a specific string first occurs in this BigText instance.

Remarks

The first character in a *BigText* variable is position 1.

Example 1

The following examples show how to use the TextPos Method. These examples require that you create the following global variables and text constant.

```
var
    MyBigText: BigText;
    VarPosition: Text;
    Text000: Label 'VarPosition = %1';
```

The following examples first initialize the content of the *BigText* variable with the text `ABCDEFG`.

In this example, the first occurrence of the character B (the first character of the specified text) is found in the second position in the MyBigText variable so the method returns 2. The return value is stored in the variable VarPosition and displayed in a message box.

```
MyBigText.AddText('ABCDEFG');
VarPosition := MyBigText.TextPos('BCD'); // Returns 2.
Message(Text000, VarPosition);
```

Example 2

In the following example, the method returns 0 because the specified string is not found in the MyBigText variable. The return value is stored in the variable VarPosition and displayed in a message box.

```
MyBigText.AddText('ABCDEFG');
VarPosition := MyBigText.TextPos(''); // Returns 0.
Message(Text000, VarPosition);
```

Example 3

In the following example, the method returns 0 because the specified string is not found in the MyBigText variable. The return value is stored in the variable VarPosition and displayed in a message box.

```
MyBigText.AddText('ABCDEFG');
VarPosition := MyBigText.TextPos('XYZ'); // Returns 0.
Message(Text000, VarPosition);
```

See Also

BigText Data Type
Getting Started with AL
Developing Extensions

BigText.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Streams a BigText object to a BLOB field in a table.

Syntax

```
[Ok := ]  BigText.Write(OutStream: OutStream)
```

Parameters

BigText Type: BigText An instance of the BigText data type.

OutStream
Type: OutStream
The stream to which you write a BigText.

Return Value

Ok Type: Boolean **true** if the write transaction was successful, otherwise **false**.

Remarks

To delete the content in a BigText variable, use the Clear Method.

```
Clear(BigText)
```

Example

This example shows how to stream a BigText to a BLOB field in a table.

```
var
    Bstr: BigText;
    Ostream: OutStream;
    ItemRec: Record Item;
begin
    Bstr.AddText('This is the text string that we want to store in a BLOB field.');
    ItemRec.Picture.CreateOutStream(Ostream);
    Bstr.Write(Ostream);
    ItemRec.Insert;
end;
```

See Also

BigText Data Type
Getting Started with AL
Developing Extensions

Blob Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a complex data type. Variables of this data type differ from normal numeric and string variables in that BLOBs have a variable length. The maximum size of a BLOB(binary large object) is 2 GB.

The following methods are available on instances of the Blob data type.

METHOD NAME	DESCRIPTION
CreateInStream(InStream [, TextEncoding])	Creates an InStream object for a binary large object (BLOB). This enables you to read data from the BLOB.
CreateOutStream(OutStream [, TextEncoding])	Creates an OutStream object for a binary large object (BLOB). This enables you to write data to the BLOB.
Export(String)	Exports a binary large object (BLOB) to a file.
HasValue()	Determines whether a binary large object (BLOB) has a value.
Import(String)	Imports a binary large object (BLOB) from a file.
Length()	Returns the number of bytes in the binary large object (BLOB).

Remarks

Use BLOBs to store memos (text), pictures (bitmaps), or user-defined types.

> **NOTE**
> You cannot view text that is stored in BLOBs from the development environment.

You can read from and write to BLOBs by creating input and output streams, respectively. To do so, use CreateInStream method (BLOB) and CreateOutStream method (BLOB).

See Also

Getting Started with AL
Developing Extensions

Blob.CreateInStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an InStream object for a binary large object (BLOB). This enables you to read data from the BLOB.

Syntax

```
Blob.CreateInStream(InStream: InStream [, Encoding: TextEncoding])
```

Parameters

Blob Type: Blob An instance of the Blob data type.

InStream
Type: InStream
The InStream object type that has been created.

Encoding
Type: TextEncoding
The encoding that will be used by the stream.

Optionally, you can specify the encoding on the stream. By specifying the File Handling and Text Encoding, you ensure that all the language-specific characters are represented correctly in Dynamics 365 when you read data and write data. The following example illustrates how you can set the encoding to Windows when you create a stream for a BLOB field.

```
MyTable.MyBLOBfield.CreateInStream(MyStream, TextEncoding::WINDOWS);
```

For more information, see File Handling and Text Encoding.

See Also

Blob Data Type
Getting Started with AL
Developing Extensions

Blob.CreateOutStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an OutStream object for a binary large object (BLOB). This enables you to write data to the BLOB.

Syntax

```
Blob.CreateOutStream(OutStream: OutStream [, Encoding: TextEncoding])
```

Parameters

Blob Type: Blob An instance of the Blob data type.

OutStream
Type: OutStream
The OutStream object type that has been created.

Encoding
Type: TextEncoding
The encoding that will be used by the stream.

Optionally, you can specify the encoding on the stream. By specifying the File Handling and Text Encoding, you ensure that all the language-specific characters are represented correctly in Dynamics 365 when you read data and write data. The following example illustrates how you can set the encoding to Windows when you create a stream for a BLOB field.

```
MyTable.MyBLOBfield.CreateOutStream(MyStream, TextEncoding::WINDOWS);
```

For more information, see File Handling and Text Encoding.

See Also

Blob Data Type
Getting Started with AL
Developing Extensions

Blob.Export Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exports a binary large object (BLOB) to a file.

Syntax

```
[ExportName := ]  Blob.Export(Name: String)
```

Parameters

Blob Type: Blob An instance of the Blob data type.

Name
Type: String
The path and name of the BLOB that you want to export. When you enter the path, consider these shortcuts:

- You can omit the drive letter if the command is located on the current drive.
- You can omit the full path if the command is located in the current directory.
- You can enter only the subdirectory name if the command is located in a subdirectory of the current directory.

Return Value

ExportName Type: String The name of the created file.

See Also

Blob Data Type
Getting Started with AL
Developing Extensions

Blob.HasValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a binary large object (BLOB) has a value.

Syntax

```
HasValue :=   Blob.HasValue()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Blob Type: Blob An instance of the Blob data type.

Return Value

HasValue Type: Boolean **True** if the BLOB has a value; otherwise **false**.

See Also

Blob Data Type
Getting Started with AL
Developing Extensions

Blob.Import Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Imports a binary large object (BLOB) from a file.

Syntax

```
[ImportName := ]  Blob.Import(Name: String)
```

Parameters

Blob Type: Blob An instance of the Blob data type.

Name
Type: String
The path and name of the BLOB that you want to import. When you enter the path, consider the following shortcuts:

- You can omit the drive letter if the command is located on the current drive.
- You can omit the full path if the command is located in the current directory.
- You can enter only the subdirectory name if the command is located in a subdirectory of the current directory.

Return Value

ImportName Type: String The name of the imported file.

See Also

Blob Data Type
Getting Started with AL
Developing Extensions

Blob.Length Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the number of bytes in the binary large object (BLOB).

Syntax

```
Length :=   Blob.Length()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Blob Type: Blob An instance of the Blob data type.

Return Value

Length Type: Integer The number of bytes in the binary large object (BLOB).

See Also

Blob Data Type
Getting Started with AL
Developing Extensions

Boolean Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Indicates true or false.

See Also

Getting Started with AL
Developing Extensions

Byte Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stores a single, 8-bit character as a value in the range 0 to 255. You can easily convert this data type from a number to a character and vice versa. This means you can use mathematical operators on Byte variables.

Example

The following example assumes that you have a Byte variable named B and a Text variable named S.

You can assign a constant string of the length 1 to a Byte variable, as shown in the first line of the following code example. You can assign a single character in a Text or Code variable to a Byte variable, as shown in the second line of the following code example. You can assign a numeric value to a Byte variable, as shown in the third line of the following code example. This causes the Byte variable to contain the character from the ASCII character set that corresponds to the numeric ASCII code.

```
B := 'A';
B := S[2];
B := 65;
```

You cannot assign a character to a position greater than the position of the null terminator. For example, if the value of the text variable *MyText* is 'abc', then the null terminator is at position 4 and the following assignment causes a run-time error to occur.

```
MyText[5] := 'e';
```

See Also

Getting Started with AL
Developing Extensions

Char Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stores a single, 16-bit character as a value in the range 0 to 65535. You can convert this data type from a number to a character and vice versa. This means you can use mathematical operators on Char variables.

Example

The following example assumes that you have a Char variable named C and a Text or Code variable named S.

You can assign a constant string of the length 1 to a Char variable, as shown in the first line of the following code example. You can assign a single Char in a Text or Code variable to a Char variable, as shown in the second line of the following code example. You can assign a numeric value to a Char variable, as shown in the third line of the following code example.

```
C := 'A';
C := S[2];
C := 65;
```

You cannot assign a Char to a position greater than the position of the null terminator. For example, if the value of the Text variable *MyText* is 'abc', then the null terminator is at position 4 and the following assignment causes a run-time error to occur.

```
MyText[5] := 'e';
```

See Also

Getting Started with AL
Developing Extensions

Code Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes a special type of string that is converted to uppercase and removes any trailing or leading spaces.

Remarks

The length of a Code variable equals the number of characters in the text without leading or trailing spaces.

You must specify the length of a Code variable or field. The maximum length of a Code variable is 1024 characters. The maximum length of a Code field in a table is 2048 characters. A Code variable cannot be null. The Code data type supports Unicode.

You can index any character position in a string, such as A[65]. The resulting value will be a Char Data Type. You cannot assign a char to a position in the code variable greater than the current length of the variable +1.

Fields that contain a date formula must not have data type Code. Instead, use the DateFormula Data Type. All fields that contain a date formula with data type Code must be converted into data type DateFormula.

Example

This example shows some typical examples of code string assignments. In these examples, assume that the variable c is a code variable with a maximum length of 4.

```
c := 'ABC';
// Results in variable c, which contains 'ABC'
// and is 3 characters in length.
c := '1';
// Results in variable c, which contains '1'
// and is 1 character in length.
c := '';
// Results in variable c, which contains '' (empty string)
// and is zero (0) characters in length.
c := ' 2 ';
// Results in variable c, which contains '2'
// and is 1 character in length.
```

See Also

Getting Started with AL
Developing Extensions

Codeunit Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a container for AL code that you can use from other application objects.

The following methods are available on the Codeunit data type.

METHOD NAME	DESCRIPTION
Run(Integer [, var Record])	Loads and runs the unit of AL code you specify. To use this method, you can specify a table associated with the codeunit when you defined the codeunit properties. This allows you to pass a variable with the method. The transaction that the codeunit contains is always committed due to the Boolean return value.

The following methods are available on instances of the Codeunit data type.

METHOD NAME	DESCRIPTION
Run(var Record)	Loads and executes the unit of AL code that you specify.

See Also

Getting Started with AL
Developing Extensions

Codeunit.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and runs the unit of AL code you specify. To use this method, you can specify a table associated with the codeunit when you defined the codeunit properties. This allows you to pass a variable with the method. The transaction that the codeunit contains is always committed due to the Boolean return value.

Syntax

```
[Ok := ]  Codeunit.Run(Number: Integer [, var Record: Record])
```

Parameters

Number

Type: Integer

An integer data type that identifies the unit of AL code. If the codeunit you specify does not exist, a run-time error occurs. If you run the codeunit with a record from a table other than the one it is associated with, a run-time error occurs.

Record

Type: Record

This optional parameter identifies a record. This parameter is a record data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

This example runs two codeunits. The first uses a record parameter. The second is defined without a source table.

```
var
    FiscalYearCloseInstance: Codeunit "Fiscal Year-Close";
    AppMgmtInstance: Codeunit ApplicationManagement;
    AccountRecord: Record "Accounting Period";
begin
    AccountRecord.Init;
    if not FiscalYearCloseInstance.Run(AccountRecord) then
      Error('Codeunit run failed (with record).');
    if not AppMgmtInstance.Run then
      Error('Codeunit run failed.');
end;
```

See Also

Codeunit Data Type
Getting Started with AL

Codeunit.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the unit of AL code that you specify.

Syntax

```
[Ok := ]  Codeunit.Run(var Record: Record)
```

Parameters

Codeunit Type: Codeunit An instance of the Codeunit data type.

Record
Type: Record
A record from the table that is associated with the codeunit.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Codeunit Data Type
Getting Started with AL
Developing Extensions

CompanyProperty Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides language support for company properties.

The following methods are available on the CompanyProperty data type.

METHOD NAME	DESCRIPTION
DisplayName()	Gets the current company display name.
UrlName()	Gets the string that represents the company name in a URL.

See Also

Getting Started with AL
Developing Extensions

CompanyProperty.DisplayName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current company display name.

Syntax

```
DisplayName :=   CompanyProperty.DisplayName()
```

Return Value

DisplayName Type: String The display name of the company as specified in the **Display Name** field in the **Company** table. If no display name is specified, the company name that is stored in the **Name** field is returned.

See Also

CompanyProperty Data Type
Getting Started with AL
Developing Extensions

CompanyProperty.UrlName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the string that represents the company name in a URL.

Syntax

```
UrlName :=   CompanyProperty.UrlName()
```

Return Value

UrlName Type: String The company name in a URL.

See Also

CompanyProperty Data Type
Getting Started with AL
Developing Extensions

Database Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides access to common database functionality.

The following methods are available on the Database data type.

METHOD NAME	DESCRIPTION
ChangeUserPassword(String, String)	Changes the password for the current user.
CheckLicenseFile(Integer)	Checks a key in the license file of the system.
Commit()	Ends the current write transaction.
CompanyName()	Gets the current company name.
CopyCompany(String, String)	Creates a new company and copies all data from an existing company in the same database.
CurrentTransactionType([TransactionType])	Gets the current transaction type and sets a new type to be assigned.
DataFileInformation(Boolean, var Text, var Text, var Boolean, var Boolean, var Boolean, var Text, var DateTime, var Record)	Specifies data from a file that has been exported from a database.
ExportData(Boolean, var Text [, String] [, Boolean] [, Boolean] [, Boolean] [, Record])	Exports data from the database to a file. The data is not deleted from the database.
GetDefaultTableConnection(TableConnectionType)	Gets the default table connection based on the specified connection type. You must already have registered a table connection of this type.
HasTableConnection(TableConnectionType, String)	Verifies if a connection to an external database exists based on the specified name.
ImportData(Boolean, var Text [, Boolean] [, Boolean] [, Record])	Imports data from a file that has been exported from a database.
LockTimeout([Boolean])	Determines whether the lock time-out setting is set to On. You can also use this method to override the default setting.
RegisterTableConnection(TableConnectionType, String, String)	Registers a table connection to an external database.
SelectLatestVersion()	Forces the latest version of the database to be used.
SerialNumber()	Gets a string that contains the serial number of the license file for your system.

METHOD NAME	DESCRIPTION
ServiceInstanceId()	Gets the ID of the service instance.
SessionId()	Gets the ID of the current session.
SetDefaultTableConnection(TableConnectionType, String [, Boolean])	Establishes a connection to an external database based on a previously registered connection of the specified type.
SetUserPassword(Guid, String)	Sets a password for the user iwith the given user security ID. If the given password is blank, an empty string will be stored instead of a password hash. This will prevent the user from logging in using a password. Only SUPER can call this method. Passwords cannot be set for the empty GUID or for the default Super ID.
SID([String])	Retrieves the security identifier (SID) of a Windows user account.
TenantId()	Gets the ID of the tenant that has started the current session. Use this method when your code must be specific about which tenant database to access in a multitenant deployment. For example, if your code imports data into a cache, you can make a cache tenant-specific by using the tenant ID as a key. Also, if you want to write code that saves documents, you can include the tenant ID in the file name or location, for example. In those cases, you can use the TENANTID method in combination with the CompanyName method to identify the company and the tenant database.
UnregisterTableConnection(TableConnectionType, String)	Unregisters a table connection to an external database.
UserId()	Gets the user name of the user account that is logged on to the current session.
UserSecurityId()	Gets the unique identifier of the user that is logged on to the current session.

See Also

Getting Started with AL
Developing Extensions

Database.ChangeUserPassword Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Changes the password for the current user.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Database.ChangeUserPassword(OldPassword: String, NewPassword: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

OldPassword
Type: String
The old password for the user.

NewPassword
Type: String
The new password for the user.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.CheckLicenseFile Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks a key in the license file of the system.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Database.CheckLicenseFile(KeyNumber: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

KeyNumber
Type: Integer
The number of the key you want to check.

Remarks

The license file turns on or off different system capabilities. Use this method to check a key in the file for the current user. If the user has no access rights to the object, a message is displayed and the process is terminated.

If you omit this method, the user will have the ability to continue regardless of whether the appropriate license file is open.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.Commit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Ends the current write transaction.

Syntax

```
Database.Commit()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Remarks

When a codeunit begins, it automatically enables write transactions to be performed. When an AL code module completes, it automatically ends the write transaction by committing the updates made by the AL code.

This means that if you want the codeunit to perform a single write transaction, it is automatically handled for you. However, if you want the codeunit to perform multiple write transactions, you must use the Commit method to end one write transaction before you can start the next. The COMMIT method separates write transactions in an AL code module.

Example

The following pseudo-code example contains two write transactions. When it begins, a write transaction is automatically started. Using the Commit method, you end the first write transaction and prepare for the second. When the code completes, the second write transaction automatically ends.

```
BeginWriteTransactions
(AL Statements) // Transaction 1
COMMIT
(AL Statements) // Transaction 2
EndWriteTransactions
```

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.CompanyName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current company name.

Syntax

```
Name :=   Database.CompanyName()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

Name Type: String The name of the company, or an empty string if no company has been selected.

Example

```
var
    CompName: Text[1024];
    Text000: Label 'The name is %1.';
begin
    CompName := CompanyName;
    Message(Text000, CompName);
end;
```

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.CopyCompany Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a new company and copies all data from an existing company in the same database.

Syntax

```
[Ok := ]  Database.CopyCompany(SourceName: String, DestinationName: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

SourceName
Type: String
The name of the company that you want to copy data from.

DestinationName
Type: String
The name of the company that you want to create and copy data to. The company name can have a maximum of 30 characters. If the database collation is case-sensitive, you can have one company called Company and another called Company. However, if the database is case-insensitive, you cannot create companies with names that differ only by case.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Links and notes on records are not copied to the new company.

Example

The following example is based on the **Copy Company** batch job, which is part of the CRONUS. The batch job takes the Company system table as a data item and uses the **Name** field as the value of the *SourceName* parameter. The value of the *DestinationName* parameter is specified in the **New Company Name** field in the request page, which is represented by the `NewCompanyName` variable.

```
CopyCompany(Name, NewCompanyName);
```

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.CurrentTransactionType Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current transaction type and sets a new type to be assigned.

Syntax

```
[TransactionType := ]  Database.CurrentTransactionType([TransactionType: TransactionType])
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Parameters

TransactionType

Type: TransactionType

The type of transaction to be set for the next transaction.

Return Value

TransactionType Type: TransactionType The type of transaction for the next transaction.

Remarks

This method sets the transaction type for the next transaction that starts when you are using Microsoft SQL Server. The transaction type determines the extent of locking that is performed on data in SQL Server tables and indexes. It also determines whether modifications to data can occur within the transaction. The following basic transaction types are available:

- Browse
- Snapshot
- UpdateNoLocks
- Update
- Report

The Report transaction type maps to one of the other basic transaction types. For more information about the behavior of the transaction types, see TransactionType Property.

Using CurrentTransactionType to set a transaction type from within a currently active transaction does not affect the transaction type. You must set the transaction type before a transaction starts, which occurs at the first

database call in a trigger or in a codeunit. If you set the current transaction type to a less isolated transaction behavior, for example, if you try to change an Update transaction to a Browse transaction, the method call is ignored. If you try to change the current transaction type to a more isolated transaction behavior, for example, from Browse to Update, you will receive an error message.

Example 1

In Example 1, CurrentTransactionType is used to set the required behavior of the next transaction in the database.

```
//Example 1
CurrentTransactionType := TransactionType::UpdateNoLocks;
```

Example 2

In Example 2, CurrentTransactionType is used to return the transaction type setting for the current transaction.

```
//Example 2
if CurrentTransactionType = TransactionType::UpdateNoLocks then...
```

Example 3

Examples 3 and 4 show how to use the CurrentTransactionType method to set the transaction type for two separate transactions.

When you set the transaction type as in Example 3, you will get an update (or write) transaction using Serializable behavior, which means that modifications are allowed within this transaction. SQL Server will guarantee the serializability of the transaction by placing the appropriate locks when you read from the table in the database.

```
//Example 3
CurrentTransactionType := TransactionType::Update;
```

Example 4

Examples 3 and 4 show how to use the CurrentTransactionType method to set the transaction type for two separate transactions.

When you set the transaction type as in Example 4, you will get a read-only, non-locking transaction. This means that no modifications are allowed within this transaction and that SQL Server does not add any locks. It is also possible to read any uncommitted data.

```
//Example 4
CurrentTransactionType := TransactionType::Browse;
```

Example 5

In Example 5, the initial transaction type is UpdateNoLocks. This is usually the default transaction type for a trigger. When you try to change the transaction type from UpdateNoLocks to Browse, which is a less isolated transaction type, the method call is ignored. The current transaction type remains UpdateNoLocks. After the first Table.Get line, again, when you try to change the transaction type from UpdateNoLocks to Browse, the method

call is ignored. The current transaction type remains UpdateNoLocks. Next, when you try to change the transaction type from UpdateNoLocks to Update, which is a more isolated transaction type, the method call causes an error message to be generated. The current transaction type remains UpdateNoLocks. After the Commit Method, you can set a new transaction type and change it again if it is required. After the second Table.Get line, the CurrentTransactionType method call is ignored. The transaction type remains Update.

```
//Example 5
OnInsert
CurrentTransactionType := TransactionType::Browse;
Table.Get;
CurrentTransactionType := TransactionType::Browse;
CurrentTransactionType := TransactionType::Update;
Commit;
CurrentTransactionType := TransactionType::Browse;
CurrentTransactionType := TransactionType::Update;
Table.Get;
CurrentTransactionType := TransactionType::Browse;
OnInsert.RETURN
Commit;
```

Example 6

In Example 6, the initial transaction type is UpdateNoLocks. The first two times that you try to change to a less isolated transaction type, the method call is ignored. Finally, when you try to change to a more isolated transaction type, the method call causes an error message to be generated.

```
//Example 6
CodeUnit
CurrentTransactionType := TransactionType::Snapshot;
Report 1
CurrentTransactionType := TransactionType::Browse;
Report 2
CurrentTransactionType := TransactionType::Update;
```

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.DataFileInformation Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies data from a file that has been exported from a database.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Database.DataFileInformation(ShowDialog: Boolean, var FileName: Text, var Description: Text, var
HasApplication: Boolean, var HasApplicationData: Boolean, var HasGlobalData: Boolean, var tenantId: Text,
var exportDate: DateTime, var CompanyRecord: Record)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

ShowDialog
Type: Boolean
Specifies if you want to display a dialog box where the user can confirm the action.

FileName
Type: Text
Specifies the name and location of the file that you want to read information from. The file must have been exported from a database.

Description
Type: Text

HasApplication
Type: Boolean
Specifies if the file contains application objects. Create a variable of type Boolean to specify this parameter.

HasApplicationData
Type: Boolean
Specifies if the file contains the data that defines the application in the database. This includes the permissions, permission sets, profiles, and style sheets. Create a variable of type Boolean to specify this parameter.

HasGlobalData
Type: Boolean
Specifies if the file contains global, non-company specific data. Create a variable of type Boolean to specify this parameter.

tenantId

Type: Text

Specifies the tenant ID of the database that the data was exported from. Create a variable of type Text to specify this parameter.

exportDate

Type: DateTime

Specifies the date and time when the data was exported. Create a variable of type DateTime to specify this parameter.

CompanyRecord

Type: Record

Specifies the company or companies in the file.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.ExportData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exports data from the database to a file. The data is not deleted from the database.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Database.ExportData(ShowDialog: Boolean, var FileName: Text [, Description: String] [,
IncludeApplication: Boolean] [, IncludeApplicationData: Boolean] [, IncludeGlobalData: Boolean] [,
CompanyRecord: Record])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

ShowDialog
Type: Boolean
Specifies if you want to display a dialog box where the user can confirm the action.

FileName
Type: Text
Specifies the name and location of the file that the data must be exported to. The file must have the .navdata extension.

Description
Type: String
Specifies a description for the exported data.

IncludeApplication
Type: Boolean
Specifies if you want to export the application objects. Create a variable of type Boolean to specify this parameter.

IncludeApplicationData
Type: Boolean
Specifies if you want to export the data that defines the application in the database. This includes the permissions, permission sets, profiles, and style sheets. Create a variable of type Boolean to specify this parameter.

IncludeGlobalData
Type: Boolean

Specifies if you want to export global, non-company specific data. Create a variable of type Boolean to specify this parameter.

CompanyRecord
Type: Record
Specifies the company or companies that must be imported.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.GetDefaultTableConnection Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the default table connection based on the specified connection type. You must already have registered a table connection of this type.

Syntax

```
Name :=   Database.GetDefaultTableConnection(Type: TableConnectionType)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Type
Type: TableConnectionType
The type of table connection as defined in the TableType property.

Return Value

Name Type: String

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.HasTableConnection Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Verifies if a connection to an external database exists based on the specified name.

Syntax

```
Ok :=   Database.HasTableConnection(Type: TableConnectionType, Name: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Type
Type: TableConnectionType
Specifies the type of table connection as defined in the TableType property.

Name
Type: String
The name of the external table connection. You must already have registered a table connection with this name.

Return Value

Ok Type: Boolean **true** if a connection to an external database exists for the specified name, otherwise **false**.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.ImportData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Imports data from a file that has been exported from a database.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Database.ImportData(ShowDialog: Boolean, var FileName: Text [, IncludeApplicationData: Boolean] [,
IncludeGlobalData: Boolean] [, CompanyRecord: Record])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

ShowDialog
Type: Boolean
Specifies if you want to display a dialog box where the user can confirm the action.

FileName
Type: Text
Specifies the name and location of the file that must be imported. The file must have been exported from a database .
IncludeApplicationData
Type: Boolean

Specifies if you want to import the data that defines the application in the database. This includes the permissions, permission sets, profiles, and style sheets. Create a variable of type Boolean to specify this parameter. To import application objects, you must use the Import-NAVData Windows PowerShell cmdlet.

IncludeGlobalData
Type: Boolean

Specifies if you want to import global, non-company specific data. Create a variable of type Boolean to specify this parameter.

CompanyRecord
Type: Record
Specifies the company or companies that must be imported.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.LockTimeout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the lock time-out setting is set to On. You can also use this method to override the default setting.

Syntax

```
[LockTimeout := ]  Database.LockTimeout([LockTimeout: Boolean])
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Parameters

LockTimeout

Type: Boolean

The new setting for whether the lock time-out is on.

Return Value

LockTimeout Type: Boolean This value shows whether to use a lock time-out.

Remarks

This method has been designed specifically for use in long-running processes that should not be terminated because of a lock time-out, for example batch jobs that run overnight.

When the AL code has finished running, the default setting is used again. This method does not change the duration of a lock time-out.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.RegisterTableConnection Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Registers a table connection to an external database.

Syntax

```
Database.RegisterTableConnection(Type: TableConnectionType, Name: String, Connection: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Type

Type: TableConnectionType

Specifies the type of table connection as defined in the TableType property.

Name

Type: String

Specifies the name of the connection in your code, or the name of the primary key field on the table.

Connection

Type: String

Specifies the connection to the external database.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.SelectLatestVersion Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Forces the latest version of the database to be used.

Syntax

```
Database.SelectLatestVersion()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Remarks

This method makes sure that the data displayed is the most current data in the database. The method clears all non-locked records from the client cache, thereby ensuring that you read the most recent data. The method also clears the current client session's cache for the CaptionClassTranslate strings. The strings will then be reevaluated by the **CaptionClassTranslate** method trigger in codeunit 42.

> **WARNING**
> Clearing the cache and reading data directly from the database adversely affects performance.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.SerialNumber Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a string that contains the serial number of the license file for your system.

Syntax

```
String :=   Database.SerialNumber()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

String Type: String The serial number.

Example

```
var
    SN: Text[30];
    Text000: Label 'The serial number for this software package is:%1.';
begin
    SN := SerialNumber;
    Message(Text000, SN);
end;
```

The output of this example is as follows:

The serial number for this software package is:

W1-ZA-002-6R75A-7

NOTE

The serial number depends on your licensed version of Dynamics 365. The serial number shown here is an example.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.ServiceInstanceId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the ID of the service instance.

Syntax

```
ID :=   Database.ServiceInstanceId()
```

NOTE

This method can be invoked without specifying the data type name.

Return Value

ID Type: Integer The ID of the service instance.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.SessionId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the ID of the current session.

Syntax

```
ID :=   Database.SessionId()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

ID Type: Integer The ID of the current session.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.SetDefaultTableConnection Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Establishes a connection to an external database based on a previously registered connection of the specified type.

Syntax

```
Database.SetDefaultTableConnection(Type: TableConnectionType, Name: String [, Scoped: Boolean])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Type
Type: TableConnectionType
The type of table connection as defined in the TableType property.

Name
Type: String
The name of the external table connection. You must already have registered a table connection with this name.

Scoped
Type: Boolean
If true, when the method ends where you have used SETDEFAULTTableCONNECTION, the default table connection returns to the value it had before. Use the Scope parameter when you want to use a specific table connection for a specific task and then return to the normal configuration.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.SetUserPassword Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a password for the user iwith the given user security ID. If the given password is blank, an empty string will be stored instead of a password hash. This will prevent the user from logging in using a password. Only SUPER can call this method. Passwords cannot be set for the empty GUID or for the default Super ID.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Database.SetUserPassword(USID: Guid, Password: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

USID
Type: Guid
User security ID of the user for which to set the password.

Password
Type: String
The password to set for the user.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.SID Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the security identifier (SID) of a Windows user account.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
SID :=   Database.SID([UserAccount: String])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

UserAccount
Type: String
The Windows user account for which you want to get the SID. You must specify a domain and user name, such as 'cronus\simon'.

Return Value

SID Type: String The SID of the specified Windows user account.

Remarks

If you create a page for adding Windows logins, then you must use the SID method to retrieve the SID for the user account so that you can enter the new login into the Windows Login table.

This method runs only on the computer that is running Dynamics 365 Business Central service. If you call this method from the client computer, then no action occurs.

Example

```
var
    NewSID: Text[119];
    UserAccount: Text[132];
begin
    UserAccount := 'cronus\simon';
    NewSID := SID(UserAccount);
end;
```

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.TenantId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the ID of the tenant that has started the current session. Use this method when your code must be specific about which tenant database to access in a multitenant deployment. For example, if your code imports data into a cache, you can make a cache tenant-specific by using the tenant ID as a key. Also, if you want to write code that saves documents, you can include the tenant ID in the file name or location, for example. In those cases, you can use the TENANTID method in combination with the CompanyName method to identify the company and the tenant database.

Syntax

```
ID :=   Database.TenantId()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

ID Type: String The ID of the tenant that has started the current session.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.UnregisterTableConnection Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Unregisters a table connection to an external database.

Syntax

```
Database.UnregisterTableConnection(Type: TableConnectionType, Name: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Type

Type: TableConnectionType

Specifies the type of table connection as defined in the TableType property. If the table is of type ExternalSQL, UNREGISTERTableCONNECTION rolls back the current transaction.

Name

Type: String

Specifies the name of the connection in your code, or the name of the primary key field on the table.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.UserId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the user name of the user account that is logged on to the current session.

Syntax

```
ID :=   Database.UserId()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

ID Type: String This string contains the value of the User Name field in table 2000000120, the User table, for the current user.

Example

```
User := UserId;
Message('The system was started by %1', User);
```

The following is an example of the output of the previous code:

The system was started by cronus\simon.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Database.UserSecurityId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the unique identifier of the user that is logged on to the current session.

Syntax

```
USID :=   Database.UserSecurityId()
```

NOTE

This method can be invoked without specifying the data type name.

Return Value

USID Type: Guid The ID that is assigned to the user by the application. This is the value of the User Security ID field in table 2000000120, the User table, for the current user.

See Also

Database Data Type
Getting Started with AL
Developing Extensions

Date Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes a date ranging from January 1, 1753 to December 31, 9999.

The displayed text format of the date is determined by your Region and Language Format setting in Windows.

Undefined dates

An undefined or blank date is specified by 0D. The undefined date is considered to be before all other dates.

Normal dates and closing dates

All normal dates have a corresponding closing date. The closing date for a given date is defined as a period of time that follows a given normal date and precedes the next normal date.

Syntax

The syntax for defining DateTime format follows the ISO standard.

- The syntax for defining Date format is `yyyymmddD`, where `D` is a mandatory letter. For example, `20180325D`, read as the 25th of March, 2018.

To assign a normal date to a variable, use the following format: `yyyymmddD`.

Storing dates in the SQL server database

SQL Server stores information about both date and time in columns of the DateTime types. For date fields, Dynamics 365 Business Central uses only the date and uses a constant value for the time. For a normal date, this constant value contains 00:00:00:000. For a closing date, it contains 23:59:59:000.

The Dynamics 365 Business Central undefined date is represented by the earliest valid date in SQL Server. The earliest valid date in SQL Server for a DateTime is 01-01-1753 00:00:00:000.

If you store a date in the database that is outside the valid range for a SQL DateTime, a run-time error occurs.

Example

This example shows a valid assignment of date. This example is compiled and run on a computer with the regional format set to English (United States).

```
var
    Date1: Date;
begin
    Date1 := 20180612D;
    Message(Format(Date1));
end;
```

The message window displays the following:

06/12/2018

See Also

Getting Started with AL
Developing Extensions

DateFormula Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a date formula that has the same capabilities as an ordinary input string for the CALCDATE Method (Date). The DateFormula data type is used to provide multilanguage capabilities to the CALCDATE Method (Date).

Remarks

When a date calculation formula is stored in a DateFormula field, it is converted to a generic, non-language dependent format. When a date calculation formula is retrieved from a DateFormula field, it is converted to a valid date conversion string for the currently selected language.

To assign a value to a DateFormula data type, whether it is a field or a variable, you must use the EVALUATE Method.

Example

This example requires that you create a DateFormulaVariable variable that is a DateFormula data type.

```
if Format(DateFormulaVariable) = ' ' then
  Evaluate(DateFormulaVariable, '1W');
```

You must use the Format Method to make a comparison with a text string. If you do not use this method, then the IF statement will fail because you cannot compare a DateFormula data type with a Text data type.

See Also

Getting Started with AL
Developing Extensions

DateTime Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes a date and time ranging from January 1, 1753, 00:00:00.000 to December 31, 9999, 23:59:59.999. An undefined or blank DateTime is specified by 0DT.

The displayed text format of a DateTime is determined by your Regional and Language Options in Windows.

Remarks

A DateTime is stored in the database as Coordinated Universal Time (UTC). UTC is the international time standard (formerly Greenwich Mean Time, or GMT). Zero hours UTC is midnight at 0 degrees longitude.

The DateTime is always displayed as local time in Dynamics 365 Business Central. Local time is determined by the time zone regional settings used by your computer. You must always enter DateTimes as local time. When you enter a DateTime as local time, it is converted to UTC using the current settings for the time zone and daylight savings time.

The DateTime data type does not support closing dates.

By default, DateTimes are displayed using the standard display format. When you use the standard display format, seconds and milliseconds are not displayed until you select the DateTime field. Furthermore, if you export your data using an XMLport or by writing it to a file, the seconds and milliseconds are not exported unless you specify that DateTime fields use another format and display this information. For more information about how DateTime objects are displayed and the formats that are available, see Format Property.

The only constant available when you use the DateTime data type is the undefined DateTime, 0DT. To assign a constant value to a DateTime variable you must use the CreateDateTime method.

If you use a date that is outside the valid date range, a run-time error occurs.

Syntax

The syntax for defining DateTime format follows the ISO standard.

- The syntax for defining Date format is `yyyymmddD`, where `D` is a mandatory letter. For example, `20180325D`, read as the 25th of March, 2018.
- The syntax for defining Time format is `hhmmssT`, where `T` is the time designator. For example, `093125H`, read as 9:13:25.

SQL Server

In SQL Server, the earliest permitted DateTime is January 1, 1753, 00:00:00.000. The latest permitted DateTime is December 31, 9999, 23:59:59.999. If you store a date in the database that is outside the valid range for a SQL DateTime, a runtime error occurs.

See Also

Getting Started with AL
Developing Extensions

CurrentDateTime Method

Format Property

Debugger Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Enables communication with a debugger.

> **NOTE**
> This data type is supported only in Business Central on-premises.

The following methods are available on the Debugger data type.

METHOD NAME	DESCRIPTION
Activate()	Activates the debugger and attaches the debugger to the next session that is started.
Attach(Integer)	Activates the debugger and attaches it to the specified session.
Break()	Breaks code execution of a debugging session.
BreakOnError(Boolean)	Sets whether the debugger breaks on errors.
BreakOnRecordChanges(Boolean)	Breaks execution before a change to a record occurs.
Continue()	Executes code until the next breakpoint or until execution ends.
Deactivate()	Deactivates the debugger.
DebuggedSessionID()	Gets the ID of the previous session that the debugger was attached to.
DebuggingSessionID()	Gets the ID of the session that the debugger is currently attached to.
EnableSqlTrace(Integer [, Boolean])	Enables or verifies SQL tracing. If you enable SQL tracing, then SQL Server events for selected sessions on the server instance are collected.
GetLastErrorText()	Gets the last error that occurred in the debugger.
IsActive()	Indicates whether the debugger is active.
IsAttached()	Specifies if the debugger is attached to a session.
IsBreakpointHit()	Specifies if a breakpoint is hit in a debugging session.

METHOD NAME	DESCRIPTION
SkipSystemTriggers(Boolean)	Enables the debugger to skip code that is inside system triggers.
StepInto()	Executes a method call and then stops at the first line of code inside the method.
StepOut()	Enables debugging to return to the calling method after it steps into a method call.
StepOver()	Executes a method call and then stops at the first line outside the method call.
Stop()	Stops execution as if the code hits an error.

> **NOTE**
> The Dynamics 365 Business Central Debugger is an example of a debugger application that is built using tables, pages, codeunits, and the AL debugger methods.

See Also

Getting Started with AL
Developing Extensions

Debugger.Activate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Activates the debugger and attaches the debugger to the next session that is started.

Syntax

```
[Ok := ]  Debugger.Activate()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.Attach Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Activates the debugger and attaches it to the specified session.

Syntax

```
[Ok := ]  Debugger.Attach(SessionID: Integer)
```

Parameters

SessionID

Type: Integer

The ID of the session that you want to attach the debugger to.The session can be any of the following:

- Windows client
- Web client
- NAS services session
- SOAP web services client session
- OData web services client session
- Background session

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The **Attach** method behaves like the **Debug** action on the **Sessions** page. You can call the **Attach** method to activate the debugger.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.Break Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Breaks code execution of a debugging session.

Syntax

```
[Ok := ]  Debugger.Break()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.BreakOnError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Sets whether the debugger breaks on errors.

Syntax

```
[Ok := ]  Debugger.BreakOnError(Ok: Boolean)
```

Parameters

Ok

Type: Boolean

Specifies whether the debugger breaks on errors.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

Remarks

The debugger must be attached to a session when you run the BreakOnError method.

Example

This example shows how to implement the code for an action on a page that sets the break on error setting to **true**. You can use this code example in the OnAction trigger of that action. This code example requires that you create a Boolean variable named *Value*.

```
Value := true;
Debugger.BreakOnError(Value);
```

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.BreakOnRecordChanges Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Breaks execution before a change to a record occurs.

Syntax

```
[Ok := ]  Debugger.BreakOnRecordChanges(Ok: Boolean)
```

Parameters

Ok

Type: Boolean

Specifies whether the debugger should break execution when a change to a record occurs. If Ok is true, then the debugger breaks before creating, updating, or deleting a record.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.Continue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Executes code until the next breakpoint or until execution ends.

Syntax

```
[Ok := ]  Debugger.Continue()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.Deactivate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Deactivates the debugger.

Syntax

```
[Ok := ]  Debugger.Deactivate()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.DebuggedSessionID Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Gets the ID of the previous session that the debugger was attached to.

Syntax

```
SessionID :=   Debugger.DebuggedSessionID()
```

Return Value

SessionID Type: Integer

The session ID that the debugger is currently attached to.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.DebuggingSessionID Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Gets the ID of the session that the debugger is currently attached to.

Syntax

```
SessionID :=   Debugger.DebuggingSessionID()
```

Return Value

SessionID Type: Integer

The session ID that the debugger is currently attached to.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.EnableSqlTrace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Enables or verifies SQL tracing. If you enable SQL tracing, then SQL Server events for selected sessions on the server instance are collected.

Syntax

```
[IsEnabled := ]  Debugger.EnableSqlTrace(SessionID: Integer [, NewIsEnabled: Boolean])
```

Parameters

SessionID

Type: Integer

The ID of the session for which you want to enable the SQL trace, or for which you want to verify if tracing is enabled.If you specify 0 and you specify the NewIsEnabled parameter, then tracing is enabled for all existing sessions and all new sessions on the current server instance. If you specify 0 and you omit the NewIsEnabled parameter, then the function returns true if tracing is enabled for new sessions on the current server instance. If the session ID that you specify does not exist and you specify the NewIsEnabled parameter, then a run-time error occurs. If the session ID that you specify does not exist and you do not specify the NewIsEnabled parameter, then the function returns false.

NewIsEnabled

Type: Boolean

If you specify the optional NewIsEnabled parameter, then the method sets whether tracing is enabled. true if you want to enable tracing for the specified session; false if you want to disable tracing. If you omit the NewIsEnabled parameter, then the function verifies if tracing is enabled.

Return Value

IsEnabled Type: Boolean

Remarks

You use Microsoft SQL Server Profiler to view traces. For more information, see Using SQL Server Profiler. To start SQL Server Profiler, in **SQL Server Management Studio**, on the **Tools** menu, choose **SQL Server Profiler**.

You can also enable and disable SQL tracing by using the **Start Full SQL Tracing** and **Stop Full SQL Tracing** buttons on the **Session List** page.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.GetLastErrorText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Gets the last error that occurred in the debugger.

Syntax

```
String :=   Debugger.GetLastErrorText()
```

Return Value

String Type: String The text string that was contained in the last error message that was generated by Microsoft Dynamics Business Central. If no error has occurred, then the function returns an empty string.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.IsActive Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Indicates whether the debugger is active.

Syntax

```
Ok :=   Debugger.IsActive()
```

Return Value

Ok Type: Boolean

Remarks

When the debugger is activated, it is in one of the following states:

- Attached to a session.
- Waiting to attach to a session.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.IsAttached Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Specifies if the debugger is attached to a session.

Syntax

```
Ok :=   Debugger.IsAttached()
```

Return Value

Ok Type: Boolean

true if the debugger is attached to a session; otherwise, **false**.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.IsBreakpointHit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Specifies if a breakpoint is hit in a debugging session.

Syntax

```
Ok :=   Debugger.IsBreakpointHit()
```

Return Value

Ok Type: Boolean

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.SkipSystemTriggers Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Enables the debugger to skip code that is inside system triggers.

Syntax

```
[Ok := ]  Debugger.SkipSystemTriggers(Ok: Boolean)
```

Parameters

Ok

Type: Boolean

Specifies if the debugger should skip system triggers.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.StepInto Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Executes a method call and then stops at the first line of code inside the method.

Syntax

```
[Ok := ]  Debugger.StepInto()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.StepOut Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Enables debugging to return to the calling method after it steps into a method call.

Syntax

```
[Ok := ]  Debugger.StepOut()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.StepOver Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Executes a method call and then stops at the first line outside the method call.

Syntax

```
[Ok := ]  Debugger.StepOver()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Debugger.Stop Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 4.0 where it was deprecated.*

Stops execution as if the code hits an error.

Syntax

```
[Ok := ]  Debugger.Stop()
```

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If you omit this optional return value and if the break is not set successfully, then a run-time error occurs. If you include the return value, then you must handle any errors.

See Also

Debugger Data Type
Getting Started with AL
Developing Extensions

Decimal Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes decimal numbers ranging from -999,999,999,999,999.99 to +999,999,999,999,999.99.

Example 1

The following are examples of decimal values.

```
546.88
3425.57
```

Example 2

The following is not a decimal, but rather an Integer Data Type.

```
342
```

Limits on Decimal Data Type Variables

The Decimal data type is mapped to the Microsoft .NET Framework common language runtime (CLR) Decimal data type, which controls and the precision and limits for variables.

The following table shows the limits for variables of type `Decimal`.

LIMIT	VALUE
Maximum format value. This is the maximum value that can be: - Formatted into a TEXT variable by the Format function. - Input from the UI or XMLPorts. - Assigned directly in source code.	+/- 999,999,999,999,999.99
Maximum field data type value. This is the maximum value that a field variable in a record can hold while not being persisted.	+/- 999,999,999,999,999.99
Maximum persisted value. This is the maximum value that can be stored in the database.	Can read previous stored values but cannot store values outside the formatting range since field variables cannot be assigned values outside the formatting range.

LIMIT	VALUE
Maximum calculating value. This is the maximum value that can be calculated by code statements while not assigning to a field variable, storing to the database, or formatting to a text variable.	+/- 79,228,162,514,264,337,593,543,950,335
Scaling factor (digits after decimal point) for calculating values	28 For example, 7.9228162514264337593543950335

The maximum safe value that will work on all Business Central versions of is +/- 999,999,999,999,999.99.

It is possible to assign to a variable the maximum value that can be formatted and then multiply that variable by a large positive number, thereby generating a greater value. However, we do not recommend doing this. If you do, you will get errors if you attempt to format this variable to a text variable or assign the variable to a field variable in a record.

See Also

Getting Started with AL
Developing Extensions

Dialog Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a dialog window.

The following methods are available on the Dialog data type.

METHOD NAME	DESCRIPTION
Confirm(String [, Boolean] [, Any,...])	Creates a dialog box that prompts the user for a yes or no answer. The dialog box is centered on the screen.
Error(String [, Any,...])	Displays an error message and ends the execution of AL code.
Error(ErrorInfo)	Displays an error message and ends the execution of AL code.
LogInternalError(String, DataClassification, Verbosity)	Log internal errors for telemetry.
LogInternalError(String, String, DataClassification, Verbosity)	Log internal errors for telemetry.
Message(String [, Any,...])	Displays a text string in a message window.
StrMenu(String [, Integer] [, String])	Creates a menu window that displays a series of options.

The following methods are available on instances of the Dialog data type.

METHOD NAME	DESCRIPTION
Close()	Closes a dialog window that has been opened by the Open method.
HideSubsequentDialogs([Boolean])	Specifies that subsequent child dialogs are not shown.
Open(String [, var Any,...])	Opens a dialog window.
Update([Integer] [, Any])	Updates the value of a '#'-or '@' field in the active window.

See Also

Getting Started with AL
Developing Extensions

Dialog.Confirm Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a dialog box that prompts the user for a yes or no answer. The dialog box is centered on the screen.

Syntax

```
Ok :=   Dialog.Confirm(String: String [, Default: Boolean] [, Value1: Any,...])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

Specifies the string that is displayed in the dialog box. Use a backslash (\) to indicate a new line. The string can be a text constant that is enabled for multilanguage functionality.

Default

Type: Boolean

Specifies the default button. If you do not specify a default button, then No is used as the default button.

Value1

Type: Any

Return Value

Ok Type: Boolean

Remarks

The message window is automatically sized. The height of the window corresponds to the number of lines and the width corresponds to the length of the longest line.

We recommend that you always end `Confirm` messages with a question mark. For more information about best practices for end-user messages, see Progress Windows, Message, Error, and Confirm Methods.

Example

In the following example, the Dialog.Confirm method prompts the user for a **true** or **false** answer.

```
var
    Question: Text;
    Answer: Boolean;
    CustomerNo: Integer;
    Text000: Label 'Exit without saving changes to customer %1?';
    Text001: Label 'You selected %1.';
begin
    CustomerNo := 01121212;
    Question := Text000;
    Answer := Dialog.Confirm(Question, true, CustomerNo);
    Message(Text001, Answer);
end;
```

The first dialog box shows:

Exit without saving changes to customer 1121212?

If you select the default **true** value, then the second dialog box is shown:

You selected true.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.Error Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Displays an error message and ends the execution of AL code.

Syntax

```
Dialog.Error(Message: String [, Value: Any,...])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Message

Type: String

This string contains the text of the error message you want to display to the user. Use percent signs (%) or number signs (#) to insert variable values into the string. Place the percent or number signs where you want to substitute the variable value. The string can be a text constant that is enabled for multilanguage functionality.

Value

Type: Any

Any variable or expression to be inserted in String. You can insert up to 10 values. For '#'-type fields, the value is truncated according to the total number of number-sign characters in String. For '%'-type fields, the full length of the value is printed.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.Error Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Displays an error message and ends the execution of AL code.

> **NOTE**
>
> This method is supported only in Business Central on-premises.

Syntax

```
Dialog.Error(Message: ErrorInfo)
```

> **NOTE**
>
> This method can be invoked without specifying the data type name.

Parameters

Message

Type: ErrorInfo

The ErrorInfo structure that contains error message, error type, verbosity, and data classification.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.LogInternalError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Log internal errors for telemetry.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Dialog.LogInternalError(Message: String, DataClassificationInstance: DataClassification, VerbosityInstance: Verbosity)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Message
Type: String
This string contains the text of the error message you want to log into telemetry. It is not what the user will get, they will only get a generic error message.

DataClassificationInstance
Type: DataClassification
Sets the classification of the data in the error message.

VerbosityInstance
Type: Verbosity
Represents the security level of events.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.LogInternalError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Log internal errors for telemetry.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Dialog.LogInternalError(Message: String, SubstitutionString: String, DataClassificationInstance:
DataClassification, VerbosityInstance: Verbosity)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Message
Type: String
This string contains the text of the error message you want to log into telemetry. Use a percent sign (%) to insert a variable value into the string. Place the percent where you want the system to substitute the variable value. You may only insert one variable value. It is not what the user will get, they will only get a generic error message.

SubstitutionString
Type: String
This string replaces a percent sign in the "Message" Parameter.

DataClassificationInstance
Type: DataClassification
Sets the classification of the data in the error message.

VerbosityInstance
Type: Verbosity
Represents the security level of events.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.Message Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Displays a text string in a message window.

Syntax

```
Dialog.Message(String: String [, Value: Any,...])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

String

Type: String

This string contains the text you want the system to display in the message window. Use a backslash (\) to start a new line. Use percent signs (%) to insert variable values into the string. Place the percent where you want the system to substitute the variable value. The string can be a text constant that is enabled for multilanguage functionality.

Value

Type: Any

Any type of AL variable you want to insert into String. You can insert up to 10 values.

Remarks

When a message statement in the AL code is executed, the message is not immediately displayed. Instead, it is displayed after the AL code is finished executing or after the AL code pauses to wait for user interaction.

The window is automatically sized to hold the longest line of text and the total number of lines.

For NAS sessions or Dynamics 365 sessions (including NAS) that are started by the StartSession Method (Sessions), messages are recorded in the event log of the computer that is running Dynamics 365 Business Central service. The message entries have the ID 100 and type Information.

Programming Guidelines

We recommend the following guidelines for messages:

- Always end a message with a period.
- Supply the user with a message when the system has finished performing a task.
- Write the message in past tense.

For more information, see Progress Windows, Message, Error, and Confirm Methods.

Example

This example shows how to use the `Message` method.

```
begin
        Message('App published: Hello world');

        Text := 'ABCDE';
        Number := 12345.678;

        // The backslash indicates a new line.
        // You can concatenate strings using the + operator.
        // You can insert variable values using the % symbol.
        Message(Text000 + Text001 + '%1\' + Text002 + '%2\', Number, Text);
    end;


    var
        Text: Text;
        Number: Decimal;
        Text000: Label 'You can use message windows to display text and numbers: \';
        Text001: Label 'The number: ';
        Text002: Label 'The text: ';
```

The message window reads:

You can use message windows to display text and numbers:

The number: 12,345.678

The text: ABCDE

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.StrMenu Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a menu window that displays a series of options.

Syntax

```
OptionNumber :=   Dialog.StrMenu(OptionMembers: String [, DefaultNumber: Integer] [, Instruction: String])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

OptionMembers

Type: String

A comma-separated string. Each substring in OptionString specifies an option on the menu. The string can be a text constant that is enabled for multilanguage functionality.

DefaultNumber

Type: Integer

Use this optional parameter to determine a default option, which is highlighted. The options are numbered 1, 2, 3, 4, and so on. If you omit this optional parameter, the first option (1) is used as the default.

Instruction

Type: String

Use this optional parameter to add a description to the option values.

Return Value

OptionNumber Type: Integer The number of the menu option that the user selected. If the user presses the Esc key to exit the menu, zero (0) is returned.

Example

This example shows how to use the Dialog.StrMenu method.

```
var
    Options: Text[30];
    Selected: Integer;
    Text000: Label 'Save,Delete,Exit,Find';
    Text001: Label 'You selected option %1.';
    Text002: Label 'Choose one of the following options:';
begin
    Options := Text000;
    // Sets the default to option 3
    Selected := Dialog.StrMenu(Options, 3, Text002);
    Message(Text001, Selected);
end;
```

The menu window displays the following text:

Choose one of the following options:

Save

Delete

Exit

Find

Option 3, **Exit**, is highlighted. The option that the user selects is stored in the variable *Selected*. The user receives following message:

You selected option 3.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes a dialog window that has been opened by the Open method.

Syntax

```
Dialog.Close()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Dialog Type: Dialog An instance of the Dialog data type.

Remarks

If `Close` is used on a dialog window that is not open, a run-time error will occur. The dialog windows of an object are automatically closed when the object terminates.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.HideSubsequentDialogs Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies that subsequent child dialogs are not shown.

Syntax

```
[HideSubsequentDialogs := ]  Dialog.HideSubsequentDialogs([HideSubsequentDialogs: Boolean])
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Parameters

Dialog Type: Dialog An instance of the Dialog data type.

HideSubsequentDialogs

Type: Boolean

A value specifying whether to hide subsequent dialogs.

Return Value

HideSubsequentDialogs Type: Boolean **True** if HideSubsequentDialogs is set to true; otherwise, false.

Remarks

You must call the `HideSubsequentDialogs` method on the dialog variable before the Open Method. Until the Open Method is called on this variable, calls on other dialog variables will behave as normal.

Example

The following code illustrates how the `HideSubsequentDialogs` method works with two dialog variables.

```
var
  MyDialog1 : Dialog;
  MyDialog2 : Dialog;
  Text000 : Label 'additional text';
  begin

    // The HideSubsequentDialogs method is used on MyDialog1 dialog.
    MyDialog1.HideSubsequentDialogs := true;

    // When MyDialog1 dialog opens, it will register as the root dialog.
    MyDialog1.Open('Dialog 1');
    Sleep(2000);

    // MyDialog2 dialog will not open. However, the code associated with the Open call will run as if it was
actually opened.
    MyDialog2.Open('Dialog 2');
    Sleep(2000);

    // Updating MyDialog2 dialog will have no effect
    MyDialog2.Update(1, Text000);
    Sleep(2000);

    // MyDialog1 dialog will open
    MyDialog1.Open('Dialog 1 #1', Text000);
    Sleep(2000);

    // As soon as MyDialog1 dialog is closed, other can be reopened and they will no longer be hidden
    MyDialog1.Close;

    MyDialog2.Open('Dialog 2');
    Sleep(2000);

  end;
```

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.Open Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Opens a dialog window.

Syntax

```
Dialog.Open(String: String [, var Variable1: Any,...])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Dialog Type: Dialog An instance of the Dialog data type.

String

Type: String

This string contains the text that you want to display in the window. Use a backslash (\) to start a new line. Use number signs (#) to insert variable values into the string. Place the number signs where you want to substitute the variable value. Place a number in the part of the string where a variable value will be substituted (for example, #1####) to be able to reference this field for updating. The number of # characters in the string indicates the length of the field. You can update the fields using the UPDATE method (Dialog) or by letting the user edit the values.

Variable1

Type: Any

Use these optional parameters to specify variables for field1, field2, and so on in the String.

Remarks

Dialog windows that are opened by an object are closed when the object terminates.

Dialog windows are automatically sized to hold the longest line of text and the total number of lines.

We recommend the following guidelines:

- Enter messages as text constants.
- Write messages using active voice. For example, write "Processing items" instead of writing "Items are being processed."
- Align the # field to the left with at least one space character between the text and the variable.

NOTE

With the Dynamics NAV Client connected to Business Central, you can use @ characters instead of # characters for the *String* parameter to display the value as percentage and a progress indicator. The percentage value that is displayed is the percentage of the variable value from 0 to 9999. This is not supported in the Business Central Web client.

Example 1

This example shows how to use the dialog.Open method.

```
var
    MyDialog: Dialog;
    MyNext: Integer;
    Text000: Label 'Counting to 4 #1:';
begin
    MyNext := 0;
    MyDialog.Open(Text000,MyNext);
    repeat
      // Do some processing.
      Sleep(1000);
      MyNext := MyNext + 1;
      MyDialog.Update(); // Update the field in the dialog.
    until MyNext = 4;
    Sleep(1000);
    MyDialog.Close();
end;
```

The dialog window opens and displays this text:

Counting to 4: 0

Every one second, the dialog window updates with the new value of *MyNext* until it reaches 4, then the dialog window closes.

Example 2

This example shows how to use the dialog.Open method to display a progress indicator in the Dynamics NAV Client connected to Business Central. The progress indicator will not display in the Business Central Web client.

```
var
    MyDialog: Dialog;
    MyNext: Integer;
    Text000: Label 'Progress from 0 to 9999 #1#####';
begin
    MyNext := 0;
    MyDialog.Open(Text000,MyNext);
    repeat
      // Do some processing.
      MyNext := MyNext + 1;
      MyDialog.Update(); // Update the field in the dialog.
    until MyNext = 9999;
    Sleep(1000);
    MyDialog.Close();
end;
```

The dialog window opens and displays the progress indicator and percentage.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dialog.Update Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Updates the value of a '#'-or '@' field in the active window.

Syntax

```
Dialog.Update([Number: Integer] [, Value: Any])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Dialog Type: Dialog An instance of the Dialog data type.

Number
Type: Integer
Each '#' or '@' field has a specific number. The Number argument tells into which field the Value should be inserted. If you omit this parameter, then all '#' or '@' fields in the active window are updated.

Value
Type: Any
This value or expression can be any simple AL data type such as Boolean, Option, Integer, Decimal, Date, Time, Text, and Code. If you omit this value, then the value from the variable in the Open method (Dialog)] call is used.

Remarks

Dialog windows that are opened by an object are closed when the object terminates.

Dialog windows are automatically sized to hold the longest line of text and the total number of lines.

We recommend the following guidelines:

- Enter messages as text constants.
- Write messages using active voice. For example, write "Processing items" instead of writing "Items are being processed."
- Align the # field to the left with at least one space character between the text and the variable.

> **NOTE**
> With the Dynamics NAV Client connected to Business Central, you can use @ characters instead of # characters for the *String* parameter to display the value as percentage and a progress indicator. The percentage value that is displayed is the percentage of the variable value from 0 to 9999. This is not supported in the Business Central Web client.

See Also

Dialog Data Type
Getting Started with AL
Developing Extensions

Dictionary Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an unordered collection of keys and values. The Dictionary data type is optimized for fast lookup of values.

The following methods are available on instances of the Dictionary data type.

METHOD NAME	DESCRIPTION
Add(TKey, TValue)	Adds the specified key and value to the dictionary.
ContainsKey(TKey)	Determines whether the Dictionary contains the specified key.
Count()	Gets the number of key/value pairs contained in the Dictionary.
Get(TKey, var TValue)	Gets the value associated with the specified key.
Get(TKey)	Gets the value associated with the specified key.
Keys()	Gets a collection containing the keys in the Dictionary.
Remove(TKey)	Removes the value with the specified key from the Dictionary.
Set(TKey, TValue)	Sets the value associated with the specified key.
Set(TKey, TValue, var TValue)	Sets the value associated with the specified key.
Values()	Gets a collection containing the values in the Dictionary.

Remarks

Each addition to the dictionary consists of a value, and its associated key. Every key in a Dictionary must be unique. A key cannot be null, but a value can be, only when the value type is a reference type.

The Dictionary data type does not support holding instantiated records. For this purpose, use temporary tables.

WARNING

Previously in C/AL, one would have typically used an in-memory temporary table to create a key-value data structure, as shown in the code below. In AL you use the Dictionary Data Type instead.

```
IF KeyCacheRec.Get('Some Value') THEN
    Complete data stack execution;
```

Example

In the following example, the variable `counter` represents the Dictionary data type to store a value representing the number of occurrences for each character in the `customerName`. Using the `Get` method, you get the number of occurrences for the character at position `i`. If `i` returns **false**, it means there is no value associated with that character, so you add the value 1. If `i` returns **true**, it means the value already exists, so you add `c + 1` to the value. The `Add` method adds the {key:value} pair to the Dictionary.

```
procedure CountCharactersInCustomerName(customerName: Text; var counter: Dictionary of [Char, Integer]);
var
    i : Integer;
    c : Integer;
begin

    for i := 1 to StrLen(customerName) do
    begin
        if counter.Get(customerName[i], c) then
            counter.Set(customerName[i], c + 1)
        else
            counter.Add(customerName[i], 1);
    end;
end;
```

See Also

Getting Started with AL
Developing Extensions
List Data Type

Dictionary.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified key and value to the dictionary.

Syntax

```
[Ok := ]  Dictionary.Add(Key: TKey, Value: TValue)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key of the element to add.

Value
Type: TValue
The value of the element to add.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.ContainsKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the Dictionary contains the specified key.

Syntax

```
Ok :=   Dictionary.ContainsKey(Key: TKey)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key to locate in the Dictionary.

Return Value

Ok Type: Boolean **true** if the Dictionary contains an element with the specified key, otherwise **false**.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of key/value pairs contained in the Dictionary.

Syntax

```
Count :=   Dictionary.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Return Value

Count Type: Integer The number of key/value pairs contained in the Dictionary.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the value associated with the specified key.

Syntax

```
[Ok := ]  Dictionary.Get(Key: TKey, var Value: TValue)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key of the value to get. If the specified key is not found an error will be reported.

Value
Type: TValue
The value associated with the specified key.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the value associated with the specified key.

Syntax

```
Value :=   Dictionary.Get(Key: TKey)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key of the value to get. If the specified key is not found an error will be reported.

Return Value

Value Type: TValue The value associated with the specified key. If the specified key is not found, an error will be raised.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Keys Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a collection containing the keys in the Dictionary.

Syntax

```
Keys :=   Dictionary.Keys()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Return Value

Keys Type: List of [TKey] A list containing the keys of the Dictionary.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the value with the specified key from the Dictionary.

Syntax

```
[Ok := ]  Dictionary.Remove(Key: TKey)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key of the element to remove.

Return Value

Ok Type: Boolean **true** if the element is successfully removed; otherwise, **false**. This method also returns **false** if the given key was not found in the original Dictionary.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the value associated with the specified key.

Syntax

```
Dictionary.Set(Key: TKey, Value: TValue)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key of the value to set.

Value
Type: TValue
The value that will be associated with the specified key.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the value associated with the specified key.

Syntax

```
[Replaced := ]  Dictionary.Set(Key: TKey, Value: TValue, var OldValue: TValue)
```

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Key
Type: TKey
The key of the value to set.

Value
Type: TValue
The value that will be associated with the specified key.

OldValue
Type: TValue
The value that was previously associated with the specified key.

Return Value

Replaced Type: Boolean **true** if the Dictionary contained a value associated with the given key that was replaced with the new value, otherwise **false**.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

Dictionary.Values Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a collection containing the values in the Dictionary.

Syntax

```
Values :=   Dictionary.Values()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Dictionary Type: Dictionary An instance of the Dictionary data type.

Return Value

Values Type: List of [TValue] A list containing the values of the Dictionary.

See Also

Dictionary Data Type
Getting Started with AL
Developing Extensions

DotNet Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Represents an unspecified .NET type.

See Also

Getting Started with AL
Developing Extensions

Duration Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Represents the difference between two DateTimes. This value can be negative. It is stored as a 64-bit integer. The integer value is the number of milliseconds during the duration.

The following are examples of durations:

DateTime-DateTime=Duration

DateTime-Duration=DateTime

DateTime+Duration=DateTime

Example

This example shows how to calculate the difference between two DateTimes. This example is run on a computer with the Current Format in the Regional and Language Options set to English (United States).

```
var
    DateTime1: DateTime;
    DateTime2: DateTime;
    Duration: Duration;
begin
    DateTime1 := CreateDateTime(20090101D, 080000T); // January 1, 2009 at 08:00:00 AM
    DateTime2 := CreateDateTime(20090505D, 133001T); // May 5, 2009 at 1:30:01 PM
    Duration := DateTime2 - DateTime1;
    Message(Format(Duration));
end;
```

The message window displays the following:

124 days 4 hours 30 minutes 1 second

See Also

Getting Started with AL
Developing Extensions

Enum Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 4.0.*

Represents the text content of an element or attribute.

The following methods are available on the Enum data type.

METHOD NAME	DESCRIPTION
FromInteger(Integer)	Returns an enum with the integer value
Names()	Gets the value names
Ordinals()	Gets the ordinal numbers/ID's for the values

The following methods are available on instances of the Enum data type.

METHOD NAME	DESCRIPTION
AsInteger()	Get the enum value as an integer value.
Names()	Gets the value names
Ordinals()	Gets the ordinal numbers/ID's for the values

Example

For more information, see Extensible Enums.

See Also

Getting Started with AL
Developing Extensions

Enum.FromInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Returns an enum with the integer value

Syntax

```
Enum with integer value :=   Enum.FromInteger(Value: Integer)
```

Parameters

Value

Type: Integer

Return Value

Enum with integer value Type: Any

Example

```
enum 50130 YesNo
{
    value(0; Yes) { }
    value(10; No) { }
}

codeunit 50130 YesNoTest
{
    procedure Test();
    var
        Answer: enum YesNo;
    begin
        Answer := YesNo.FromInteger(10); // Ordinal value for 'No'
        if Answer = YesNo::No then
            Message('Success');
    end;
}
```

See Also

Enum Data Type
Getting Started with AL
Developing Extensions

Enum.Names Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the value names

Syntax

```
List of enum value names :=   Enum.Names()
```

Return Value

List of enum value names Type: List of [Text]

See Also

Enum Data Type
Getting Started with AL
Developing Extensions

Enum.Ordinals Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the ordinal numbers/ID's for the values

Syntax

```
List of ordinals :=   Enum.Ordinals()
```

Return Value

List of ordinals Type: List of [Integer]

See Also

Enum Data Type
Getting Started with AL
Developing Extensions

Enum.AsInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Get the enum value as an integer value.

Syntax

```
The Enum Value as an Integer value :=   Enum.AsInteger()
```

Parameters

Enum Type: Enum An instance of the Enum data type.

Return Value

The Enum Value as an Integer value Type: Integer

See Also

Enum Data Type
Getting Started with AL
Developing Extensions

Enum.Names Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the value names

Syntax

```
List of enum value names :=   Enum.Names()
```

Return Value

List of enum value names Type: List of [Text]

See Also

Enum Data Type
Getting Started with AL
Developing Extensions

Enum.Ordinals Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the ordinal numbers/ID's for the values

Syntax

```
List of ordinals :=   Enum.Ordinals()
```

Return Value

List of ordinals Type: List of [Integer]

See Also

Enum Data Type
Getting Started with AL
Developing Extensions

ErrorInfo Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Provides a structure for grouping information about an error.

> **NOTE**
> This data type is supported only in Business Central on-premises.

The following methods are available on instances of the ErrorInfo data type.

METHOD NAME	DESCRIPTION
DataClassification([DataClassification])	Specifies the severity level of the error. Values include 'Critical', 'Error', 'Warning', 'Normal', and 'Verbose'
ErrorType([ErrorType])	Specifies type of the error. 'Client' shows the specified message in the client and sends it to telemetry. 'Internal' shows a generic message in the client and sends the specified message to telemetry.
Message([String])	Specifies the message that will be sent to telemetry. For a 'Client' error type, the message will also be appear in the client.
Verbosity([Verbosity])	Specifies the severity level of the error. This can determine whether the error should be sent to telemetry (which is based on the trace level setting of the server).

See Also

Getting Started with AL
Developing Extensions

ErrorInfo.DataClassification Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Specifies the severity level of the error. Values include 'Critical', 'Error', 'Warning', 'Normal', and 'Verbose'

Syntax

```
[DataClassification := ]  ErrorInfo.DataClassification([DataClassification: DataClassification])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ErrorInfo Type: ErrorInfo An instance of the ErrorInfo data type.

DataClassification
Type: DataClassification
The data classification of the content in the message.

Return Value

DataClassification Type: DataClassification The current data classification of the ErrorInfo.

See Also

ErrorInfo Data Type
Getting Started with AL
Developing Extensions

ErrorInfo.ErrorType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Specifies type of the error. 'Client' shows the specified message in the client and sends it to telemetry. 'Internal' shows a generic message in the client and sends the specified message to telemetry.

Syntax

```
[ErrorType := ]  ErrorInfo.ErrorType([ErrorType: ErrorType])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ErrorInfo Type: ErrorInfo An instance of the ErrorInfo data type.

ErrorType
Type: ErrorType
The error type of the error.

Return Value

ErrorType Type: ErrorType The current error type of the ErrorInfo.

See Also

ErrorInfo Data Type
Getting Started with AL
Developing Extensions

ErrorInfo.Message Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Specifies the message that will be sent to telemetry. For a 'Client' error type, the message will also be appear in the client.

Syntax

```
[Message := ]  ErrorInfo.Message([Message: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

ErrorInfo Type: ErrorInfo An instance of the ErrorInfo data type.

Message

Type: String

The message of the the ErrorInfo

Return Value

Message Type: String The current message of the ErrorInfo.

See Also

ErrorInfo Data Type
Getting Started with AL
Developing Extensions

ErrorInfo.Verbosity Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Specifies the severity level of the error. This can determine whether the error should be sent to telemetry (which is based on the trace level setting of the server).

Syntax

```
[Verbosity := ]  ErrorInfo.Verbosity([Verbosity: Verbosity])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ErrorInfo Type: ErrorInfo An instance of the ErrorInfo data type.

Verbosity
Type: Verbosity
The verbosity that the error should be sent with.

Return Value

Verbosity Type: Verbosity The current verbosity of the ErrorInfo.

See Also

ErrorInfo Data Type
Getting Started with AL
Developing Extensions

FieldRef Data Type

2/17/2021 • 4 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Identifies a field in a table and gives you access to this field.

The following methods are available on instances of the FieldRef data type.

METHOD NAME	DESCRIPTION
Active()	Checks whether the field that is currently selected is enabled.
CalcField()	Updates FlowFields in a record.
CalcSum()	Calculates the total of all values of a SumIndexField in a table.
Caption()	Gets the current caption of a field as a String.
Class()	Gets the value of the FieldClass Property of the field that is currently selected. This method returns an error if no field is selected.
EnumValueCount()	Gets the number of Enum values (or Option members) from the Enum metadata for the field that is currently selected.
FieldError([String])	Stops the execution of the code, causing a run-time error, and creates an error message for a field.
GetEnumValueCaption(Integer)	Gets an Enum value (or Option member) caption for the from the Enum metadata for the field that is currently selected.
GetEnumValueCaptionFromOrdinalValue(Integer)	Gets an Enum value (or Option member) caption for the from the Enum metadata for the field that is currently selected.
GetEnumValueName(Integer)	Gets an Enum value (or Option member) name from the Enum metadata for the field that is currently selected.
GetEnumValueNameFromOrdinalValue(Integer)	Gets an Enum value (or Option member) name from the Enum metadata for the field that is currently selected.
GetEnumValueOrdinal(Integer)	Gets the Enum value (or Option member) ordinal value from the Enum metadata for the field that is currently selected.
GetFilter()	Gets the filter that is currently applied to the field referred to by FieldRef.
GetRangeMax()	Gets the maximum value in a range for a field.
GetRangeMin()	Gets the minimum value in a range for a field.

METHOD NAME	DESCRIPTION
Length()	Gets the maximum size of the field (the size specified in the DataLength property of the field). This method is usually used for finding the defined length of code and text fields.
Name()	Gets the name of a field as a string.
Number()	Gets the number of a field as an integer.
OptionCaption()	Gets the option caption of the field that is currently selected.
OptionMembers()	Gets the list of options that are available in the field that is currently selected.
OptionString()	The 'OptionString' property has been deprecated and will be removed in the future. Use the 'OptionMembers' property instead.
Record()	Gets the RecordRef of the field that is currently selected. This method returns an error if no field is selected.
Relation()	Finds the table relationship of a given field.
SetFilter(String [, Any,...])	Assigns a filter to a field that you specify.
SetRange([Any] [, Any])	Sets a simple filter on a field, such as a single range or a single value.
TestField()	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Byte)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Boolean)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Char)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Option)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Integer)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(BigInteger)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Decimal)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

METHOD NAME	DESCRIPTION
TestField(Guid)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(String)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Label)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Text)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Code)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Date)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(DateTime)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Time)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Variant)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Enum)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
TestField(Any)	Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.
Type()	Gets the data type of the field that is currently selected.
Validate([Any])	Use this method to enter a new value into a field and have the new value validated by the properties and code that have been defined for that field.
Value([Any])	Sets or gets the value of the field that is currently selected. This method returns an error if no field is selected.

See Also

Getting Started with AL
Developing Extensions

FieldRef.Active Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether the field that is currently selected is enabled.

Syntax

```
Ok :=   FieldRef.Active()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Ok Type: Boolean

Remarks

Each field in a record can be set as enabled or disabled in the table description. You cannot use a disabled field because disabled fields cannot contain data.

This method is like the FieldActive Method (Record).

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named Recref. The Field Method (RecordRef) uses Recref to create a FieldRef variable that is named MyFieldRef. MyFieldRef sets a reference to the first field (field 1) in the table. The SetRange Method (FieldRef) sets a filter that selects record 30000. The Field Method (RecordRef) selects the record and then loops through fields 1 through 6. For each field, the Active method determines whether the field is enabled. If the field is enabled, a message that states that the field is enabled is displayed. Otherwise, a message that states that the field is not enabled is displayed.

NOTE

You can use the name of the table instead of the table number to open the table by using the following syntax:

```
Recref.Open\(DATABASE::Customer\)
```

```
var
    Recref: RecordRef;
    MyFieldRef: FieldRef;
    i: Integer;
    Text000: Label 'Field %1 is enabled.';
    Text001: Label 'Field %1 is not enabled.';
begin
    Recref.Open(18);
    MyFieldRef := Recref.Field(1);
    MyFieldRef.SetRange('30000');
    Recref.Find('-');
    for i := 1 to 5 do begin
      MyFieldRef := Recref.FieldIndex(i);
      if MyFieldRef.Active then
        Message(Text000, i)
      else begin
        Message(Text001, i)
      end;
    end;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.CalcField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Updates FlowFields in a record.

Syntax

```
[Ok := ]  FieldRef.CalcField()
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

FlowFields are virtual fields. The values in these fields are not saved in the table. This means that you must use the CalcFields method to update them. For example, if you retrieve a record using the Find Method (RecordRef) and Next Method (RecordRef) methods, the FlowFields in those records are set to zero (0). Then, you can call `FieldRef.CalcField`, to calculate the value in one of the FlowFields.

When a FlowField is a direct source expression of a control on a page or a report, the calculation is automatically performed. You can also use the CALCFieldS method to calculate binary large objects (BLOBs). For more information, see BLOB Data Type.

This method is similar to the CalcFields Method (Record) method.

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustRecordref. The Find Method (RecordRef) selects the first record in the table and then loops through all the records until no records could be found. For each record, the Field Method (RecordRef) creates a FieldRef variable that is named MyFieldref for the Balance Due field (field 66), which is a flow field. The CalcField method is called to update the field before the customer ID and the balance due are displayed. Otherwise, the balance due for every record will be set to 0.

```
var
    CustRecordref: RecordRef;
    MyFieldRef: FieldRef;
    Count: Integer;
    Text000: Label '%1: \\Balance Due: %2.';
begin
    Count := 0;
    CustRecordref.Open(18);
    if CustRecordref.Find('-') then
      repeat
        MyFieldRef := CustRecordref.Field(66);
        MyFieldRef.CalcField;
        Message(Text000, CustRecordref.RecordId, MyFieldRef);
        Count := Count + 1;
      until CustRecordref.Next = 0;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

FieldRef.CalcSum Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates the total of all values of a SumIndexField in a table.

Syntax

```
[Ok := ]  FieldRef.CalcSum()
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method is like the CalcSums Method (Record) method. This method operates only on records that meet the conditions of any filters associated with the record.

If possible, the CalcSum method uses SumIndexField Technology (SIFT). SIFT is used only if the following conditions are true:

- The Dynamics 365 key contains the fields that are used in the filters that are defined for the FlowField.
- The SumIndexFields on the Dynamics 365 key contain the field to which the FieldRef parameter refers.
- The MaintainSIFTIndex Property is set to **true**.

> **NOTE**
> By default this property is set to **true** for all keys.

For Dynamics 365 Business Central, CalcSum execution is decoupled from Dynamics 365 SIFT index definitions. This means that if any of the conditions for using SIFT indexes are not true, then Dynamics 365 traverses all records in the base table to perform the calculation instead of using SIFT. This can reduce the number of required SIFT indexes, which can improve performance. In earlier versions of Dynamics 365, if the conditions for using SIFT indexes were not true and the **MaintainSIFTIndex** property was enabled, then you received an error when you called the CalcSum method. This provided a degree of protection in earlier versions against accidentally requesting a sorting for which no index existed. In Dynamics 365 Business Central, an index is not required to support a certain sorting, but sorting without an index could lead to bad performance if a search returns a large result set, which would then have to be sorted before the first row is returned.

Example

This example sets a RecordRef variable to refer to table 21, the **Cust. Ledger Entry** table. Next, it creates a reference to field 18, the **Sales (LCY)** field, in the **Cust. Ledger Entry** table and assigns the field reference to a FieldRef variable. The **Sales (LCY)** field is a decimal field and is one of the SumIndexFields on a Dynamics 365 key in the **Cust. Ledger Entry** table. The code displays the original Value of the FieldRef variable, then calls the CalcSum method and displays the calculated value of the field.

```
var
    MyFieldRef: FieldRef;
    MyRecRef: RecordRef;
begin
    MyRecRef.Open(21);
    MyFieldRef := MyRecRef.Field(18);
    Message('Before CalcSum, Sales (LCY) is %1.', MyFieldRef.Value);
    MyFieldRef.CalcSum;
    Message('After CalcSum, Sales (LCY) is %1.', MyFieldRef.Value);
end;
```

On a computer that has the regional format set to English (United States), the first message window displays the following:

Before CalcSum, Sales (LCY) is 0.

The second message window displays the following:

After CalcSum, Sales (LCY) is 55,162.67.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions
CalcSum Method (Record)
AL Database Methods and Performance on SQL Server

FieldRef.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current caption of a field as a String.

Syntax

```
Caption :=   FieldRef.Caption()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Caption Type: String

Remarks

The Caption method returns the caption of a field. Caption first looks for a CaptionML Property. If it does not find one, it will use the Name Property. This means that Caption is enabled for multilanguage functionality.

This method is similar to the FieldCaption Method (Record).

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The code uses the Field Method (RecordRef) to loop through field 1 through 9 and creates a FieldRef variable that is named MyFieldRef. For each field, the Caption method retrieves the caption of the field, stores it in the varCaption variable and displays it in a message box.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    varCaption: Text;
    i: Integer;
    Text000: Label 'The caption for field %1 is "%2"';
begin
    CustomerRecref.Open(18);
    for i := 1 to 9 do begin
      MyFieldRef := CustomerRecRef.Field(i);
      varCaption := MyFieldRef.Caption;
      Message(Text000, i, varCaption);
    end;
    CustomerRecref.Close;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Class Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the value of the FieldClass Property of the field that is currently selected. This method returns an error if no field is selected.

Syntax

```
Class :=   FieldRef.Class()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Class Type: FieldClass

Remarks

The FieldRef refers to the field that you are interested in.

The Class method returns the class as an Option. However, you cannot assign the class to an Option variable directly. Instead, you must use the Evaluate Method. The Evaluate method has a variable parameter to which the value is assigned and a string parameter. You use the forMAT method to convert the result of the FieldRef.Class method to Text, and then use the Evaluate method to convert the Text to an Option.

Example 1

In this example, the return value of the Class method is converted to Text and then converted to an Option. The value of the **OptionString** property of OptionVar is Normal,FlowFilter,FlowField.

```
var
    OptionVar: Option;
    FldRef: FieldRef;
begin
    Evaluate(OptionVar,Format(FldRef.Class));
end;
```

Example 2

In this example, the return value of the Class method is converted to Text and then converted to an Option. This

example uses the Field virtual table instead of an Option variable. The Field virtual table has a Class field, which is an Option and already has the correct OptionString.

```
var
    FieldRec: Record Field;
    FldRef: FieldRef;
begin
    Evaluate(FieldRec.Class,Format(FldRef.Class));
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.EnumValueCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the number of Enum values (or Option members) from the Enum metadata for the field that is currently selected.

Syntax

```
Number of Enum values :=   FieldRef.EnumValueCount()
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Number of Enum values Type: Integer The number of Enum values.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.FieldError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stops the execution of the code, causing a run-time error, and creates an error message for a field.

Syntax

```
FieldRef.FieldError([Text: String])
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Text

Type: String

Use this optional parameter to include the text of the error message. If this parameter is not present, default text will be used.

Remarks

Similar to a run-time error, this method causes any transaction to be ended automatically.

This method is like the FieldError Method for the Record data type. For examples, see FieldError Method (Record).

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The CustomerName variable is initialized with a blank text. `CustomerRecref.Field` creates a FieldRef that is named MyFieldRef for field1 (No.) and selects record 30000. Field 2 (Name) is then selected for record 30000. If the CustomerName variable is a blank text, then `MyFieldRef.FieldError` is executed and an error message is displayed. The text in Text000 text constant is inserted into the error message that is displayed by Dynamics 365.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    CustomerName: Text;
    Text000: Label 'cannot be blank';
begin
    CustomerRecref.Open(18);
    CustomerName := '';
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value('30000');
    MyFieldRef := CustomerRecref.Field(2);
    if CustomerName = '' then
      MyFieldRef.FieldError(Text000)
    else
      //Do some processing
end;
```

This code example displays the following error message:

Name cannot be blank in Customer No.="30000".

Programming Guidelines

We recommend the following guidelines for error messages:

- Describe what is wrong and how to solve the problem.
- Write a short descriptive message. Do not use more words than necessary.
- Note that a period is automatically inserted at the end of a FieldError.
- Use a text constant for the *Text* parameter.

For more information, see Progress Windows, Message, Error, and Confirm Methods.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetEnumValueCaption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 7.0.*

Gets an Enum value (or Option member) caption for the from the Enum metadata for the field that is currently selected.

Syntax

```
The Enum value caption :=   FieldRef.GetEnumValueCaption(Index: Integer)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Index

Type: Integer

The index in the list of Enum values to get the Enum value (or Option member) caption for. The index is 1-based.

Return Value

The Enum value caption Type: String The Enum value caption.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetEnumValueCaptionFromOrdinalValue Method

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 7.0.*

Gets an Enum value (or Option member) caption for the from the Enum metadata for the field that is currently selected.

Syntax

```
The Enum value caption :=   FieldRef.GetEnumValueCaptionFromOrdinalValue(Ordinal: Integer)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Ordinal
Type: Integer
The Enum value's ordinal value to get the Enum value (or Option member) caption for.

Return Value

The Enum value caption Type: String The Enum value caption.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetEnumValueName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets an Enum value (or Option member) name from the Enum metadata for the field that is currently selected.

Syntax

```
The Enum value name :=   FieldRef.GetEnumValueName(Index: Integer)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Index
Type: Integer
The index in the list of Enum values to get the Enum value (or Option member) name for. The index is 1-based.

Return Value

The Enum value name Type: String The Enum value name.

Example

```
procedure GetOptionNo(Value: Text; FieldRef: FieldRef): Integer
    var
        FieldRefValueVar: Variant;
        FieldRefValueInt: Integer;
    begin
        if (Value = '') and (FieldRef.GetEnumValueName(1) = ' ') then
            exit(0);

        FieldRefValueVar := FieldRef.Value();
        FieldRefValueInt := -1;
        if Evaluate(FieldRef, Value) then begin
            FieldRefValueInt := FieldRef.Value();
            FieldRef.Value(FieldRefValueVar);
        end;

        exit(FieldRefValueInt);
    end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetEnumValueNameFromOrdinalValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets an Enum value (or Option member) name from the Enum metadata for the field that is currently selected.

Syntax

```
The Enum value name or empty if the ordinal value doesn't exist :=
FieldRef.GetEnumValueNameFromOrdinalValue(Ordinal: Integer)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Ordinal

Type: Integer

The Enum value's ordinal value to get the Enum value (or Option member) name for.

Return Value

The Enum value name or empty if the ordinal value doesn't exist Type: String The Enum value name.

Example

```
procedure OptionNoExists(var FieldRef: FieldRef; OptionValue: Text): Boolean
    var
        OptionNo: Integer;
    begin
        if Evaluate(OptionNo, OptionValue) then
            exit((FieldRef.GetEnumValueNameFromOrdinalValue(OptionNo) <> '') or
((FieldRef.GetEnumValueNameFromOrdinalValue(OptionNo) = '') and (OptionNo = 0)));

        exit(false);
    end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetEnumValueOrdinal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the Enum value (or Option member) ordinal value from the Enum metadata for the field that is currently selected.

Syntax

```
The Enum value ordinal value :=   FieldRef.GetEnumValueOrdinal(Index: Integer)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Index

Type: Integer

The index in the list of Enum ordinal values to get the Enum value (or Option member) ordinal value for. The index is 1-based.

Return Value

The Enum value ordinal value Type: Integer The ordinal value.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the filter that is currently applied to the field referred to by FieldRef.

Syntax

```
String :=   FieldRef.GetFilter()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

String Type: String

Remarks

See also GetFilters Method (RecordRef), SetFilter Method (FieldRef), and SetRange Method (FieldRef).

This method is like the GetFilter Method (Record).

Example

The following example opens the Customer table as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef for the first field (No.) and stores the reference in the MyFieldRef variable. The GetFilter method retrieves the filters that are set on the No. field and stores the value in the Filters1 variable. The value of any filter that is set is displayed in a message box. The SetFilter Method (FieldRef) sets the filter that selects records from 10000 to 40000 in the No. field. The GetFilter method retrieves and stores the filter in the Filter2 variable and displays it in a message. The value in the Filter1 variable is blank because no filter is set. The value in Filter2 is 10000..40000 because of the filter that is set by the SetFilter Method (FieldRef).

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    Filters1: Text;
    Filters2: Text;
    Text000: Label 'Filter before filters set: %1.';
    Text001: Label 'Filter after filters set: %1.';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    Filters1 := MyFieldRef.GetFilter;
    Message(Text000, Filters1);
    MyFieldRef.SetFilter('10000..40000');
    Filters2 := MyFieldRef.GetFilter;
    Message(Text001, Filters2);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetRangeMax Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the maximum value in a range for a field.

Syntax

```
Value :=   FieldRef.GetRangeMax()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Value Type: Any

Remarks

This method is like the GetRangeMax Method (Record) method.

Example

The following example opens the Customer table as RecordRef variable, creates a FieldRef for the first field (No.) and stores the reference in the MyFieldRef variable. The SetFilter Method (FieldRef) sets a filter that selects records in the range 10000 to 40000 from the No. field. The GetRangeMax method retrieves and stores the maximum value that was set in the filter, stores the value in the varMax variable and displays it in a message box. The varMax variable contains 40000 which is the maximum value that is set in the filter.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    varMax: Text;
    Text000: Label 'The maximum value in the filter is %1.';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetFilter('10000..40000');
    varMax := MyFieldRef.GetRangeMax();
    Message(Text000, varMax);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.GetRangeMin Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the minimum value in a range for a field.

Syntax

```
Value :=   FieldRef.GetRangeMin()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Value Type: Any

Remarks

This method is like the GetRangeMin Method (Record) method.

Example

The following example opens the Customer table as RecordRef variable, creates a FieldRef for the first field (No.) and stores the reference in the MyFieldRef variable. The SetFilter Method (FieldRef) sets a filter that selects records in the range 10000 to 40000 from the No. field. The GetRangeMin method retrieves and stores the minimum value that is set in the filter, stores the value in the varMin variable and displays it in a message box. The varMin variable contains 10000 which is the minimum value that is set in the filter.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    varMin: Text;
    Text000: Label 'The minimum value in the filter is %1.';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetFilter('10000..40000');
    varMin := MyFieldRef.GetRangeMin();
    Message(Text000, varMin);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Length Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the maximum size of the field (the size specified in the DataLength property of the field). This method is usually used for finding the defined length of code and text fields.

Syntax

```
Length :=   FieldRef.Length()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Length Type: Integer

Remarks

For Text and Code fields this method returns the defined length. For other types of fields, it returns the fixed byte size, for example, Integer fields returns 4.

Example

The following example opens the Customer table as a RecordRef variable. The Field Method (RecordRef) creates a FieldRef for any specified field and stores the reference in the MyFieldRef variable. The LENGTH method retrieves the maximum size of the field and stores the value in the varLength variable. The value that is stored in the varLength is displayed in a message box.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    varLength: Integer;
    varFieldNo: Integer;
    Text000: Label 'The maximum size of the field is %1.';
begin
    varFieldNo := 1;
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(varFieldNo);
    varLength := MyFieldRef.Length();
    Message(Text000, varLength);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of a field as a string.

Syntax

```
Name :=   FieldRef.Name()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Name Type: String

Remarks

The Caption Method (FieldRef, TestPage Field) method retrieves the Caption Property of a field. To enable your application for multilanguage functionality, you must use the FieldCaption Method (Record) instead.

This method is similar to the FieldName Method (Record).

Example

The following example opens the Customer table as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a reference to the fields in the table and stores the FieldRef in the MyFieldRef variable. The code loops through field 1 through 5. For each field, the Name method retrieves the name of the field and stores the value in the varName variable. The field number and the value in the varName variable are displayed in a message box.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    varName: Text;
    i: Integer;
    Text000: Label 'The name of field %1 is "%2".\\';
begin
    for i := 1 to 5 do begin
        CustomerRecref.Open(Database::Customer);
        MyFieldRef := CustomerRecref.Field(i);
        varName := MyFieldRef.Name;
        Message(Text000, i, varName);
        CustomerRecref.Close;
    end;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Number Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of a field as an integer.

Syntax

```
No :=   FieldRef.Number()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

No Type: Integer

Remarks

This method is like the FieldNo Method (Record) method.

Example

The following example displays the caption and the field number of the first 10 fields in the Location table. The Location table is open as a RecordRef Data Type object and the reference is stored in the LocationRecref variable. The FieldIndex variable that stores the field index is initialized to 0. The LocationRecref variable uses the FieldIndex Method (RecordRef) to create a FieldRef that is named MyFieldRef for the specified field index. MyFiledRef now references the field that is specified by the FieldIndex. MyFieldref is then used to display the number and caption of the field The Number Method (FieldRef) method retrieves the field number. This is repeated for the first ten fields in the table.

```
var
    MyFieldRef: FieldRef;
    LocationRecref: RecordRef;
    FieldIndex: Integer;
begin
    LocationRecref.Open(DATABASE::Location);
    FieldIndex := 0;
    repeat
      FieldIndex  := FieldIndex + 1;
      MyFieldRef := LocationRecref.FieldIndex(FieldIndex);
      Message('Field Number: %1  Field Caption: %2.' , MyFieldRef.Number, MyFieldRef.Caption);
    until FieldIndex = 10;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.OptionCaption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the option caption of the field that is currently selected.

Syntax

```
OptionCaption :=   FieldRef.OptionCaption()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

OptionCaption Type: String

Remarks

The option caption of the field is returned as a comma-separated string.

If the field is not an Option, an empty string is returned.

This method returns an error if no field is selected.

Example

The following example opens the Item table as a RecordRef variable that is named ItemRecref. and creates a reference to field 19 (Price/Profit Calculation field), which is an Option field. The OptionCaption method retrieves the caption of the option field and displays the options as a comma-separated list.

```
var
    MyFieldRef: FieldRef;
    ItemRecref: RecordRef;
    OptionCaption: Text;
begin
    ItemRecref.Open(Database::Item);
    MyFieldRef := ItemRecref.Field(19);
    OptionCaption := MyFieldRef.OptionCaption;
    Message('%1', OptionCaption);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.OptionMembers Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the list of options that are available in the field that is currently selected.

Syntax

```
OptionMembers :=   FieldRef.OptionMembers()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

OptionMembers Type: String

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.OptionString Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

The 'OptionString' property has been deprecated and will be removed in the future. Use the 'OptionMembers' property instead.

Syntax

```
OptionMembers :=   FieldRef.OptionString()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

OptionMembers Type: String

Remarks

All the options for this field are returned as a comma separated-string.

This method returns an error if no field is selected.

If the field is not an option an empty string is returned.

Example

The following example opens the Item table with RecordRef variable that is named ItemRecref and creates a reference to field 19 (Price/Profit Calculation), which is an Options field. The OptionString method retrieves the options in the field and displays them as a comma-separated list.

```
var
    MyFieldRef: FieldRef;
    ItemRecref: RecordRef;
    OptionString: Text;
begin
    ItemRecref.Open(Database::Item);
    MyFieldRef := ItemRecref.Field(19);
    OptionString := MyFieldRef.OptionString;
    Message(' %1', OptionString);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Record Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the RecordRef of the field that is currently selected. This method returns an error if no field is selected.

Syntax

```
Record :=   FieldRef.Record()
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Record Type: RecordRef

Example

The following example opens the Item table as a RecordRef variable that is named ItemRecref, creates a reference to field 1 (No.), and stores the value in the variable named MyFieldRef. The Record method uses the MyFieldRef variable to return the RecordRef of field 1 and stores the reference in a variable named MyRecRef. `MyRecRef.Number` returns the table that the selected field belongs to.

```
var
    MyFieldRef: FieldRef;
    ItemRecref: RecordRef;
    MyRecRef: RecordRef;
    Text000: Label 'The selected field is from table %1.';
begin
    ItemRecref.Open(Database::Item);
    MyFieldRef := ItemRecref.Field(1);
    MyRecRef := MyFieldRef.Record;
    Message(Text000, MyRecRef.Number);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Relation Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the table relationship of a given field.

Syntax

```
TableNumber :=   FieldRef.Relation()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

TableNumber Type: Integer

Remarks

You can use this method for several purposes such as to determine lookups or to check to see if you have permission to read from a table.

This method is similar to the Relation Method (Record).

Example

The following example opens table 37, the Sales Line table, as a RecordRef variable and creates a reference to field 2 (Sell-to Customer No.). The FieldRef Data Type of field 2 is stored in the MyFieldRef variable. The RELATION method retrieves the number of the table that has a relation with the Sell-To-Customer field (field 2). The table number is stored the varRelation variable and displayed in the message box.

```
var
    MyFieldRef: FieldRef;
    SaleRecref: RecordRef;
    varRelation: Integer;
    Text000: Label 'Field 2 in the Sales Line (37) table has a relation with table %1.';
begin
    SaleRecref.Open(37);
    MyFieldRef := SaleRecref.Field(2);
    varRelation := MyFieldRef.Relation;
    Message(Text000, varRelation);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.SetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Assigns a filter to a field that you specify.

Syntax

```
FieldRef.SetFilter(String: String [, Value: Any,...])
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

String
Type: String
The filter expression. A valid expression consists of alphanumeric characters and one or more of the following operators: <, >, , &, |, and =. You can use replacement fields (%1, %2, and so on) to insert values at run time.

Value
Type: Any
Replacement values to insert in replacement fields in the filter expression. The data type of Value must match the type of FieldRef.

Remarks

If the method is called with a field for which a filter already exists, the filter is removed before a new one is set. You can construct filters using the following operators.

OPERATOR	DESCRIPTION
..	Range
&	And
|	Or
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<>	Different from
*	Forms a part of value

OPERATOR	DESCRIPTION
@	Case-insensitive

This method is like the SetFilter Method (Record) method.

Example

The following example opens the Customer table as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef for the first field (No.) and stores the reference in the MyFieldRef variable. The GetFilter Method (FieldRef) retrieves the filters that are set on the No. field and stores the value in the Filter1 variable. The value of any filter that is set is displayed in a message box. The SetFilter method sets a filter that selects records from 10000 to 40000 in the No. field. The GetFilter Method (FieldRef) retrieves and stores the new filter in the Filter2 variable and displays it in a message. The value in the Filter1 variable is blank because no filter is set. The value in Filter2 is 10000..40000 because of the filter that is set by the SetFilter method.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    Filter1: Text;
    Filter3: Text;
    Text000: Label 'Filter before filters set: %1.';
    Text001: Label 'Filter after filters set: %1.';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    Filter1 := MyFieldRef.GetFilter;
    Message(Text000, Filter1);
    MyFieldRef.SetFilter('10000..40000');
    Filter2 := MyFieldRef.GetFilter;
    Message(Text001, Filter2);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.SetRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a simple filter on a field, such as a single range or a single value.

Syntax

```
FieldRef.SetRange([FromValue: Any] [, ToValue: Any])
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

FromValue
Type: Any
The lower limit of the range. The data type of FromValue must match the data type of the field referred to by FieldRef.

ToValue
Type: Any
The upper limit of the range. If you omit this parameter, the FromValue you specified is used. The data type of ToValue must match the data type of the field referred to by FieldRef.

Remarks

The SetRange method provides a quick way to set a simple filter on a field. If you call this method by using a field that already has a filter, that filter is removed before the new filter is set.

If you omit all of the optional parameters, all filters set for that field are removed. The SetRange method fails if no field is selected.

This method is like the SetRange Method (Record) method.

Example

The following example opens the Customer table as a RecordRef object, creates a reference to the first (No.) field, and stores the reference in the MyFieldRef variable. The SetRange method sets a filter that selects all records from 10000 to 40000 in the No. field. The Find Method (RecordRef) searches and selects the first record in the filter and counts the number of records that are found. The number of records is stored in the Count variable. The process is repeated by looping through all the records in the filter until no more records are found. The number of records that are found in the range is stored in the Count variable and displayed in a message box.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    Count: Integer;
    Text000: Label '%1 records were retrieved.';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetRange('10000' , '40000');
    Count := 0;
    if CustomerRecref.Find('-') then
      repeat
        Count := Count + 1;
      until CustomerRecref.Next = 0;
    Message(Text000 , Count);
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField()
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Remarks

This method is like the TestField Method (Record) method.

Example 1

The following example opens the Customer table as a RecordRef variable that is named CustomerRecref, created a reference to the first field (No) and stores the reference in the MyFieldRef variable. The Value Method (FieldRef, TestPage Field) sets the No. field to a blank text. The TestField method determines whether the contents of the field match 10000, the specified value. In this case, the content does not match so the Dynamics 365 throws an exception. If the there is a match, no exception is thrown.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value := '';
    MyFieldRef.TestField('10000');
end;
```

In this example, the Dynamics 365 displays following error message:

No. must be equal to 10000 in Customer: No.=. Current value is ''.

Example 2

If the value of the No. field is set to a value other than 10000, Dynamics 365 displays the following error message:

No. must be equal to 10000 in Customer: No.=AAA10000. Current value is 'AAA 10000'.

```
CustomerRecref.Open(DataBase::Customer);
MyFieldRef := CustomerRecref.Field(1);
MyFieldRef.Value := 'AAA 10000';
MyFieldRef.TestField('10000');
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Byte)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Byte

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Boolean)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Boolean

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Char)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Char

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Option)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Option

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Integer)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Integer

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: BigInteger)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: BigInteger

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Decimal)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Decimal

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Guid)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Guid

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: String)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: String

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Label)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Label

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Text)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Text

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Code)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Code

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Date)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Date

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: DateTime)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: DateTime

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Time)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Time

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Variant)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Variant

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Enum)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Enum

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the contents of a field matches a given value. If the contents differ from the given value, an error message is displayed.

Syntax

```
FieldRef.TestField(Value: Any)
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Value

Type: Any

The value that you want to compare with the contents of the field referred to by FieldRef. The data type of Value must match the type of the field. If you include Value and the contents of the field do not match, an error message is displayed. If you omit Value and the content of the field is zero or blank (empty string), an error message is displayed.

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Type Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the data type of the field that is currently selected.

Syntax

```
Type :=   FieldRef.Type()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

Return Value

Type Type: FieldType

Example

The following example opens the Customer table as a RecordRef variable that is named CustomerRecref. The code loops through fields 1 to 5 and creates a FieldRef that is named MyFieldRef for each field that is selected. `MyFieldRef.Type` retrieves the data of each field and displays it in a message box.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    varType: Variant;
    Text000: Label 'Field %1 is a %2 data type.';
begin
    CustomerRecref.Open(Database::Customer);
    for i := 1 to 5 do begin
      MyFieldRef := CustomerRecref.Field(i);
      Message(Text000, i, MyFieldRef.Type);
    end;
END;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Validate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Use this method to enter a new value into a field and have the new value validated by the properties and code that have been defined for that field.

Syntax

```
FieldRef.Validate([NewValue: Any])
```

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

NewValue
Type: Any
The value to insert in the field.

Remarks

The Validate method first checks any TableRelation Property, and then executes the OnValidate (Fields) Trigger of the field.

If you omit *NewValue*, the method validates the current value.

This method is like the Validate Method (Record).

Example

The following example opens table 17 (G/L Entry) as a RecordRef that is named EntryRecref. The FindFirst Method (RecordRef) searches for the first record in the table. The Field Method (RecordRef) sets the field to 3, which is the G/L Account No. field. The **VALIDATE** method validates and inserts the specified value (1210) into the field. The Modify Method (RecordRef) modifies the table. A message that indicates the G/L Account No. field has changed is displayed. To show that the code in the **OnValidate** trigger is executed, design the **G/L Entry** table and add the following code to the **G/L Account No. – OnValidate** trigger:

```
Message('The OnValidate trigger is called.');
```

```
var
    MyFieldRef: FieldRef;
    EntryRecref: RecordRef;
    Text000: Label 'The G/L Account No. for record %1 is %2.';
    Text001: Label 'The G/L Account No. for record %1 has changed to %2.';
begin
    EntryRecref.Open(17);
    if EntryRecref.FindFirst then begin
      MyFieldRef := EntryRecref.Field(3);
      Message(Text000, EntryRecref.RecordId, MyFieldRef.Value);
      MyFieldRef.Validate('1210');
      EntryRecref.Modify;
      Message(Text001, EntryRecref.RecordId, MyFieldRef.Value);
    end;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

FieldRef.Value Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets or gets the value of the field that is currently selected. This method returns an error if no field is selected.

Syntax

```
[Value := ]  FieldRef.Value([NewValue: Any])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

FieldRef Type: FieldRef An instance of the FieldRef data type.

NewValue
Type: Any

Return Value

Value Type: Any

Remarks

If you omit NewValue, the method returns the current value of the field.

If you want to format the value of a FieldRef you must use `Format(FieldRef)` instead of `Format(FieldRef.Value)`.

Example

The following example opens table 18, the **Customer** table, a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef for the first field (No.). The reference to the field is stored in the MyFieldRef variable. The Active Method (FieldRef) method determines whether the field is enabled. If the field is enabled, then the record that is identified by the number in the CustomerNo variable is selected. The MyFieldRef variable is updated to refer to the second field (Name). Then the value in the second field is changed to Contoso. The Modify Method (RecordRef) modifies the record in the table to reflect the change. `MyFieldRef.Value` retrieves the new name and displays it in message box.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    CustomerNo: Code;
    Text000: Label 'Customer name has changed to %1.';
begin
    CustomerNo := '50000';
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    if MyFieldRef.Active then begin
      MyFieldRef.Value(CustomerNo);
      MyFieldRef := CustomerRecref.Field(2);
      MyFieldRef.Value('Contoso');
      CustomerRecref.Modify;
      Message(Text000, MyFieldRef.Value);
    end;
end;
```

See Also

FieldRef Data Type
Getting Started with AL
Developing Extensions

File Data Type

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a file.

The following methods are available on the File data type.

METHOD NAME	DESCRIPTION
Copy(String, String)	Copies a file.
Download(String, String, String, String, var Text)	Sends a file from a server computer to the client computer. The client computer is the computer that is running the Windows client or the computer that is running a browser that accesses the web client.
DownloadFromStream(InStream, String, String, String, var Text)	Sends a file from server computer to the client computer. The client computer is the computer that is running the Windows client or the computer that is running the browser that accesses the web client.
Erase(String)	Deletes a file.
Exists(String)	Determines whether a file exists.
GetStamp(String, var Date [, var Time])	Gets the exact time that a file was last written to.
IsPathTemporary(String)	Validates whether the given path is located in the current users temporary folder within the current service.
Rename(String, String)	Renames an ASCII or binary file.
SetStamp(String, Date [, Time])	Sets a timestamp for a file.
Upload(String, String, String, String, var Text)	Sends a file from the client computer to the server computer. The client computer is the computer that is running the Windows client or the computer that is running a browser that accesses the web client.
UploadIntoStream(String, String, String, var Text, var InStream)	Sends a file from the client computer to the corresponding server. The client computer is the computer that is running the Windows client or the computer that is running a browser that accesses the web client.

The following methods are available on instances of the File data type.

METHOD NAME	DESCRIPTION
Close()	Closes a file that has been opened by the Open method (File).

METHOD NAME	DESCRIPTION
Create(String [, TextEncoding])	Creates an Automation object.
CreateInStream(InStream)	Creates an InStream object for a file. This enables you to import or read data from the file.
CreateOutStream(OutStream)	Creates an OutStream object for a file. This enables you to export or write data to the file.
CreateTempFile([TextEncoding])	Creates a temporary file. This enables you to save data of any format to a temporary file. This file has a unique name and will be stored in a temporary file folder.
Len()	Gets the length of an ASCII or binary file.
Name()	Gets the name of an ASCII or binary file.
Open(String [, TextEncoding])	Opens an ASCII or binary file. This method does not create the file if it does not exist.
Pos()	Gets the current position of the file pointer in an ASCII or binary file.
Read(var Any)	Reads from an MS-DOS encoded file or binary file.
Seek(Integer)	Sets a file pointer to a new position in an ASCII or binary file.
TextMode([Boolean])	Sets whether a file should be opened as an ASCII file or a binary file. Gets the current setting of this option for a file.
Trunc()	Truncate an ASCII or binary file to the current position of the file pointer.
Write(Boolean)	Writes to an MS-DOS encoded file or binary file.
Write(Byte)	Writes to an MS-DOS encoded file or binary file.
Write(Char)	Writes to an MS-DOS encoded file or binary file.
Write(Integer)	Writes to an MS-DOS encoded file or binary file.
Write(BigInteger)	Writes to an MS-DOS encoded file or binary file.
Write(Decimal)	Writes to an MS-DOS encoded file or binary file.
Write(Guid)	Writes to an MS-DOS encoded file or binary file.
Write(Text)	Writes to an MS-DOS encoded file or binary file.
Write(Code)	Writes to an MS-DOS encoded file or binary file.
Write(Label)	Writes to an MS-DOS encoded file or binary file.
Write(BigText)	Writes to an MS-DOS encoded file or binary file.

METHOD NAME	DESCRIPTION
Write(Date)	Writes to an MS-DOS encoded file or binary file.
Write(Time)	Writes to an MS-DOS encoded file or binary file.
Write(DateTime)	Writes to an MS-DOS encoded file or binary file.
Write(DateFormula)	Writes to an MS-DOS encoded file or binary file.
Write(Duration)	Writes to an MS-DOS encoded file or binary file.
Write(Option)	Writes to an MS-DOS encoded file or binary file.
Write(Record)	Writes to an MS-DOS encoded file or binary file.
Write(RecordId)	Writes to an MS-DOS encoded file or binary file.
Write(String)	Writes to an MS-DOS encoded file or binary file.
Write(Any)	Writes to an MS-DOS encoded file or binary file.
WriteMode([Boolean])	Use this method before you use Open method (File)] to set or test whether you can write to a file in later calls.

See Also

Getting Started with AL
Developing Extensions

File.Copy Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies a file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Copy(FromName: String, ToName: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

FromName
Type: String
The name of the file that you want to make a copy of, including its path. When you enter the path, consider these shortcuts:

- You can omit the drive designation if the file is located on the current drive.
- You can omit the full path if the file is located in the current directory.
- You can enter only the subdirectory name if the file is located in a subdirectory of the current directory.

ToName
Type: String
The name that you want to assign to the copy that includes its path. When you enter the path, consider these shortcuts:

- You can omit the drive designation if the file is located on the current drive.
- You can omit the full path if the file is located in the current directory.
- You can enter only the subdirectory name if the file is located in a subdirectory of the current directory.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If you do not use the return value and the file cannot be copied, a run-time error will occur. If you do include the

return value in your code, you must handle any errors yourself.

Example

The following example copies a file that is named OldFile from a folder that is named Old on drive C to a folder that is named New. If the file is copied, a message is displayed and the program continues. Otherwise, an error occurs. This example assumes that you have created the following file 'c:\Old\' OldFile.

```
 var
    OldFile: Text;
    NewFile: Text;
begin
    OldFile := 'old.txt';
    NewFile := 'new.txt';
    if File.Copy('c:\Old\' + OldFile, 'c:\New\' + NewFile) then
      // Continue your program.
      Message('The file was copied.');
    else
      // Handle the error.
      Message('The file was not copied.');
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Download Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a file from a server computer to the client computer. The client computer is the computer that is running the Windows client or the computer that is running a browser that accesses the web client.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Download(FromFile: String, DialogTitle: String, ToFolder: String, ToFilter: String, var
ToFile: Text)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

FromFile
Type: String
The name of the file on the server computer that you want to download to the client computer.

DialogTitle
Type: String
The title that you want to display in the dialog box for downloading the file. This parameter is not supported by the web client. The title is determined by the end-user's browser.

ToFolder
Type: String
The default folder in which to save the file to be downloaded. The folder name is displayed in the dialog box for downloading the file. The folder can be changed by the user. This parameter is not supported by the web client. By default, the files are saved to the default download location that is configured in the end-user's browser.

ToFilter
Type: String
The type of file that can be downloaded to the client computer. The type is displayed in the dialog box for downloading the file. This parameter is not supported by the web client.

ToFile
Type: Text
The name to give the downloaded file. This is the default file name that is shown in the dialog box for downloading the file. This value can be changed by the user.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

> **NOTE**
> On devices that run Apple iOS, such as iPad, you can only download a file if the Apple iOS device on which you are downloading the file has an application which supports the file type.

Files are saved to the default download location that is configured in the end-user's browser.

The business logic is run on the computer that is running Dynamics 365 Business Central service and not on the client. Files are created on the computer that is running Dynamics 365 Business Central service and not locally on the client computer.

Upload Method (File) and UploadIntoStream Method (File) are used to send a file from the client to a Dynamics 365 Business Central service instance.

Download Method (File) and DownloadFromStream Method (File) are used to send a file from a Dynamics 365 Business Central service instance to the client.

We recommend that you use the methods in codeunit **419 File Management** to upload and download files.

> **NOTE**
> Internet browsers can only handle one file per request. Therefore, with the Web client, if this method is called in a repetitive statement (or loop) that generates multiple files, only the last file will be sent to the browser. Alternatively, when designing for the Web client, bundle the files in an archive file (.zip), for example, by using the methods found in codeunit **419 File Management**. For more details about this design pattern, see Multi-File Download. Although this article is written for Dynamics NAV, it is still relevant for Business Central. The methods in codeunit 419 are not external, therefore cannot be used in extensions. Instead, when developing extensions in AL, use the external methods of codeunit **425 Data Compression**. The approach is similar.

Example

This example shows how to use the Download method.

```
 var
    ToFile: Text;
begin
    ToFile := 'ToFile.txt';
    Download('FromFile.txt','Download file','C:\','Text file(*.txt)|*.txt',ToFile);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.DownloadFromStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a file from server computer to the client computer. The client computer is the computer that is running the Windows client or the computer that is running the browser that accesses the web client.

Syntax

```
[Ok := ]  File.DownloadFromStream(InStream: InStream, DialogTitle: String, ToFolder: String, ToFilter:
String, var ToFile: Text)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

InStream

Type: InStream

An InStream that you want to use to send the data in a file on Business Central Server to a file on the client computer.

DialogTitle

Type: String

The title that you want to display in the dialog box for downloading the file. This parameter is not supported by the web client. The title is determined by the end-user's browser.

ToFolder

Type: String

The default folder in which to save the file to be downloaded. The folder name is displayed in the dialog box for downloading the file. The folder can be changed by the user. This parameter is not supported by the web client. By default, files are saved to the default download location that is configured in the end-user's browser.

ToFilter

Type: String

The type of file that can be downloaded to the client computer. The type is displayed in the dialog box for downloading the file. This parameter is not supported by the web client.

ToFile

Type: Text

The name to give the downloaded file. This is the default file name that is shown in the dialog box for downloading the file. This value, can be changed by the user.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

> **NOTE**
> On devices that run Apple iOS, such as iPad, you can only download a file if the Apple iOS device on which you are downloading the file has an application that supports the file type.

The business logic runs on the computer that is running Dynamics 365 Business Central service and not on the client. Files are created on a Dynamics 365 service and not locally on the client computer. When you write code, you must consider where files are created.

Use Upload Method (File) and UploadIntoStream Method (File) to send a file from a client to a Dynamics 365 Business Central service instance.

Use Download Method (File) and DownloadFromStream Method (File) to send a file from a Dynamics 365 Business Central service instance to a client.

We recommend that you use the methods in codeunit **419 File Management** to upload and download files.

> **NOTE**
> Internet browsers can only handle one file per request. Therefore, with the Web client, if this method is called in a repetitive statement (or loop) that generates multiple files, only the last file will be sent to the browser. Alternatively, when designing for the Web client, bundle the files in an archive file (.zip), for example, by using the methods found in codeunit **419 File Management**. For more details about this design pattern, see Multi-File Download. Although this article is written for Dynamics NAV, it is still relevant for Business Central. The methods in codeunit 419 are not external, therefore cannot be used in extensions. Instead, when developing extensions in AL, use the external methods of codeunit **425 Data Compression**. The approach is similar.

Example

```
 var
    TempFile: File;
    NewStream: InsTream;
    ToFileName: Variant;
begin
    TempFile.CreateTempFile();
    TempFile.Write('abc');
    TempFile.CreateInStream(NewStream);
    ToFileName := 'SampleFile.txt';
    DownloadFromStream(NewStream,'Export','','All Files (*.*)|*.*',ToFileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Erase Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes a file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Erase(Name: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Name
Type: String
The name of the file that you want to delete, including the path. When you enter the path, consider these shortcuts:

- You can omit the drive designation if the file is located on the current drive.
- You can omit the full path if the file is located in the current directory.
- You can enter only the subdirectory name if the file is located in a subdirectory of the current directory.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

If the user who runs this method does not have the required permission to delete the file or if the file is read-only, then the file is not deleted.

Example

The following example deletes the file that is named C:\TestFolder\NewTestFile.txt.This example assumes that you have created the file on your computer.

```
Erase('C:\TestFolder\NewTestFile.txt');
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Exists Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a file exists.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Exists(Name: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Name
Type: String
The name of the file that you want to check. This includes the path. When you enter the path, consider these shortcuts:

- You can omit the drive designation if the file is located on the current drive.
- You can omit the full path if the file is located in the current directory.
- You can enter only the subdirectory name if the file is located in a subdirectory of the current directory.

Return Value

Ok Type: Boolean **true** if the server instance has access to the file; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

The following example uses the Exists method to determine whether the specified file exists. If the file exists, then the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file, and then the Close Method (File) method closes the file. If the file does not exist, an error message is displayed. This example assumes that you have created the following file C:\TestFolder\TestFile2.txt.

```
 var
    TestFile: File;
    FileName: Text;
begin
    FileName := 'C:\TestFolder\TestFile2.txt';
    if exists(FileName) then begin
      TestFile.WriteMode(true);
      TestFile.Open(FileName);
      TestFile.Write('Hello World');
      TestFile.Close;
    end else
        Message('%1 does not exist.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.GetStamp Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the exact time that a file was last written to.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.GetStamp(Name: String, var Date: Date [, var Time: Time])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Name
Type: String
The name of the file, including the path. When you enter the path, consider these shortcuts:

- You can omit the drive designation if the file is located on the current drive.
- You can omit the full path if the file is located in the current directory.
- You can enter only the subdirectory name if the file is located in a subdirectory of the current directory.

Date
Type: Date
The date that the file was last written to.

Time
Type: Time
The time that the file was last written to. Optional.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

The following example gets the date and time that a file was written to and displays the data in a message box. The code example assumes that you have created the file 'C:\MyFolder\MyText.txt'. This example requires that you create the following global variables and text constant.

```
 var
    varDate: Date;
    varTime: Time;
    varFileName: Text;
    Text000: Label 'This document was last written to on %1 at %2.';
begin
    varFileName := 'C:\MyFolder\MyText.txt';
    GetStamp(VarFileName, varDate, varTime);
    Message(Text000, varDate, varTime);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.IsPathTemporary Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Validates whether the given path is located in the current users temporary folder within the current service.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.IsPathTemporary(Name: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Name
Type: String
The name of the file, including the path.

Return Value

Ok Type: Boolean **true** if the name point to a location is the users temporary folder within the current service; **false** otherwise. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Rename Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Renames an ASCII or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Rename(OldName: String, NewName: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

OldName
Type: String
The current name of the file that you want to change, including its path. When you enter the path, consider these shortcuts:

- You can omit the drive designation, if the file is located on the current drive.
- You can omit the full path, if the file is located in the current directory.
- You can enter only the subdirectory name, if the file is located in a subdirectory of the current directory.

NewName
Type: String
The new name that you want to assign to the file, including its path. When you enter the path, consider these shortcuts:

- You can omit the drive designation, if the file is located on the current drive.
- You can omit the full path, if the file is located in the current directory.
- You can enter only the subdirectory name, if the file is located in a subdirectory of the current directory.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Typically, the return value is **false** if the file does not exist, or if the file is a system or hidden file.

Example

The following example changes the name of a text file that is named Testfile.txt to NewTestFile.txt. The path of the file that is renamed is C:\TestFolder\Testfile.txt. The name and path are stored in the varOldFile variable. The new name and path of the file are stored the varNewfile variable. The RENAME method uses the variables to change the name of the file. This example assumes that you have created the following file on your computer: C:\TestFolder\Testfile.txt.

```
 var
    varOldfile: Text;
    varNewfile: Text;
begin
    varOldfile := 'C:\TestFolder\Testfile.txt' ;
    varNewfile := 'C:\TestFolder\NewTestFile.txt';
    Rename(varOldfile, varNewfile);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.SetStamp Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a timestamp for a file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.SetStamp(Name: String, Date: Date [, Time: Time])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Name
Type: String
The name of the file, including its path. When you enter the path, keep in mind these shortcuts:

- You can omit the drive designation, if the file is located on the current drive.
- You can omit the full path, if the file is located in the current directory.
- You can enter only the subdirectory name, if the file is located in a subdirectory of the current directory.

Date
Type: Date
The date that you want stamped on the file.

Time
Type: Time
The time that you want stamped on the file.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

The following example sets timestamp for that a file that is named varFileName. The date and time are set to the current date and on your computer respectively. The code example assumes that you have created the following file: 'C:\MyFolder\MyText.txt'. The following example requires that you create the following global variables and text constant.

```
 var
    varFileName: Text;
    varDate: Date;
    varTime: Time;
    Text000: Label 'The timestamp for this file is Date: %1 Time: %2.';
begin
    VarFileName := 'C:\MyFolder\MyText.txt';
    varDate := Today;
    varTime := Time;
    SetStamp(VarFileName, varDate, varTime);
    Message(Text000, varDate, varTime);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Upload Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a file from the client computer to the server computer. The client computer is the computer that is running the Windows client or the computer that is running a browser that accesses the web client.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Upload(DialogTitle: String, FromFolder: String, FromFilter: String, FromFile: String, var
ToFile: Text)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

DialogTitle

Type: String

The title that you want to display in the dialog box for uploading the file. This parameter is not supported by the web client. The title is determined by the end-user's browser.

FromFolder

Type: String

The name of the folder that is displayed in the dialog box. This is the default value, and the user can change it. This parameter is not supported by the web client. The browser uses the folder that was last accessed.

FromFilter

Type: String

The type of file that can be uploaded to the server. In the Windows client, the type is displayed in the upload dialog box, so that the user can only select files of the specified type. For the web client, a user can try to upload any file type but an error occurs if the file is not the specified type.

FromFile

Type: String

The default file that you want to upload to the service. The name displays in the dialog box for uploading the file. The user can change the file. This parameter is not supported by the web client.

ToFile

Type: Text

The path and file name to give the uploaded file. If you do not provide a path, or you upload the file that uses web client, then the file is uploaded to the following folder on the computer that is running the server:

\ProgramData\Microsoft\Microsoft Dynamics NAV\110\Server\MicrosoftDynamicsNAVServer$DynamicsNAV110\users\ServiceAccount

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

> **NOTE**
> This method is not supported on devices that run Apple iOS, such as iPad. The dialog box for uploading a file displays, but it is disabled and the user cannot select a file.

The business logic is run on the Dynamics 365 Business Central service and not on the client. Files are created on the Dynamics 365 Business Central service and not locally on the client.

Upload Method (File) and UploadIntoStream Method (File) are used to send a file from the client to a Dynamics 365 Business Central service.

Download Method (File) and DownloadFromStream Method (File) are used to send a file from a Dynamics 365 Business Central service to the client.

We recommend that you use the methods in codeunit 419, File Management, to upload and download files.

Example

```
Upload('Upload file','C:\','Text file(*.txt)|*.txt','Test.txt',varTest);
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.UploadIntoStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a file from the client computer to the corresponding server. The client computer is the computer that is running the Windows client or the computer that is running a browser that accesses the web client.

Syntax

```
[Ok := ]  File.UploadIntoStream(DialogTitle: String, FromFolder: String, FromFilter: String, var FromFile:
Text, var InStream: InStream)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

DialogTitle

Type: String

The text displayed in the title bar of the Open dialog box. This parameter is not supported by the web client. The title is determined by the end-user's browser.

FromFolder

Type: String

The path of the folder that is displayed in the File Open dialog box. This is the default folder, but the user can browse to any available location. This parameter is not supported by the web client. By default, the browser uses the folder that was last accessed.

FromFilter

Type: String

The type of file that can be uploaded to the server. In the Windows client, the type is displayed in the upload dialog box, so the user can only select files of the specified type. For the web client, a user can try to upload any file type but an error occurs if the file is not the specified type.

FromFile

Type: Text

The default file to upload to the service. The name displays in the dialog box for uploading the file. The user can change the file. This parameter is not supported by the web client.

InStream

Type: InStream

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

> **NOTE**
>
> This method is not supported on devices that run Apple iOS, such as iPad. The dialog box for uploading a file displays, but it is turned off and the user cannot select a file.

The business logic runs on the Dynamics 365 Business Central service and not on the client. Files are created on Dynamics 365 Business Central service and not locally on the client computer.

Use Upload Method (File) and UploadIntoStream Method (File) to send a file from the client to the Dynamics 365 Business Central service.

Use Download Method (File) and DownloadFromStream Method (File) to send a file from the Dynamics 365 Business Central service to the client.

We recommend that you use the methods in codeunit 419, File Management, to upload and download files.

Example

```
 var
    FileName: Text;
    NVInStream: InStream;
begin
    FileName := 'c:\SomeFile.txt';
    UploadIntoStream('Import','',' All Files (*.*)|*.*',FileName,NVInStream);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes a file that has been opened by the Open method (File).

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Close()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Remarks

If the file is not open, a run-time error will occur.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file, and then the Close method closes the file. If the file does not exist, an error message is displayed. This example assumes that you have created the following file C:\TestFolder\TestFile2.txt.

```
var
    FileName: Text;
    TestFile: File;
begin
    FileName := 'C:\TestFolder\TestFile2.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(true);
      TestFile.Open(FileName);
      TestFile.Write('Hello World');
      TestFile.Close;
    end else
    Message('%1 does not exist.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an Automation object.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Create(Name: String [, Encoding: TextEncoding])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Name
Type: String

Encoding
Type: TextEncoding

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the TextMode Method (File) returns **true** and you read or write to the file, text is put in the buffer.

If the TextMode Method (File) method returns **false**, binary information is put in the buffer.

If you call Create on a File variable that refers to an open file, the method does not automatically close the existing file and create the new file. You must explicitly call the Close Method (File) to close the existing file. Otherwise, a run-time error occurs.

Example

The following example creates a file that is named TestFile.txt in the path C:\TestFolder\. The TestFile variable stores the file and path that is created. If the file is created, a message that states that the file is created is displayed. Otherwise, an error message is displayed. This example requires that you create the following global

variable.

```
 var
    TestFile: File;
begin
    if TestFile.Create('C:\TestFolder\TestFile.txt') then begin
      Message('%1 is created', TestFile.Name);
    end else
    Error('The file could not be created');
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.CreateInStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an InStream object for a file. This enables you to import or read data from the file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.CreateInStream(InStream: InStream)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

InStream
Type: InStream

Example

The following example imports data from an XML document to a table. The code uses the Open Method (File) to open the XML document named NewCustomer.xml from a folder named Import on the C drive. The CreateInStream Method (File) creates a data stream to get the data from the XML document into the table. The Import Method (XMLport) then imports, parses the data, and adds it as a record to the table by using an XMLport (50004). The Close Method (File) then closes the data stream. This example assumes that you have created a NewCustomer.xml file in a folder that is named Import on the C drive and you have created an XMLport and given it ID 50004.

```
 var
    ImportXmlFile: File;
    XmlINStream: InStream;
begin
    ImportXmlFile.Open('C:\Import\NewCustomer.xml');
    ImportXmlFile.CreateInStream(XmlINStream);
    XMLPORT.Import(50004, XmlINStream);
    ImportXmlFile.Close;
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.CreateOutStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an OutStream object for a file. This enables you to export or write data to the file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.CreateOutStream(OutStream: OutStream)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

OutStream
Type: OutStream

Example

The following example uses the CreateOUTSTREAM method to export data from a table to an XML document. The code uses the Create Method (File) to create an XML file that is named CustXmlFile.xml in a folder that is named xmlData on drive C. The CreateOUTSTREAM Method (File) opens a data stream to output the data from the table to the XML file. The Export Method (XMLport) then exports the data and saves it at the specified location. The Close Method (File) closes the data stream. This example assumes that you have created a folder named xmlData on drive C.

```
 var
    CustXmlFile: File;
    XmlStream: OutStream;
begin
    CustXmlFile.Create('C:\XmlData\Customer.xml');
    CustXmlFile.CreateOUTSTREAM(XmlStream);
    XMLPORT.Export(50002, XmlStream);
    CustXmlFile.Close;
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.CreateTempFile Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a temporary file. This enables you to save data of any format to a temporary file. This file has a unique name and will be stored in a temporary file folder.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.CreateTempFile([Encoding: TextEncoding])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Encoding
Type: TextEncoding

Return Value

Ok Type: Boolean

Remarks

You can use this method together with Name Method (File) and Close Method (File).

Example

This example creates a temporary file that has the text Hello and then deletes the file by using the File.Close method.

```
 var
    FileName: File;
begin
    FileName.CreateTempFile;
    FileName.Write('Hello');
    FileName.Close;
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Len Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the length of an ASCII or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Length :=   File.Len()
```

> **NOTE**
> This method can be invoked using property access syntax.

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Return Value

Length Type: Integer

Remarks

This method is often used with Pos Method (File) and Seek Method (File).

Example

The following example opens a text file that is named 'C:\TestFolder\TestFile.txt' and contains the text 'Hello World'. The Seek Method (File) sets a pointer to position 6 in the file. The Read Method (File) reads the file and stores the retrieved contents in the varString variable. The LEN method retrieves the length of the file and stores it the varLength variable. The text that is read starts from the position of the pointer, so the text 'World' and the length of 12 are displayed in the message box. The length of the file is not affected by the Seek Method (File). This example assumes that you have created the text file that is named C:\TestFolder\TestFile.txt and contains the text 'Hello World'. This example requires that you create the following global variables.

```
var
   Testfile: File;
   varString: Text[200];
   varLength: Integer;
begin
   Testfile.Open('C:\TestFolder\TestFile.txt');
   Testfile.Seek(6);
   Testfile.Read(varString);
   varLength := Testfile.LEN;
   Message('The text is: %1. The length of the file is: %2', varString, varLength);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of an ASCII or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Name :=   File.Name()
```

> **NOTE**
> This method can be invoked using property access syntax.

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Return Value

Name Type: String

Remarks

You must use the Open Method (File) to open the file before you can use this method.

Example

The following example opens a text file that is named C:\TestFolder\TestFile.txt. The Name Method (File) retrieves the name and path of the text file and stores it in the varName variable. The value in the variable is displayed in a message box. This example assumes that you have created a text file named C:\TestFolder\TestFile.txt.

```
var
   Testfile: File;
   varName: Text;
begin
   TestFile.Open('C:\TestFolder\TestFile.txt');
   varName := TestFile.Name;
   Message('The name of the file is: %1',varName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Open Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Opens an ASCII or binary file. This method does not create the file if it does not exist.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  File.Open(Name: String [, Encoding: TextEncoding])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Name
Type: String

Encoding
Type: TextEncoding

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If you call Open on a file variable that refers to an open file, then the method does not automatically close the existing file and open the new file. You must explicitly call the Close Method (File) to close the existing file. If you call Open on a file that is already open, then a run-time error occurs.

Example

This example shows how to open an .xml file for reading in text mode. To use this example, you must create the simple.xml file at C:\temp and create the following variable.

```
var
   TestFile: File;
```

```
TestFile.TextMode(True);
TestFile.WriteMode(False);
TestFile.Open('C:\temp\simple.xml');
```

If you want to be explicit about the encoding of a file, you can set the *TextEncoding* parameter. The following code example replaces the last statement in the previous example.

```
TestFile.TextMode(True);
TestFile.WriteMode(False);
TestFile.Open('C:\temp\simple.xml', TextEncoding::Windows);
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Pos Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current position of the file pointer in an ASCII or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Position :=   File.Pos()
```

> **NOTE**
> This method can be invoked using property access syntax.

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Return Value

Position Type: Integer

Remarks

This method is often used with Len Method (File) and Seek Method (File).

Example

The following example opens a text file that is named C:\TestFolder\TestFile.txt. The WriteMode Method (File) enables the file to be open in write mode. The POS method retrieves the position of the file pointer and stores it in the Position variable. When the file is open, the position of the pointer is 0 because a pointer is not set. The Seek Method (File) method sets a file pointer at position 5. After the Seek Method (File) is executed, the POS method returns 5 as the file pointer position. This example assumes that you have created a text file named C:\TestFolder\TestFile.txt.

```
 var
    Testfile: File;
    Position: Integer;
begin
    File.WriteMode(true);
    TestFile.Open('C:\TestFolder\TestFile.txt');
    Position := TestFile.Pos;
    Message('Pointer position before Seek: %1', Position);
    Testfile.Seek(5);
    Position := Testfile.POS;
    Message('Pointer position after Seek: %1', Position);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads from an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Read := ]  File.Read(var Read: Any)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Read
Type: Any
Streams a BigText object that is stored as a BLOB in a table to a BigText variable.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

To read from a file that is larger than 1024 bytes, use streams instead of the **File.Read** method.

MS-DOS encoding, which is also referred to as OEM encoding, is an older format than UTF-8 and UTF-16, but it is still widely supported. MS-DOS encoding was the only format that was supported by earlier versions of Dynamics 365.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you read a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data in the file that is being read. For example, if the file contains text in Danish, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to Danish before you call the **Read** method (File) or Write Method (File).

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example opens a text file that is named C:\TestFolder\TestFile.txt. The READ method read the contents of the file and stores it in the String variable. The method returns the size of the text that was read, stores it in the varSize variable, and displays it in a message box. This example assumes that you have created a text file named C:\TestFolder\TestFile.txt that contains less than 500 bytes.

```
 var
    Testfile: File;
    String: Text[500];
    varSize: Integer;
begin
    TestFile.Open('C:\TestFolder\TestFile.txt');
    varSize := TestFile.Read(String);
    Message('The text "%1" is %2 bytes.', String, varSize);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Seek Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a file pointer to a new position in an ASCII or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Seek(Position: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Position
Type: Integer
The position at which to set the pointer.

Remarks

This method is often used with Pos Method (File) and Len Method (File).

Example

The following example sets a pointer at position 20 in a file and truncates the file at the pointer position. The WriteMode Method (File) enables a file named C:\TestFolder\TestFile.txt to open in write mode. The Seek method sets a pointer at position 20 in the file and then the Trunc Method (File) truncates the contents of the file at the pointer position. This example assumes that you have created the text file C:\TestFolder\TestFile.txt.

```
 var
    TestFile: File;
begin
    TestFile.WriteMode(True);
    TestFile.Open('C:\TestFolder\TestFile.txt');
    TestFile.Seek(20);
    TestFile.Trunc;
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.TextMode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets whether a file should be opened as an ASCII file or a binary file. Gets the current setting of this option for a file.

NOTE

This method is supported only in Business Central on-premises.

Syntax

```
[Textmode := ]  File.TextMode([Mode: Boolean])
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Mode
Type: Boolean

Return Value

Textmode Type: Boolean

Remarks

This method should be used before File.Open is used to open the file. If you use this method on a file that is already open, then an error occurs.

Example

The following example sets the TextMode to **true** when the file is open for writing. This means the file contents will be written to a text file that is named 'C:\TestFolder\TestFile.txt' by using ASCII characters. The WriteMode Method (File) and the Open Method (File) open the file for writing and the text 'Hello World' is written. The Close Method (File) closes the file after the file is written to. This example assumes that you have created a text file that is named C:\TestFolder\TestFile.txt.

```
var
    TestFile: File;
begin
    TestFile.TextMode(True);
    TestFile.WriteMode(True);
    TestFile.Open('C:\TestFolder\TestFile.txt');
    TestFile.Write('Hello World');
    TestFile.Close;
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Trunc Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Truncate an ASCII or binary file to the current position of the file pointer.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Trunc()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Remarks

Typically, you use this method together with Seek Method (File). Use File.Seek to position the pointer in the file and then use File.Trunc to truncate the file at that point.

Example

The following example sets a pointer at position 20 in a file and truncates the file at the pointer position. The WriteMode Method (File) allows the file that is named C:\TestFolder\TestFile.txt to open in write mode. The Seek method sets a pointer at position 20 in the file and then the Trunc Method (File) truncates the contents at the pointer position. This example assumes that you have created a text file named C:\TestFolder\TestFile.txt. The file is then saved a truncated file.

```
 var
    TestFile: File;
begin
    TestFile.WriteMode(True);
    TestFile.Open('C:\TestFolder\TestFile.txt');
    TestFile.Seek(20);
    TestFile.Trunc;
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Boolean)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Boolean
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  Bool: Boolean;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Bool);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Byte)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Byte
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt. .

```
var
  FileName: Text;
  TestFile: File;
  Bte: Byte;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Bte);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Char)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Char
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  Chr: Char;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Chr);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Integer
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  Inte: Integer;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Int);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: BigInteger)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: BigInteger
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(true);
      TestFile.Open(FileName);
      TestFile.Write('Hello World');
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Decimal)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Decimal
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  DteTme: DateTime;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Dcml);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Guid)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Guid
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  Guid1: GUID;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Guid1);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Text)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Text
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    Txt: Text;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write(Txt);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Code)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Code
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt. This example requires that you create the following global variables.

```
var
  FileName: Text;
  TestFile: File;
  Cde: Code;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Cde);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Label)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Label
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    Lbl: Label 'HelloWorld';
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write(Lbl);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: BigText)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: BigText
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    BigTxt: BigText;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(true);
      TestFile.Open(FileName);
      TestFile.Write(BigTxt);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Date)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Date
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  Dte: Date;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Dte);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Time)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Time
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    Tme: Time;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write(Tme);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: DateTime)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: DateTime
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  DteTme: DateTime;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(DteTme);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: DateFormula)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: DateFormula
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  DteForm: DateFormula;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(DteForm);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Duration)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Duration
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
var
  FileName: Text;
  TestFile: File;
  Dur: Duration;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write(Dur);
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
  ```SAGE('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions
```

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Option)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Option
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    Opt: Option;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write(Opt);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Record)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Record
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    Rec: Record Customer;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write(Rec);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: RecordId)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: RecordId
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
    RecId: RecordId;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write(RecId);
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: String
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write('Hello World');
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes to an MS-DOS encoded file or binary file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
File.Write(Value: Any)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Value
Type: Any
The data that you want to write to the file.

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt. This example requires that you create the following global variables.

```
var
  FileName: Text;
  TestFile: File;
begin
  FileName := 'C:\TestFolder\TestFile.txt';
  if Exists(FileName) then begin
    TestFile.WriteMode(True);
    TestFile.Open(FileName);
    TestFile.Write('Hello World');
    TestFile.Close;
  end else
    Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

File.WriteMode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Use this method before you use Open method (File)] to set or test whether you can write to a file in later calls.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Writemode := ]  File.WriteMode([Mode: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

File Type: File An instance of the File data type.

Mode
Type: Boolean

Return Value

Writemode Type: Boolean

Remarks

You must call the TextMode Method (File) before you call the Write method.

If TextMode Method (File) is set to **true** and *Value* is an integer, then the integer is written as text, followed by a new line character.

If *Value* is a record, each field is separated by a tab character.

If TextMode Method (File) is **false** and *Value* is an integer, an integer is written that is four bytes long.

MS-DOS encoding requires a different character set for each language. MS-DOS text is encoded to the internal Unicode data type by using the system locale language of the computer that is running Dynamics 365 Business Central service. If you write to a file that uses MS-DOS encoding, then you must set the system locale language

of the computer that is running Dynamics 365 Business Central service to match the language of the data that you want to write to the file.

We recommend that you use the File data type for files that were created in earlier versions of Dynamics 365.

To read or write files in Unicode or in other formats, we recommend that you use .NET Framework interoperability and use the System.IO Namespace.

Example

The following example determines whether the specified file exists. If it exists, the WriteMode Method (File) allows the file to be open for writing. The Open Method (File) opens the file, the Write Method (File) writes the text "Hello World" to the file and then the Close Method (File) closes the file. If the file does not exist, then an error message is displayed. This example assumes that you have created the following file: C:\TestFolder\TestFile.txt.

```
 var
    FileName: Text;
    TestFile: File;
begin
    FileName := 'C:\TestFolder\TestFile.txt';
    if Exists(FileName) then begin
      TestFile.WriteMode(True);
      TestFile.Open(FileName);
      TestFile.Write('Hello World');
      TestFile.Close;
    end else
      Message('%1 does not exit.', FileName);
end;
```

See Also

File Data Type
Getting Started with AL
Developing Extensions

FilterPageBuilder Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stores filter configurations for a filter page. A filter page is a dynamic page type that contains one or more filter controls that enables users to set filters on fields of the underlying tables.

The following methods are available on instances of the FilterPageBuilder data type.

METHOD NAME	DESCRIPTION
AddField(String, Any [, String])	Adds a table field to the filter control for a table on filter page.
AddField(String, FieldRef [, String])	Adds a table field to the filter control for a table on filter page.
AddFieldNo(String, Integer [, String])	Adds a table field to the filter control for a table on the filter page.
AddRecord(String, Record)	Adds a filter control for a table to a filter page. The table is specified by a record data type variable that is passed to the method.
AddRecordRef(String, RecordRef)	Adds a filter control for a table to a filter page. The table is specified by a RecordRef variable that is passed to the method. This creates a filter control on the filter page, where users can set filter table data.
AddTable(String, Integer)	Adds filter control for a table to a filter page.
Count()	Gets the number of filter controls that are specified in the FilterPageBuilder object instance.
GetView(String [, Boolean])	Gets the filter view (which defines the sort order, key, and filters) for the record in the specified filter control of a filter page. The view contains all fields in the filter control that have a default filter value.
Name(Integer)	Gets the name of a table filter control that is included on a filter page based on an index number that is assigned to the filter control.
PageCaption([String])	Gets or sets the FilterPageBuilder UI caption. Defaults to the resource text if not explicitly set.
RunModal()	Builds and runs the filter page that includes the filter controls that are stored in FilterPageBuilder object instance.

METHOD NAME	DESCRIPTION
SetView(String, String)	Sets the current filter view, which defines the sort order, key, and filters, for a record in a filter control on a filter page. The view contains all fields that have default filters, but does not contain fields without filters.

See Also

Getting Started with AL
Creating Filter Pages for Tables
Developing Extensions

FilterPageBuilder.AddField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a table field to the filter control for a table on filter page.

Syntax

```
[Ok := ]  FilterPageBuilder.AddField(Name: String, Field: Any [, Filter: String])
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name
Type: String

Field
Type: Any
The name of the table field to add to the filter control for a table.

Filter
Type: String
A default filter on the field that is specified by the Field parameter.

Return Value

Ok Type: Boolean **true** if the field was added to the field list for the specified filter control; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the filter page implementation will call a SetView Method, then the SetView Method must be called before the AddField method call, otherwise the filter that is specified by the *Filter* parameter will be cleared by SetView Method.

The filter that is specified by the *Filter* parameter will overwrite any previously defined filters for the field which were set by AddView method or read from the record or recordRef instance that defined the filter control.

Example

The following example initializes a filter page object that includes a filter control for the **Date** system table. The filter control has the caption of **Date record**. The example adds two fields of the **Date** record variable which will be available in the filter control on the filter page: **Period End** and **Period Start**. A default filter is set on the **Period End** field.

```
var
    varDateItem|: Text[30];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddRecord(varDateItem, varDateRecord);
    varFilterPageBuilder.AddField(varDateItem, varDateRecord."Period End",'12122015D');
    varFilterPageBuilder.AddField(varDateItem, varDateRecord."Period Start");
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.AddField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Adds a table field to the filter control for a table on filter page.

Syntax

```
[Ok := ]  FilterPageBuilder.AddField(Name: String, Field: FieldRef [, Filter: String])
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name
Type: String
The name that is assigned to the table in the filter control. This value must match the value of the Name parameter that was specified by AddTable, AddRecord, or AddRecordRef method that adds the table to the filter control.

Field
Type: FieldRef
The name of the table field to add to the filter control for a table.

Filter
Type: String
A default filter on the field that is specified by the Field parameter.

Return Value

Ok Type: Boolean **true** if the field was added to the field list for the specified filter control, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the filter page implementation will call a SetView method, then the SetView method must be called before the AddField method call, otherwise the filter that is specified by the *Filter* parameter will be cleared by SetView.

The filter that is specified by the *Filter* parameter will overwrite any previously defined filters for the field which were set by AddView method or read from the record or recordRef instance that defined the filter control.

Example

The following example initializes a filter page object that includes a filter control for the **Date** system table. The filter control has the caption of **Date record**. The example adds two fields of the **Date** record variable which will be available in the filter control on the filter page: **Period End** and **Period Start**. A default filter is set on the **Period End** field.

```
var
    varDateItem: Text[30];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddRecord(varDateItem, varDateRecord);
    varFilterPageBuilder.AddField(varDateItem, varDateRecord."Period End",'12122015D');
    varFilterPageBuilder.AddField(varDateItem, varDateRecord.);
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.AddFieldNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a table field to the filter control for a table on the filter page.

Syntax

```
[Ok := ]  FilterPageBuilder.AddFieldNo(Name: String, FieldNo: Integer [, Filter: String])
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name
Type: String
The name that is assigned to the table in the filter control. This value must match the value of the Name parameter that was specified by AddTable, AddRecord, or AddRecordRef method that adds the table to the filter control.

FieldNo
Type: Integer
The number that is assigned to the field in the table as specified by the Field No. Property.

Filter
Type: String
A default filter on the field that is specified by the Field parameter.

Return Value

Ok Type: Boolean **true** if the field was added to the field list for the specified filter control; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the filter page implementation will call a SetView Method, then the SetView Method must be called before the AddFieldNo method call, otherwise the default filter that is specified by the *Filter* parameter for field, if any, will be cleared by SetView Method.

The filter that is specified by the *Filter* parameter will overwrite any previously defined filters for the field which were set by AddView method or read from the record or recordRef instance that defined the filter control.

Example

The following example initializes a filter page object that includes a filter control for the **Date** system table. The filter control has the caption of **Date record**. The example adds two fields of the **Date** record variable which will be available in the filter control on the filter page: **Period End** and **Period Start**. A default filter is set on the **Period End** field.

```
var
    varDateItem: Text[30];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddRecord(varDateItem, varDateRecord);
    varFilterPageBuilder.AddFieldNo(varDateItem, varDateRecord.FieldNo(varDateRecord."Period
End"),'03032014D');
    varFilterPageBuilder.AddFieldNo(varDateItem, varDateRecord.FieldNo(varDateRecord."Period Start"));
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.AddRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a filter control for a table to a filter page. The table is specified by a record data type variable that is passed to the method.

Syntax

```
[Name := ]  FilterPageBuilder.AddRecord(Name: String, Record: Record)
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name
Type: String
Assigns a name to the filter control for the table. The text displays as the caption for the filter control on the rendered filter page in the client.

Record
Type: Record
The record to use in the filter control.

Return Value

Name Type: String The text that is specified by the Name parameter. If an error occurs at runtime, an empty text string is returned. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

In the filter page that is rendered in the client, the AddRecord method defines a filter control for the specified table where the user can set filters on specific fields in the table.

Fields in the table that already have filters are automatically included in the filter control. Filters in the record passed to the method will not be modified by any method in the FilterPageBuilder object.

Example

The following example initializes a filter page object that includes a filter control that uses the Date system table. The filter control has the caption of **Date record**. The example set two filters are on the **Date** record variable, which results in a filter control that includes two fields by default.

```
var
    varDateItem: Text[30];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varDateRecord.SetFilter("Period End",'12122015D');
    varDateRecord.SetFilter("Period Start",'01012015D');
    varFilterPageBuilder.AddRecord(varDateItem, varDateRecord);
    varFilterPageBuilder.RunModal();

end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.AddRecordRef Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a filter control for a table to a filter page. The table is specified by a RecordRef variable that is passed to the method. This creates a filter control on the filter page, where users can set filter table data.

Syntax

```
[Name := ]  FilterPageBuilder.AddRecordRef(Name: String, RecordRef: RecordRef)
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name
Type: String
Assigns a name to the filter control for the table. The text displays as the caption for the filter control on the rendered filter page in the client.

RecordRef
Type: RecordRef
The record reference to use in the filter control.

Return Value

Name Type: String The text that is specified by the Name parameter. If an error occurs at runtime, an empty text string is returned. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

In the filter page that is rendered in the client, the AddRecordREF method defines a filter control for the specified table where the user can add and set the filters on the table.

Fields in the table that already have filters are automatically included in the filter control. Filters in the record reference passed to the method will not be modified by any method in the FilterPageBuilder object.

Example

The following example initializes a filter page object that includes a filter control that uses the Date system table. The filter control has the caption of **Date record**. The example set two filters are on the **Date** record variable, which results in a filter control that includes two fields by default.

```
var

    varDateItem: Text[30];
    varDateRecord: Record Date;
    varDateRecordRef: RecordRef;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varDateRecord.SetFilter("Period End",'12122015D');
    varDateRecord.SetFilter("Period Start",'01012015D');
    varDateRecordRef.GetTable(varDateRecord);
    varFilterPageBuilder.AddRecordREF(varDateItem,varDateRecordRef);
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.AddTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds filter control for a table to a filter page.

Syntax

```
[Name := ]  FilterPageBuilder.AddTable(Name: String, TableNo: Integer)
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name
Type: String
Assigns a name to the filter control for the table. The text displays as the caption for the filter control on the rendered filter page in the client.

TableNo
Type: Integer
The ID of the table object that you want to filter on.

Return Value

Name Type: String The text that is specified by the Name parameter. If an error occurs at runtime, an empty text string is returned. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

In the filter page that is rendered in the client, the AddTable method defines a filter control for the specified table where the user can set filters on specific fields in the table.

You can use the **AddField Method** or AddFieldNo Method method to add field of the table to the filter control.

Example

The following example initializes a filter page object that includes a filter control that uses the Date system table. The filter control has the caption of **Date record**.

```
var
    varDateItem: Text[30];
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddTable(varDateItem, Database::Date);
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of filter controls that are specified in the FilterPageBuilder object instance.

Syntax

```
Count :=   FilterPageBuilder.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Return Value

Count Type: Integer The number of filter controls in the current FilterPageBuilder object instance.

Example

The following example uses the COUNT method on a filter page object that includes a two filter controls for the **Date** system table.

```
var
    varDateItem: Text[30];
    varCount: Integer;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddTable(varDateItem + ' 1',DATABASE::Date);
    varFilterPageBuilder.AddTable(varDateItem + ' 2',DATABASE::Date);
    varCount := varFilterPageBuilder.Count;
    if varCount <> 2 then
      Error('There should be two controls in varFilterPageBuilder');
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.GetView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the filter view (which defines the sort order, key, and filters) for the record in the specified filter control of a filter page. The view contains all fields in the filter control that have a default filter value.

Syntax

```
View :=   FilterPageBuilder.GetView(Name: String [, UseNames: Boolean])
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name

Type: String

The name of the filter control. This value must match the value of the Name parameter that was specified by AddTable, AddRecord, or AddRecordRef method that adds the table to the filter control.

UseNames

Type: Boolean

Specifies whether a field caption or field number should be returned. This parameter is optional. If this parameter is **true** (default value) or if it is empty, then the returned string contains references to field captions in the table with which the record is associated. If this parameter is **false**, then the returned string contains references to field numbers.

Return Value

View Type: String The view that is configured for the filter control that is identified by Name.

Example

The following example initializes a filter page object that includes a filter control for the **Date** system table. The filter control has the caption of **Date record**. The example adds two filter fields to the filter control on the filter page as the result of applying a default view. The GetView method is used to capture that filter view from the FilterPageBuilder object, and then apply it to the record.

```
var
    varDateItem: Text[300];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;
    varDefaultView: Text[300];

begin
    varDateItem := 'Date record';
    varDateRecord.SetFilter("Period End", '12122015D');
    varDateRecord.SetFilter("Period Start", '01012015D');
    varDefaultView := varDateRecord.GetView;
    varFilterPageBuilder.AddTable(varDateItem, DATABASE::Date);
    varFilterPageBuilder.SetView(varDateItem, varDefaultView);
    if varFilterPageBuilder.RunModal = true then
      varDateRecord.SetView(varFilterPageBuilder.GetView(varDateItem));
      varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of a table filter control that is included on a filter page based on an index number that is assigned to the filter control.

Syntax

```
Name :=   FilterPageBuilder.Name(Index: Integer)
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Index

Type: Integer

The index of a filter control. The value must be in the range 1 to N, where N is the number of filter controls on the filter page.

Return Value

Name Type: String The name of the filter control.

Example

The following example initializes a filter page object that includes two filter controls for the **Date** system table. The NAME method returns the names of filter control in a message dialog box.

```
var
    varDateItem: Text[30];
    varCount: Integer;
    varIndex: Integer;
    varFilterPageBuilder: FilterPageBuilder;

begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddTable(varDateItem + ' 1', Database::Date);
    varFilterPageBuilder.AddTable(varDateItem + ' 2', Database::Date);
    varCount := varFilterPageBuilder.COUNT;
    if varCount <> 2 then
        Error('There should be two controls in FilterPageBuilder');
    for varIndex := 1 to varCount do
        Message('Control item %1 is named %2', varIndex, varFilterPageBuilder.Name(varIndex));
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.PageCaption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the FilterPageBuilder UI caption. Defaults to the resource text if not explicitly set.

Syntax

```
[PageCaption := ]  FilterPageBuilder.PageCaption([PageCaption: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

PageCaption

Type: String

The value to set for the FilterPageBuilder UI caption.

Return Value

PageCaption Type: String The current value of the FilterPageBuilder UI caption.

Example

```
var
    varFilterPageBuilder: FilterPageBuilder;
    Customer: Record Customer;
    Item: Record Item;
    varDateItem: Text[30];
begin
    varFilterPageBuilder.AddRecord('Item Table', Item);
    varFilterPageBuilder.Addfield('Item Table', Item."No.", '>100');
    varFilterPageBuilder.PageCaption := 'Item Filter Page';
    varFilterPageBuilder.RunModal;
    Item.SetView(varFilterPageBuilder.Getview('Item Table'));
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

FilterPageBuilder.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Builds and runs the filter page that includes the filter controls that are stored in FilterPageBuilder object instance.

Syntax

```
[Ok := ]  FilterPageBuilder.RunModal()
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The page is run modally and includes an **OK** and **Cancel** button for closing to modal popup.

You can call the GetView Method method to retrieve the current filter view that is configured on the filter control and apply to the record.

Because the filter page runs modally in the context of where it was invoked from, users cannot bookmark a link to this page from the user interface.

Example

The following example initializes a filter page object that includes a filter control for the **Date** system table. The filter control has the caption of **Date record**. The example adds two filter fields to the filter control on the filter page as the result of applying a default view. The GetView Method is used to capture that filter view from the FilterPageBuilder object, and then apply it to the record.

```
var
    varDateItem: Text[300];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;
    varDefaultView: Text[300];

begin
    varDateItem := 'Date record';
    varDateRecord.SetFilter("Period End", '20151212D');
    varDateRecord.SetFilter("Period Start", '20150101D');
    varDefaultView := varDateRecord.GetView;
    varFilterPageBuilder.AddTable(varDateItem, Database::Date);
    varFilterPageBuilder.SetView(varDateItem, varDefaultView);
    if varFilterPageBuilder.RunModal = true then
      varDateRecord.SetView(varFilterPageBuilder.GetView(varDateItem));
end;
```

See Also

FilterPageBuilder Data Type
Getting Started with AL
Developing Extensions

FilterPageBuilder.SetView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current filter view, which defines the sort order, key, and filters, for a record in a filter control on a filter page. The view contains all fields that have default filters, but does not contain fields without filters.

Syntax

```
[Ok := ]  FilterPageBuilder.SetView(Name: String, View: String)
```

Parameters

FilterPageBuilder Type: FilterPageBuilder An instance of the FilterPageBuilder data type.

Name

Type: String

The name that is assigned to the filter control. This value must match the value of the ItemName parameter that was specified by AddTable, AddRecord, or AddRecordRef method that adds the table to the filter control.

View

Type: String

The filter view to apply. This can be the output of the GetView method invoked on a Record or a RecordRef value.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The SetView method will overwrite any previously defined filters for the fields that are also included in the view.

Example

The following example initializes a filter page object that includes a filter control for the **Date** system table. The filter control has the caption of **Date record**. The example adds two filter fields to the filter control on the filter page as the result of applying a default view from the GetView Method.

```
var
    varDateItem: Text[30];
    varDateRecord: Record Date;
    varFilterPageBuilder: FilterPageBuilder;
    varDefaultView: Text;

begin
    varDateItem := 'Date record';
    varDateRecord.SetFilter("Period End", '20151212D');
    varDateRecord.SetFilter("Period Start", '20150101D');
    varDefaultView := varDateRecord.GetView;
    varFilterPageBuilder.AddTable(varDateItem, Database::Date);
    varFilterPageBuilder.SetView(varDateItem, varDefaultView);
    varFilterPageBuilder.RunModal();
end;
```

See Also

FilterPageBuilder Data Type
Creating Filter Pages for Tables
Getting Started with AL
Developing Extensions

Guid Data Type

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a 16 byte binary data type. This data type is used for the global identification of objects, programs, records, and so on. The important property of a GUID is that each value is globally unique. The value is generated by an algorithm, developed by Microsoft, which assures this uniqueness.

The GUID is a 16-byte binary data type that can be logically grouped into the following subgroups:

4byte-2byte-2byte-2byte-6byte.

The standard textual representation is {12345678-1234-1234-1234-1234567890AB}.

The virtual table OLE Control (2000000042) does not use the GUID data type. It uses a textual representation of the GUID in a text field instead. It is easier to make operations and references to this text field using the GUID data type than it is using the textual representation. The GUID data type is compatible with the existing textual representation.

The GUID data type is useful when you want to uniquely identify some data, so that it can be exchanged with external applications. For example, if you want to transfer an item catalog to an external application, you add a GUID field to the record in the table and use this as the primary reference when you communicate with the external application.

Compatibility

You can assign and compare the Text data type and the GUID data type. Assigning a Text to a GUID can be done as follows:

```
MyTableRec.MyGuid  :=  MyTableRec.MyText;
```

The supported formats of `MyText` are:

`'11111111-1111-1111-1111-111111111111'` `'{22222222-2222-2222-2222-222222222222}'`

Methods and properties

The following AL methods can be used with the GUID data type:

```
Guid := CreateGUID();
```

This method creates a new unique GUID value. The value can then be assigned to a field of the GUID data type or of the Text data type.

```
Ok := IsNullGUID(Guid);
```

This method is a convenient way to check if a value has already been assigned to a GUID. A NULL GUID (consisting only of zeroes) is valid, but should never be used for reference purposes.

A NULL GUID is valid but is not useful in a table. Therefore, the **AutoSplitKey** property is implemented for the GUID data type when it is used in a page. When GUID is selected as a primary key, **AutoSplitKey** is enabled for the page, and the GUID value remains NULL. When you create a new record, a valid GUID is created and assigned automatically.

The CreateGUID method and IsNullGUID method methods are available in the AL Symbol Menu under SYSTEM, Variables.

CreateGUID takes no arguments and returns a valid 16-byte GUID value. If the result is assigned to a TEXT variable or field, the value is converted to a string and follows the syntax explained earlier. The algorithm that generates the new GUID value uses Microsoft's CoCreateGuid method.

IsNullGUID takes a GUID value as a required argument and returns True/False depending on whether the GUID value is NULL. This method does not accept a Text value as an argument.

AutoSplitKey is a property, not a method and can be applied to pages. If you have defined a GUID field as part of the primary key, the **AutoSplitKey** property automatically generates a new valid GUID value. When a new record is created and the GUID field is left as NULL, the **AutoSplitKey** property ensures that a valid GUID value is automatically inserted into the field. If you then enter a NULL GUID into this record, for example, by using the Clear method, this new NULL GUID value is not automatically replaced by the **AutoSplitKey** property. The **AutoSplitKey** property only applies to new records.

Format

The GUID value can also be represented as text. You can use the standard AL methods Format and Evaluate to convert from GUID values to Text values. If you do not use the correct format when you edit a GUID value in its textual format, the following error message is displayed:

Invalid Format of GUID string. The correct format of the GUID string is {CDEF7890-ABCD-1234-ABCD-1234567890AB} where 0-9, A-F symbolizes hexadecimal digits.

See Also

Getting Started with AL
Developing Extensions

HttpClient Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides a data type for sending HTTP requests and receiving HTTP responses from a resource identified by a URI.

The following methods are available on instances of the HttpClient data type.

METHOD NAME	DESCRIPTION
AddCertificate(String [, String])	Adds a certificate to the HttpClient class.
Clear()	Sets the HttpClient variable to the default value.
DefaultRequestHeaders()	Gets the default request headers which should be sent with each request.
Delete(String, var HttpResponseMessage)	Sends a DELETE request to delete the resource identified by the request URL.
Get(String, var HttpResponseMessage)	Sends a Get request to get the resource identified by the request URL.
GetBaseAddress()	Gets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.
Post(String, HttpContent, var HttpResponseMessage)	Sends a POST request to the specified URI as an asynchronous operation.
Put(String, HttpContent, var HttpResponseMessage)	Sends a PUT request to the specified URI as an asynchronous operation.
Send(HttpRequestMessage, var HttpResponseMessage)	Sends an HTTP request as an asynchronous operation.
SetBaseAddress(String)	Sets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.
Timeout([Duration])	Gets or sets the duration in milliseconds to wait before the request times out.
UseDefaultNetworkWindowsAuthentication()	Sets the HttpClient credentials to use the default network credentials for Windows authentication. If this method is invoked after any HTTP request has started; a runtime error occurs.
UseWindowsAuthentication(String, String [, String])	Sets the HttpClient credentials to use the specified network credentials for Windows authentication. If this method is invoked after any HTTP request has started; a runtime error occurs.

Remarks

The supported security protocols are controlled by the **SecurityProtocol** configuration setting. For more information, see Microsoft Dynamics 365 Business Central Server Configuration.

See Also

Getting Started with AL
Developing Extensions

HttpClient.AddCertificate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a certificate to the HttpClient class.

Syntax

```
HttpClient.AddCertificate(Certificate: String [, Password: String])
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Certificate
Type: String
The Base64 encoded certificate.
Password
Type: String
The certificate password.

Remarks

The certificate must be in base 64 format.

With the **AddCertificate** method you set the certificates that you want to be associated to the request of the http client connection. These have the only purpose of authenticating the client.

The system caches SSL sessions as they are created and attempts to reuse a cached session for a new request, if possible. When attempting to reuse an SSL session, it uses the first certificate that was added or tries to reuse an anonymous session if no certificates have been specified.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Clear Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the HttpClient variable to the default value.

Syntax

```
HttpClient.Clear()
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.DefaultRequestHeaders Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the default request headers which should be sent with each request.

Syntax

```
CurrentDefaultRequestHeaders :=   HttpClient.DefaultRequestHeaders()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Return Value

CurrentDefaultRequestHeaders Type: HttpHeaders The default request headers which should be sent with each request.

Remarks

The HttpHeaders variable is a reference type. When you add a header to this variable, the default headers are changed. You cannot set another HttpHeaders object as a default header, you have to update the header fetched from HttpClient.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Delete Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a Delete request to delete the resource identified by the request URL.

Syntax

```
[Ok := ]  HttpClient.Delete(Path: String, var Response: HttpResponseMessage)
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Path
Type: String
The path the request is sent to.

Response
Type: HttpResponseMessage
The response received from the remote endpoint.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a Get request to get the resource identified by the request URL.

Syntax

```
[Ok := ]  HttpClient.Get(Path: String, var Response: HttpResponseMessage)
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Path
Type: String
The path the request is sent to.

Response
Type: HttpResponseMessage
The response received from the remote endpoint.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.GetBaseAddress Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.

Syntax

```
CurrentBaseAddress :=   HttpClient.GetBaseAddress()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Return Value

CurrentBaseAddress Type: String The base address of URI of the Internet resource used when sending requests.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Post Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a POST request to the specified URI as an asynchronous operation.

Syntax

```
[Ok := ]  HttpClient.Post(Path: String, Content: HttpContent, var Response: HttpResponseMessage)
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Path
Type: String
The path the request is sent to.
Content
Type: HttpContent
The HTTP request content sent to the server.

Response
Type: HttpResponseMessage
The response received from the remote endpoint.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Put Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends a PUT request to the specified URI as an asynchronous operation.

Syntax

```
[Ok := ]  HttpClient.Put(Path: String, Content: HttpContent, var Response: HttpResponseMessage)
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Path
Type: String
The path the request is sent to.

Content
Type: HttpContent
The HTTP request content sent to the server.

Response
Type: HttpResponseMessage
The response received from the remote endpoint.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Send Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends an HTTP request as an asynchronous operation.

Syntax

```
[Ok := ]  HttpClient.Send(Request: HttpRequestMessage, var Response: HttpResponseMessage)
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Request
Type: HttpRequestMessage
The HTTP request message to send.

Response
Type: HttpResponseMessage
The response received from the remote endpoint.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.SetBaseAddress Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests.

Syntax

```
[Ok := ]  HttpClient.SetBaseAddress(NewBaseAddress: String)
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

NewBaseAddress
Type: String
The base address of the Uniform Resource Identifier (URI) of the Internet resource used when sending requests.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.Timeout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the duration in milliseconds to wait before the request times out.

Syntax

```
[CurrentTimeout := ]  HttpClient.Timeout([SetTimeout: Duration])
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

SetTimeout

Type: Duration

The duration in milliseconds to wait before the request times out.

Return Value

CurrentTimeout Type: Duration The duration in milliseconds to wait before the request times out.

Remarks

The *SetTimeout* duration is limited by the **NavHttpClientMaxTimeout** parameter that is configured for Business Central Server instance. If you set the duration to a value that is greater than the value of the **NavHttpClientMaxTimeout** parameter, a 'NavNclHttpClientTimeoutTooLargeException' error is thrown. The default value of the **NavHttpClientMaxTimeout** parameter is 00:05:00. To change the **NavHttpClientMaxTimeout** parameter, see Configuring Business Central Server.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.UseDefaultNetworkWindowsAuthentication Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Sets the HttpClient credentials to use the default network credentials for Windows authentication. If this method is invoked after any HTTP request has started; a runtime error occurs.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Result := ]  HttpClient.UseDefaultNetworkWindowsAuthentication()
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

Return Value

Result Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpClient.UseWindowsAuthentication Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Sets the HttpClient credentials to use the specified network credentials for Windows authentication. If this method is invoked after any HTTP request has started; a runtime error occurs.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Result := ]  HttpClient.UseWindowsAuthentication(UserName: String, Password: String [, Domain: String])
```

Parameters

HttpClient Type: HttpClient An instance of the HttpClient data type.

UserName
Type: String
The Windows user name.
Password
Type: String
The password.
Domain
Type: String
The user's domain.

Return Value

Result Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpClient Data Type
Getting Started with AL
Developing Extensions

HttpContent Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an HTTP entity body and content headers.

The following methods are available on instances of the HttpContent data type.

METHOD NAME	DESCRIPTION
Clear()	Sets the HttpContent object to a default value. The content contains an empty string and empty headers.
GetHeaders(var HttpHeaders)	Gets the HTTP content headers as defined in RFC 2616.
ReadAs(var Text)	Reads the content into the provided text.
ReadAs(var InStream)	Reads the content into the provided text.
WriteFrom(Text)	Sets HttpContent content to the provided text or stream.
WriteFrom(InStream)	Sets HttpContent content to the provided text or stream.

An instance of HttpContent encapsulates the body and the associated headers of an HTTP request that will be sent to a remote endpoint or that is being received from a remote endpoint. The HttpContent data type is a value type. This means that when assigning an instance of HttpContent to a variable, a copy will be created.

Example

The following example illustrates how to use the HttpContent type to send a simple POST request containing JSON data.

```
codeunit 50110 MyCodeunit
{
    procedure MakeRequest(uri: Text; payload: Text) responseText: Text;
    var
        client: HttpClient;
        request: HttpRequestMessage;
        response: HttpResponseMessage;
        contentHeaders: HttpHeaders;
        content: HttpContent;
    begin
        // Add the payload to the content
        content.WriteFrom(payload);

        // Retrieve the contentHeaders associated with the content
        content.GetHeaders(contentHeaders);
        contentHeaders.Clear();
        contentHeaders.Add('Content-Type', 'application/json');

        // Assigning content to request.Content will actually create a copy of the content and assign it.
        // After this line, modifying the content variable or its associated headers will not reflect in
        // the content associated with the request message
        request.Content := content;

        request.SetRequestUri(uri);
        request.Method := 'POST';

        client.Send(request, response);

        // Read the response content as json.
        response.Content().ReadAs(responseText);
    end;
}
```

See Also

Getting Started with AL
Developing Extensions

HttpContent.Clear Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the HttpContent object to a default value. The content contains an empty string and empty headers.

Syntax

```
HttpContent.Clear()
```

Parameters

HttpContent Type: HttpContent An instance of the HttpContent data type.

See Also

HttpContent Data Type
Getting Started with AL
Developing Extensions

HttpContent.GetHeaders Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the HTTP content headers as defined in RFC 2616.

Syntax

```
[Ok := ]  HttpContent.GetHeaders(var Headers: HttpHeaders)
```

Parameters

HttpContent Type: HttpContent An instance of the HttpContent data type.

Headers
Type: HttpHeaders
The HTTP headers associated with the content.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpContent Data Type
Getting Started with AL
Developing Extensions

HttpContent.ReadAs Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the content into the provided text.

Syntax

```
[Ok := ]  HttpContent.ReadAs(var OutputString: Text)
```

Parameters

HttpContent Type: HttpContent An instance of the HttpContent data type.

OutputString
Type: Text
The variable that will contain the HTTP content as a string.

Return Value

Ok Type: Boolean Accessing the HttpContent property of HttpResponseMessage in a case when the request fails will result in an error. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpContent Data Type
Getting Started with AL
Developing Extensions

HttpContent.ReadAs Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the content into the provided text.

Syntax

```
[Ok := ]  HttpContent.ReadAs(var InStream: InStream)
```

Parameters

HttpContent Type: HttpContent An instance of the HttpContent data type.

InStream
Type: InStream
The InStream variable that will contain the HTTP content stream.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpContent Data Type
Getting Started with AL
Developing Extensions

HttpContent.WriteFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets HttpContent content to the provided text or stream.

Syntax

```
HttpContent.WriteFrom(Text: Text)
```

Parameters

HttpContent Type: HttpContent An instance of the HttpContent data type.

Text

Type: Text

A new HttpContent is constructed with this value and headers from before.

See Also

HttpContent Data Type
Getting Started with AL
Developing Extensions

HttpContent.WriteFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets HttpContent content to the provided text or stream.

Syntax

```
HttpContent.WriteFrom(InStream: InStream)
```

Parameters

HttpContent Type: HttpContent An instance of the HttpContent data type.

InStream

Type: InStream

A new HttpContent is constructed with this value and headers from before.

See Also

HttpContent Data Type
Getting Started with AL
Developing Extensions

HttpHeaders Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a collection of headers and their values.

The following methods are available on instances of the HttpHeaders data type.

METHOD NAME	DESCRIPTION
Add(String, String)	Adds the specified header and its value into the HttpHeaders collection. Validates the provided value.
Clear()	Sets the HttpHeaders variable to the default value.
Contains(String)	Checks if the specified header exists in the HttpHeaders collection.
GetValues(String, Array of [Text])	Gets the values for the specified key.
Remove(String)	Removes the specified header from the HttpHeaders collection.
TryAddWithoutValidation(String, String)	Adds the specified header and its value into the HttpHeaders collection. Doesn't validate the provided value.

See Also

Getting Started with AL
Developing Extensions

HttpHeaders.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified header and its value into the HttpHeaders collection. Validates the provided value.

Syntax

```
[Ok := ]  HttpHeaders.Add(Name: String, Value: String)
```

Parameters

HttpHeaders Type: HttpHeaders An instance of the HttpHeaders data type.

Name
Type: String
The header to add to the collection.

Value
Type: String
The content of the header.

Return Value

Ok Type: Boolean **true** if the value was added successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpHeaders Data Type
Getting Started with AL
Developing Extensions

HttpHeaders.Clear Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the HttpHeaders variable to the default value.

Syntax

```
HttpHeaders.Clear()
```

Parameters

HttpHeaders Type: HttpHeaders An instance of the HttpHeaders data type.

See Also

HttpHeaders Data Type
Getting Started with AL
Developing Extensions

HttpHeaders.Contains Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks if the specified header exists in the HttpHeaders collection.

Syntax

```
Result :=   HttpHeaders.Contains(Name: String)
```

Parameters

HttpHeaders Type: HttpHeaders An instance of the HttpHeaders data type.

Name
Type: String
The specific header.

Return Value

Result Type: Boolean **true** if the specified header exists in the collection; otherwise **false**.

See Also

HttpHeaders Data Type
Getting Started with AL
Developing Extensions

HttpHeaders.GetValues Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the values for the specified key.

Syntax

```
[Ok := ]  HttpHeaders.GetValues(Key: String, Values: Array of [Text])
```

Parameters

HttpHeaders Type: HttpHeaders An instance of the HttpHeaders data type.

Key
Type: String
The specified header.

Values
Type: Text
The specified header values.

Return Value

Ok Type: Boolean **true** if the specified header name and values are stored in the collection; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpHeaders Data Type
Getting Started with AL
Developing Extensions

HttpHeaders.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified header from the HttpHeaders collection.

Syntax

```
[Ok := ]  HttpHeaders.Remove(Name: String)
```

Parameters

HttpHeaders Type: HttpHeaders An instance of the HttpHeaders data type.

Name
Type: String
The name of the header to remove from the collection.

Return Value

Ok Type: Boolean **true** if the element is successfully removed; otherwise, **false**. This method also returns **false** if the given header was not found in the HttpHeaders collection. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpHeaders Data Type
Getting Started with AL
Developing Extensions

HttpHeaders.TryAddWithoutValidation Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Adds the specified header and its value into the HttpHeaders collection. Doesn't validate the provided value.

Syntax

```
[Ok := ]  HttpHeaders.TryAddWithoutValidation(Name: String, Value: String)
```

Parameters

HttpHeaders Type: HttpHeaders An instance of the HttpHeaders data type.

Name
Type: String
The header to add to the collection.
Value
Type: String
The content of the header.

Return Value

Ok Type: Boolean **true** if the value was added successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpHeaders Data Type
Getting Started with AL
Developing Extensions

HttpRequestMessage Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an HTTP request message.

The following methods are available on instances of the HttpRequestMessage data type.

METHOD NAME	DESCRIPTION
Content([HttpContent])	Gets or sets the contents of the HTTP message.
GetHeaders(var HttpHeaders)	Gets a reference to the collection of HTTP request headers.
GetRequestUri()	Gets the URI used for the HTTP request.
Method([String])	Gets or sets the method type as defined in the HTTP standard.
SetRequestUri(String)	Sets the URI used for the HTTP request.

NOTE

For performance reasons all HTTP, JSON, TextBuilder, and XML types are reference types, not value types. Reference types holds a pointer to the data elsewhere in memory, whereas value types store its own data.

See Also

Getting Started with AL
Developing Extensions

HttpRequestMessage.Content Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the contents of the HTTP message.

Syntax

```
[CurrentContent := ]  HttpRequestMessage.Content([SetContent: HttpContent])
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpRequestMessage Type: HttpRequestMessage An instance of the HttpRequestMessage data type.

SetContent

Type: HttpContent

The contents of the HTTP message.

Return Value

CurrentContent Type: HttpContent The contents of the HTTP message.

See Also

HttpRequestMessage Data Type
Getting Started with AL
Developing Extensions

HttpRequestMessage.GetHeaders Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a reference to the collection of HTTP request headers.

Syntax

```
[Ok := ]  HttpRequestMessage.GetHeaders(var Headers: HttpHeaders)
```

Parameters

HttpRequestMessage Type: HttpRequestMessage An instance of the HttpRequestMessage data type.

Headers
Type: HttpHeaders
A variable that will contain a reference to the collection of HTTP request headers after the method completes successfully.

Return Value

Ok Type: Boolean **true** if the operation was completed successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpRequestMessage Data Type
Getting Started with AL
Developing Extensions

HttpRequestMessage.GetRequestUri Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the URI used for the HTTP request.

Syntax

```
RequestUri :=   HttpRequestMessage.GetRequestUri()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpRequestMessage Type: HttpRequestMessage An instance of the HttpRequestMessage data type.

Return Value

RequestUri Type: String The URI used for the HTTP request.

See Also

HttpRequestMessage Data Type
Getting Started with AL
Developing Extensions

HttpRequestMessage.Method Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the method type as defined in the HTTP standard.

Syntax

```
[CurrentMethod := ]  HttpRequestMessage.Method([NewMethod: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

HttpRequestMessage Type: HttpRequestMessage An instance of the HttpRequestMessage data type.

NewMethod
Type: String
The HTTP method used by the request message.

Return Value

CurrentMethod Type: String The HTTP method used by the request message. The default is the Get method.

See Also

HttpRequestMessage Data Type
Getting Started with AL
Developing Extensions

HttpRequestMessage.SetRequestUri Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the URI used for the HTTP request.

Syntax

```
[Ok := ]  HttpRequestMessage.SetRequestUri(RequestUri: String)
```

Parameters

HttpRequestMessage Type: HttpRequestMessage An instance of the HttpRequestMessage data type.

RequestUri
Type: String
The URI to use for the HTTP request.

Return Value

Ok Type: Boolean **true** if the URI was set successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

HttpRequestMessage Data Type
Getting Started with AL
Developing Extensions

HttpResponseMessage Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a HTTP response message including the status code and data.

The following methods are available on instances of the HttpResponseMessage data type.

METHOD NAME	DESCRIPTION
Content()	Gets the contents of the HTTP response.
Headers()	Gets the HTTP response's HTTP headers.
HttpStatusCode()	Gets the status code of the HTTP response.
IsBlockedByEnvironment()	Gets a value that indicates if the HTTP response is the result of the environment blocking an outgoing HTTP request.
IsSuccessStatusCode()	Gets a value that indicates if the HTTP response was successful.
ReasonPhrase()	Gets the reason phrase which typically is sent by servers together with the status code.

Remarks

The size of the `HttpResponseMessage` is determined by the `HttpClient AL Function Response Size` setting on the Dynamics 365 Business Central server. The default value is `150`. For more information, see Microsoft Dynamics 365 Business Central Server Configuration.

> **NOTE**
> For performance reasons all HTTP, JSON, TextBuilder, and XML types are reference types, not value types. Reference types holds a pointer to the data elsewhere in memory, whereas value types store its own data.

See Also

Getting Started with AL
Developing Extensions

HttpResponseMessage.Content Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the contents of the HTTP response.

Syntax

```
Content :=   HttpResponseMessage.Content()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpResponseMessage Type: HttpResponseMessage An instance of the HttpResponseMessage data type.

Return Value

Content Type: HttpContent The contents of the HTTP response.

See Also

HttpResponseMessage Data Type
Getting Started with AL
Developing Extensions

HttpResponseMessage.Headers Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the HTTP response's HTTP headers.

Syntax

```
Headers :=   HttpResponseMessage.Headers()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpResponseMessage Type: HttpResponseMessage An instance of the HttpResponseMessage data type.

Return Value

Headers Type: HttpHeaders The HTTP headers.

See Also

HttpResponseMessage Data Type
Getting Started with AL
Developing Extensions

HttpResponseMessage.HttpStatusCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the status code of the HTTP response.

Syntax

```
StatusCode :=   HttpResponseMessage.HttpStatusCode()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

HttpResponseMessage Type: HttpResponseMessage An instance of the HttpResponseMessage data type.

Return Value

StatusCode Type: Integer The status code of the HTTP response.

See Also

HttpResponseMessage Data Type
Getting Started with AL
Developing Extensions

HttpResponseMessage.IsBlockedByEnvironment Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets a value that indicates if the HTTP response is the result of the environment blocking an outgoing HTTP request.

Syntax

```
IsBlockedByEnvironment :=   HttpResponseMessage.IsBlockedByEnvironment()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpResponseMessage Type: HttpResponseMessage An instance of the HttpResponseMessage data type.

Return Value

IsBlockedByEnvironment Type: Boolean **true** if the HTTP response is the result of the environment blocking an outgoing HTTP request, otherwise **false**.

See Also

HttpResponseMessage Data Type
Getting Started with AL
Developing Extensions

HttpResponseMessage.IsSuccessStatusCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value that indicates if the HTTP response was successful.

Syntax

```
IsSuccessStatusCode :=   HttpResponseMessage.IsSuccessStatusCode()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpResponseMessage Type: HttpResponseMessage An instance of the HttpResponseMessage data type.

Return Value

IsSuccessStatusCode Type: Boolean A value that indicates if the HTTP response was successful. **true** if StatusCode was in the range 200-299; otherwise **false**.

See Also

HttpResponseMessage Data Type
Getting Started with AL
Developing Extensions

HttpResponseMessage.ReasonPhrase Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the reason phrase which typically is sent by servers together with the status code.

Syntax

```
ReasonPhrase :=   HttpResponseMessage.ReasonPhrase()
```

NOTE

This method can be invoked using property access syntax.

Parameters

HttpResponseMessage Type: HttpResponseMessage An instance of the HttpResponseMessage data type.

Return Value

ReasonPhrase Type: String The reason phrase sent by the server.

See Also

HttpResponseMessage Data Type
Getting Started with AL
Developing Extensions

InStream Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a generic stream object that you can use to read from or write to files and BLOBs. You can define the internal structure of a stream as a flat stream of bytes. You can assign one stream to another. Reading from and writing to a stream occurs sequentially.

The following methods are available on instances of the InStream data type.

METHOD NAME	DESCRIPTION
EOS()	Indicates whether an input stream has reached End of Stream (EOS).
Read(var Boolean [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var Byte [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var Char [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var Integer [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var BigInteger [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var Decimal [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var Guid [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var String [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
Read(var Any [, Integer])	Reads a specified number of bytes from an InStream object. Data is read in binary format.
ReadText(var Text [, Integer])	Reads text from an InStream object.

See Also

Getting Started with AL
Developing Extensions

InStream.EOS Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an input stream has reached End of Stream (EOS).

Syntax

```
IsEOS :=   InStream.EOS()
```

NOTE

This method can be invoked using property access syntax.

Parameters

InStream Type: InStream An instance of the InStream data type.

Return Value

IsEOS Type: Boolean **true** if the stream has reached End of Stream; otherwise **false**.

Remarks

If you are reading data from an external component, EOS can return **false** even though no more data is available. This may occur if you have not called Read first.

Example

The following example opens the text file in a folder that is named MyFolder. The data in the text file is read into and input stream variable named StreamInTest. The InStream.EOS Method is used to determine whether the input stream has reached the end. If the stream has not reached the end, the stream is read into a text buffer, which indicates that the stream has not reached the end until the stream reaches the end. You must also create the following file 'c:\MyFolder\MyText.txt'.

```
 var
    StreamInTest: InStream;
    FileTest: File;
    Buffer: Text;
begin
    FileTest.Open('c:\MyFolder\MyText.txt');
    FileTest.CreateInStream(StreamInTest);
    while not StreamInTest.EOS do begin
      StreamInTest.ReadTet(Buffer);
      //Do some processing
      Message('Stream is still processing')
    end;
    Message('End of Stream');
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Boolean [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Boolean

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see WRITE, WriteText, Read, and ReadTEXT Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Byte [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Byte

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Char [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Char

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see WRITE, WriteText, Read, and ReadTEXT Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Integer [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Integer

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: BigInteger [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: BigInteger

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see WRITE, WriteText, Read, and ReadTEXT Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Decimal [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Decimal

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see WRITE, WriteText, Read, and ReadTEXT Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Guid [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Guid

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: String [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: String

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see WRITE, WriteText, Read, and ReadTEXT Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads a specified number of bytes from an InStream object. Data is read in binary format.

Syntax

```
[Read := ]  InStream.Read(var Variable: Any [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Any

Length
Type: Integer
Describes the number of characters to be read. If you do not specify Length, the size of the variable is used. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, you receive an error message.

Return Value

Read Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Read reads until the specified length or a zero byte. For more information about how zero bytes and line endings are read, see WRITE, WriteText, Read, and ReadTEXT Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value (*Read*) is not present and the data being read is less than the length requested to be read, then you receive an error message.

If the return value is present, then you must verify the validity of the data that has been read.

Example

The following example shows how to use the **Instream.Read** method to read data in binary format. The **Find** method finds the first record from the **Company Information** table. The **CalcFields** method retrieves the **Picture** field, which is a BLOB field. The **CreateInStream** method uses the recBinaries variable to create an InStream object that is named varInstream. The **varInstream.Read** method then reads three characters from the varInstream variable and stores the binary data in the varChars variable. The number of characters that is read is stored in the numChars variable. The binary data and the number of characters that is read are displayed in a message box.

```
 var
    recBinaries: Record "Company Information";
    varInstream: Instream;
    varChars: Text[50];
    numChars: Integer;
    Text000: Label 'Number of characters read: %1. Characters read: %2.';
begin
    recBinaries.Find('-');
    recBinaries.CalcFields(recBinaries.Picture);
    recBinaries.Picture.CreateInStream(varInstream);
    numChars := varInstream.Read(varChars,3);
    Message(Text000, numChars, varChars);
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

InStream.ReadText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads text from an InStream object.

Syntax

```
[Read := ]  InStream.ReadText(var Variable: Text [, Length: Integer])
```

Parameters

InStream Type: InStream An instance of the InStream data type.

Variable
Type: Text
The variable that receives the characters that were read.
Length
Type: Integer
The number of characters to be read. If you do not specify this parameter, the maximum length of the string is used.

Return Value

Read Type: Integer The number of characters that were read. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

ReadText reads the until the specified number of bytes, the maximum length of the string, a zero byte, or until the end of the line. For more information about how zero bytes and line endings are read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

Data is read in text format.

If you do not use the optional return value and the data being read is less than the length requested to be read, an error message is displayed.

If you use the return value, you must verify the validity of the data that has been read.

Example

```
 var
    FileTest: File;
    StreamInTest: Instream;
    Txt: Text;
    Int: Integer;
begin
    FileTest.Open('c:\XMLDocs\NewTest.txt');
    FileTest.CreateInStream(StreamInTest);
    // Starting a loop
    while not (StreamInTest.EOS) do begin
      Int := StreamInTest.ReadText(Txt,100);
      Message(Txt + '\Size: ' + Format(Int));
    end;
    FileTest.Close();
end;
```

See Also

InStream Data Type
Getting Started with AL
Developing Extensions

Integer Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stores whole numbers with values that range from -2,147,483,647 to 2,147,483,647.

Remarks

In addition to representing whole numbers in this range, you can use integers to represent Boolean values. For Boolean values, 1 represents **true** and 0 represents **false**.

If you assign -2,147,483,648 directly to an Integer variable, then you get an error when you try to compile the code. However, you can indirectly assign -2,147,483,648 to an Integer variable by using the following code.

```
IntegerVar := -2147483647;
IntegerVar := IntegerVar - 1;
```

If you try to indirectly assign a value that is smaller than -2,147,483,648 or larger than 2,147,483,647, then you get a run-time error.

Example 1

The following are examples of integer values.

```
546
-3425
```

Example 2

The following example is a decimal and not an integer.

```
342.45
```

See Also

Getting Started with AL
Developing Extensions

IsolatedStorage Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Provides data isolation for extensions.

The following methods are available on the IsolatedStorage data type.

METHOD NAME	DESCRIPTION
Contains(String [, DataScope])	Determines whether the storage contains a value with the specified key.
Delete(String [, DataScope])	Deletes the value with the specified key from the isolated storage.
Get(String [, DataScope], var Text)	Gets the value associated with the specified key.
Get(String, var Text)	Gets the value associated with the specified key.
Set(String, String [, DataScope])	Sets the value associated with the specified key.
SetEncrypted(String, String [, DataScope])	Encrypts and sets the value associated with the specified key. The input string cannot exceed a length of 215 plain characters; be aware that special characters take up more space.

See Also

Getting Started with AL
Developing Extensions

IsolatedStorage.Contains Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Determines whether the storage contains a value with the specified key.

Syntax

```
HasValue :=   IsolatedStorage.Contains(Key: String [, DataScope: DataScope])
```

Parameters

Key

Type: String

The key to locate in the storage.

DataScope

Type: DataScope

The scope in which to check for the existence of a value with the given key. If a value is not passed in, the default value DataScope::Module will be used.

Return Value

HasValue Type: Boolean **true** if a value with the specified key exists in the storage, otherwise **false**.

See Also

IsolatedStorage Data Type
Getting Started with AL
Developing Extensions

IsolatedStorage.Delete Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Deletes the value with the specified key from the isolated storage.

Syntax

```
[Ok := ]  IsolatedStorage.Delete(Key: String [, DataScope: DataScope])
```

Parameters

Key
Type: String
The key of the value to remove.

DataScope
Type: DataScope
The scope from which to remove the value with the given key. If a value is not passed in, the default value DataScope::Module will be used.

Return Value

Ok Type: Boolean **true** if the value with the given key was successfully deleted from isolated storage, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

IsolatedStorage Data Type
Getting Started with AL
Developing Extensions

IsolatedStorage.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the value associated with the specified key.

Syntax

```
[Ok := ]  IsolatedStorage.Get(Key: String [, DataScope: DataScope], var Value: Text)
```

Parameters

Key

Type: String

The key of the value to get. If the specified key is not found an error will be reported.

DataScope

Type: DataScope

The scope of the data to retrieve. If a value is not passed in, the default value DataScope::Module will be used.

Value

Type: Text

The value that is associated with the specified key.

Return Value

Ok Type: Boolean **true** if the value was retrieved successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

IsolatedStorage Data Type
Getting Started with AL
Developing Extensions

IsolatedStorage.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the value associated with the specified key.

Syntax

```
[Ok := ]  IsolatedStorage.Get(Key: String, var Value: Text)
```

Parameters

Key

Type: String

The key of the value to get. If the specified key is not found an error will be reported.

Value

Type: Text

The value that is associated with the specified key.

Return Value

Ok Type: Boolean **true** if the value was retrieved successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

IsolatedStorage Data Type
Getting Started with AL
Developing Extensions

IsolatedStorage.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Sets the value associated with the specified key.

Syntax

```
[Ok := ]  IsolatedStorage.Set(Key: String, Value: String [, DataScope: DataScope])
```

Parameters

Key
Type: String
The key of the value to set.

Value
Type: String
The value that will be associated with the specified key.

DataScope
Type: DataScope
The scope of the stored data. If a value is not passed in, the default value DataScope::Module will be used.

Return Value

Ok Type: Boolean **true** if the value was saved successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The **Set** method initiates a write transaction, which means that it cannot be succeeded by code that opens a modal page. If you want to open a modal page, it must be done before the **Set** method is called.

See Also

IsolatedStorage Data Type
Getting Started with AL
Developing Extensions

IsolatedStorage.SetEncrypted Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Encrypts and sets the value associated with the specified key. The input string cannot exceed a length of 215 plain characters; be aware that special characters take up more space.

Syntax

```
[Ok := ]  IsolatedStorage.SetEncrypted(Key: String, Value: String [, DataScope: DataScope])
```

Parameters

Key

Type: String

The key of the value to set.

Value

Type: String

The value that will be associated with the specified key.

DataScope

Type: DataScope

The scope of the stored data. If a value is not passed in, the default value DataScope::Module will be used.

Return Value

Ok Type: Boolean **true** if the value was saved successfully, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

IsolatedStorage Data Type
Getting Started with AL
Developing Extensions

Any Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

This data type can be substituted by any other data type.

> **NOTE**
> The Any Data type cannot be used for declaring constructs in AL. Any is a type that is used for the parameters or return type of methods in the platform.

See Also

Getting Started with AL
Developing Extensions

JsonArray Data Type

2/17/2021 • 5 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a container for any well-formed JSON array. A default JsonArray contains an empty JSON array.

The following methods are available on instances of the JsonArray data type.

METHOD NAME	DESCRIPTION
Add(JsonToken)	Adds a new value at the end of the JsonArray.
Add(JsonArray)	Adds a new value at the end of the JsonArray.
Add(JsonObject)	Adds a new value at the end of the JsonArray.
Add(JsonValue)	Adds a new value at the end of the JsonArray.
Add(Boolean)	Adds a new value at the end of the JsonArray.
Add(Char)	Adds a new value at the end of the JsonArray.
Add(Byte)	Adds a new value at the end of the JsonArray.
Add(Option)	Adds a new value at the end of the JsonArray.
Add(Integer)	Adds a new value at the end of the JsonArray.
Add(BigInteger)	Adds a new value at the end of the JsonArray.
Add(Decimal)	Adds a new value at the end of the JsonArray.
Add(Duration)	Adds a new value at the end of the JsonArray.
Add(Date)	Adds a new value at the end of the JsonArray.
Add(Time)	Adds a new value at the end of the JsonArray.
Add(DateTime)	Adds a new value at the end of the JsonArray.
Add(String)	Adds a new value at the end of the JsonArray.
AsToken()	Converts the value in a JsonArray to a JsonToken data type.
Clone()	Creates a deep-copy of the JsonArray value.
Count()	Gets the number of elements in the JsonArray.
Get(Integer, var JsonToken)	Retrieves the value at the given index in the JsonArray.
IndexOf(JsonToken)	Determines the index of a specific value in the JsonArray.
IndexOf(JsonArray)	Determines the index of a specific value in the JsonArray.
IndexOf(JsonObject)	Determines the index of a specific value in the JsonArray.

METHOD NAME	DESCRIPTION
IndexOf(JsonValue)	Determines the index of a specific value in the JsonArray.
IndexOf(Boolean)	Determines the index of a specific value in the JsonArray.
IndexOf(Char)	Determines the index of a specific value in the JsonArray.
IndexOf(Byte)	Determines the index of a specific value in the JsonArray.
IndexOf(Option)	Determines the index of a specific value in the JsonArray.
IndexOf(Integer)	Determines the index of a specific value in the JsonArray.
IndexOf(BigInteger)	Determines the index of a specific value in the JsonArray.
IndexOf(Decimal)	Determines the index of a specific value in the JsonArray.
IndexOf(Duration)	Determines the index of a specific value in the JsonArray.
IndexOf(Date)	Determines the index of a specific value in the JsonArray.
IndexOf(Time)	Determines the index of a specific value in the JsonArray.
IndexOf(DateTime)	Determines the index of a specific value in the JsonArray.
IndexOf(String)	Determines the index of a specific value in the JsonArray.
Insert(Integer, JsonToken)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, JsonArray)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, JsonObject)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, JsonValue)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Boolean)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Char)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Byte)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Option)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Integer)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, BigInteger)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Decimal)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Duration)	Inserts the value at the given index in the array while shifting all the values to the right by one position.

METHOD NAME	DESCRIPTION
Insert(Integer, Date)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, Time)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, DateTime)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Insert(Integer, String)	Inserts the value at the given index in the array while shifting all the values to the right by one position.
Path()	Retrieves the JSON path of the array relative to the root of its containing tree.
ReadFrom(String)	Reads the JSON data from the string into a JsonArray variable.
ReadFrom(InStream)	Reads the JSON data from the stream into a JsonArray variable.
RemoveAt(Integer)	Removes the token at the given index.
SelectToken(String, var JsonToken)	Selects a JsonToken using a JPath expression.
Set(Integer, JsonToken)	Replaces the value at the given index with a new value.
Set(Integer, JsonObject)	Replaces the value at the given index with a new value.
Set(Integer, JsonArray)	Replaces the value at the given index with a new value.
Set(Integer, JsonValue)	Replaces the value at the given index with a new value.
Set(Integer, Boolean)	Replaces the value at the given index with a new value.
Set(Integer, Char)	Replaces the value at the given index with a new value.
Set(Integer, Byte)	Replaces the value at the given index with a new value.
Set(Integer, Option)	Replaces the value at the given index with a new value.
Set(Integer, Integer)	Replaces the value at the given index with a new value.
Set(Integer, BigInteger)	Replaces the value at the given index with a new value.
Set(Integer, Decimal)	Replaces the value at the given index with a new value.
Set(Integer, Duration)	Replaces the value at the given index with a new value.
Set(Integer, Date)	Replaces the value at the given index with a new value.
Set(Integer, Time)	Replaces the value at the given index with a new value.
Set(Integer, DateTime)	Replaces the value at the given index with a new value.
Set(Integer, String)	Replaces the value at the given index with a new value.
WriteTo(var Text)	Serializes and writes the JSON data of the JsonArray to a given Text object.

METHOD NAME	DESCRIPTION
WriteTo(OutStream)	Serializes and writes the JSON data of the JsonArray to a given OutStream object.

> **NOTE**
> For performance reasons all HTTP, JSON, TextBuilder, and XML types are reference types, not value types. Reference types holds a pointer to the data elsewhere in memory, whereas value types store its own data.

See Also

Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: JsonToken)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonToken

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: JsonArray)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonArray

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: JsonObject)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonObject

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: JsonValue)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonValue

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Boolean)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Boolean

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Char)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Char

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Byte)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Byte

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Option)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Option

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Integer)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Integer

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: BigInteger)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: BigInteger

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Decimal)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Decimal

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Duration)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Duration

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Date)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Date

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: Time)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Time

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: DateTime)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: DateTime

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new value at the end of the JsonArray.

Syntax

```
JsonArray.Add(Value: String)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: String

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.AsToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonArray to a JsonToken data type.

Syntax

```
Token :=   JsonArray.AsToken()
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Return Value

Token Type: JsonToken

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Clone Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a deep-copy of the JsonArray value.

Syntax

```
Clone :=   JsonArray.Clone()
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Return Value

Clone Type: JsonToken The Result will be a full, deep-copy of the Value.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of elements in the JsonArray.

Syntax

```
Count :=   JsonArray.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Return Value

Count Type: Integer The number of elements in the JsonArray.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the value at the given index in the JsonArray.

Syntax

```
[Ok := ]  JsonArray.Get(Index: Integer, var Result: JsonToken)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer
The index of the element in the JsonArray that you want to retrieve.
Result
Type: JsonToken
A variable of type JsonToken that will contain the result if the operation is successful.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the Index is smaller than 0 or greater or equal than JsonArray.Count.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: JsonToken)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonToken

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: JsonArray)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonArray

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: JsonObject)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonObject

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: JsonValue)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: JsonValue

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Boolean)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Boolean

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Char)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Char

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Byte)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Byte

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Option)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Option

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Integer)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Integer

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: BigInteger)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: BigInteger

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Decimal)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Decimal

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Duration)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Duration

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Date)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Date

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: Time)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: Time

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: DateTime)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: DateTime

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the index of a specific value in the JsonArray.

Syntax

```
Index :=   JsonArray.IndexOf(Value: String)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Value
Type: String

Return Value

Index Type: Integer The position of the value in the JsonArray. -1 will be returned if Value cannot be found in the array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: JsonToken)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: JsonToken

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: JsonArray)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: JsonArray

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: JsonObject)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: JsonObject

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: JsonValue)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: JsonValue

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Boolean)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Boolean

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Char)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Char

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Byte)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Byte

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Option)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Option

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Integer)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Integer

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: BigInteger)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: BigInteger

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the Index is smaller than 0 or greater or equal than JsonArray.Count.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Decimal)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Decimal

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Duration)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Duration

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Date)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Date

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: Time)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: Time

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: DateTime)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: DateTime

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts the value at the given index in the array while shifting all the values to the right by one position.

Syntax

```
[Ok := ]  JsonArray.Insert(Index: Integer, Value: String)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Value
Type: String

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Path Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the JSON path of the array relative to the root of its containing tree.

Syntax

```
Path :=   JsonArray.Path()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Return Value

Path Type: String The path of the array relative to its containing JSON tree. If the object is the root of the JSON tree, the path will be empty.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the string into a JsonArray variable.

Syntax

```
[Ok := ]  JsonArray.ReadFrom(String: String)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

String
Type: String
The String object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the JSON data is malformed. If the operation succeeds, the JsonArray will be disconnected from its current JSON tree and the data contained by the JsonArray will be replaced with the new value. To delete the contents in a JsonArray variable use the Clear function.

```
Clear(JsonArray)
```

Example

This example shows how to read JSON data from a string into a JsonArray variable.

```
local procedure ReadJson(data : Text) result : JsonArray;
begin
    result.ReadFrom(data);
end;
```

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the stream into a JsonArray variable.

Syntax

```
[Ok := ]  JsonArray.ReadFrom(Data: InStream)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Data
Type: InStream
The InStream object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the stream is in an invalid state or if the JSON data is malformed. If the operation succeeds, the JsonArray will be disconnected from its current JSON tree and the data contained by the JsonArray will be replaced with the new value. To delete the contents in a JsonArray variable use the Clear function.

```
Clear(JsonArray)
```

Example

This example shows how to read JSON data from a stream into a JsonArray variable.

```
local procedure ReadJson(source : InStream) result : JsonArray;
begin
    result.ReadFrom(source);
end;
```

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.RemoveAt Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the token at the given index.

Syntax

```
[Ok := ]  JsonArray.RemoveAt(Index: Integer)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer
The position of the element that will be removed.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

1. The operation will fail if the Index is smaller than 0 or (greater or equal) than JsonArray.Count.
2. Objects of type JsonArray represent a 0-based array.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.SelectToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a JsonToken using a JPath expression.

Syntax

```
[Ok := ]  JsonArray.SelectToken(Path: String, var Result: JsonToken)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Path
Type: String
A valid JPath expression.

Result
Type: JsonToken
A **JsonToken** variable that will contain the result if the operation is successful.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if more or less than 1 tokens are the result of evaluating the JPath.

Example

The following example shows how to select a value from a complex JSON Object. We build up a select expression in the query variable, we select the token corresponding to the salary of the employee with the given employeeId, and finally, we convert the token to a Decimal value.

We assume that the company token contains JSON data similar to the one below.

```
{ "company": {
    "employees": [
      { "id": "Marcy",
        "salary": 8.95
      },
      { "id": "John",
        "salary": 7
      },
      { "id": "Diana",
        "salary": 10.95
      }
    ]
  }
}
```

```
local procedure SelectEmployeeSalary(companyData : JsonToken; employeeId : Text) salary : Decimal;
var
    query : Text;
    salaryToken : JsonToken;
begin
    query := '$.company.employees[?(@.id=='''+employeeId+''')].salary';
    companyData.SelectToken(query, salaryToken);

    salary := salaryToken.AsValue().AsDecimal();
end;
```

NOTE

Ensure that the selected expression contains ' (single quotation mark) and not " (double quotation mark) to decorate the string value.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: JsonToken)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: JsonToken

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: JsonObject)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: JsonObject

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: JsonArray)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: JsonArray

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: JsonValue)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: JsonValue

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Boolean)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Boolean

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Char)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Char

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Byte)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Byte

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Option)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Option

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Integer)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Integer

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: BigInteger)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: BigInteger

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Decimal)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Decimal

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Duration)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Duration

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Date)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Date

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: Time)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: Time

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: DateTime)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: DateTime

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value at the given index with a new value.

Syntax

```
[Ok := ]  JsonArray.Set(Index: Integer, Result: String)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

Index
Type: Integer

Result
Type: String

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonArray to a given Text object.

Syntax

```
[Ok := ]  JsonArray.WriteTo(var String: Text)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

String
Type: Text
The Text object to which the JSON data will be written.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonArray.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonArray to a given OutStream object.

Syntax

```
[Ok := ]  JsonArray.WriteTo(OutStream: OutStream)
```

Parameters

JsonArray Type: JsonArray An instance of the JsonArray data type.

OutStream
Type: OutStream
The OutStream object to which the JSON data will be written.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonArray Data Type
Getting Started with AL
Developing Extensions

JsonObject Data Type

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a container for any well-formed JSON object. A default JsonObject contains an empty JSON object.

The following methods are available on instances of the JsonObject data type.

METHOD NAME	DESCRIPTION
Add(String, JsonToken)	Adds a new property to a JsonObject.
Add(String, JsonObject)	Adds a new property to a JsonObject.
Add(String, JsonValue)	Adds a new property to a JsonObject.
Add(String, JsonArray)	Adds a new property to a JsonObject.
Add(String, Boolean)	Adds a new property to a JsonObject.
Add(String, Char)	Adds a new property to a JsonObject.
Add(String, Byte)	Adds a new property to a JsonObject.
Add(String, Option)	Adds a new property to a JsonObject.
Add(String, Integer)	Adds a new property to a JsonObject.
Add(String, BigInteger)	Adds a new property to a JsonObject.
Add(String, Decimal)	Adds a new property to a JsonObject.
Add(String, Duration)	Adds a new property to a JsonObject.
Add(String, String)	Adds a new property to a JsonObject.
Add(String, Date)	Adds a new property to a JsonObject.
Add(String, Time)	Adds a new property to a JsonObject.
Add(String, DateTime)	Adds a new property to a JsonObject.
AsToken()	Converts the value in a JsonObject to a JsonToken data type.
Clone()	Creates a deep-copy of the JsonToken value.
Contains(String)	Verifies if a JsonObject contains a property with a given key.
Get(String, var JsonToken)	Retrieves the value of a property with a given key from a JsonObject.

METHOD NAME	DESCRIPTION
Keys()	Gets a set of keys of the JsonObject.
Path()	Retrieves the JSON path of the object relative to the root of its containing tree.
ReadFrom(String)	Reads the JSON data from the string into a JsonObject variable.
ReadFrom(InStream)	Reads the JSON data from the stream into a JsonObject variable.
Remove(String)	Removes the property with the given key from the object.
Replace(String, JsonToken)	Replaces the value of the property with the given key with the new value.
Replace(String, JsonArray)	Replaces the value of the property with the given key with the new value.
Replace(String, JsonObject)	Replaces the value of the property with the given key with the new value.
Replace(String, JsonValue)	Replaces the value of the property with the given key with the new value.
Replace(String, Boolean)	Replaces the value of the property with the given key with the new value.
Replace(String, Char)	Replaces the value of the property with the given key with the new value.
Replace(String, Byte)	Replaces the value of the property with the given key with the new value.
Replace(String, Integer)	Replaces the value of the property with the given key with the new value.
Replace(String, Option)	Replaces the value of the property with the given key with the new value.
Replace(String, BigInteger)	Replaces the value of the property with the given key with the new value.
Replace(String, Decimal)	Replaces the value of the property with the given key with the new value.
Replace(String, Duration)	Replaces the value of the property with the given key with the new value.
Replace(String, Date)	Replaces the value of the property with the given key with the new value.
Replace(String, Time)	Replaces the value of the property with the given key with the new value.
Replace(String, DateTime)	Replaces the value of the property with the given key with the new value.

METHOD NAME	DESCRIPTION
Replace(String, String)	Replaces the value of the property with the given key with the new value.
SelectToken(String, var JsonToken)	Selects a JsonToken using a JPath expression.
Values()	Gets a set of values of the JsonObject.
WriteTo(var Text)	Serializes and writes the JSON data of the JsonObject to a given Text object.
WriteTo(OutStream)	Serializes and writes the JSON data of the JsonObject to a given OutStream object.

NOTE

For performance reasons all HTTP, JSON, TextBuilder, and XML types are reference types, not value types. Reference types holds a pointer to the data elsewhere in memory, whereas value types store its own data.

Remarks

An unitialized variable of JsonObject type represents an empty JSON object. Given a value of JsonObject type, you can check if it is empty by checking that the number of keys in the object is 0.

```
jsonObject.Keys.Count =  0
```

See Also

Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: JsonToken)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonToken

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: JsonObject)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonObject

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: JsonValue)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonValue

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: JsonArray)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonArray

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Boolean)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Boolean

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Char)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Char

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Byte)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Byte

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Option)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Option

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Integer)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Integer

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: BigInteger)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: BigInteger

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Decimal)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Decimal

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Duration)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Duration

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: String)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: String

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Date)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Date

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: Time)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Time

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a new property to a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Add(Key: String, Value: DateTime)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: DateTime

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object already contains a property with the given key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.AsToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonObject to a JsonToken data type.

Syntax

```
Token :=   JsonObject.AsToken()
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Return Value

Token Type: JsonToken The returned JsonToken contains the same data as the JsonObject, but allows for treating the data in a generic manner.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Clone Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a deep-copy of the JsonToken value.

Syntax

```
Clone :=   JsonObject.Clone()
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Return Value

Clone Type: JsonToken The Result will be a full, deep-copy of the Value.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Contains Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Verifies if a JsonObject contains a property with a given key.

Syntax

```
Ok :=   JsonObject.Contains(Key: String)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Return Value

Ok Type: Boolean **true** if the object contains a property with the given key; otherwise, **false**.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the value of a property with a given key from a JsonObject.

Syntax

```
[Ok := ]  JsonObject.Get(Key: String, var Result: JsonToken)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Result
Type: JsonToken
A variable of type JsonToken that will contain the result if the operation is successful.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if the object does not contain a property with the given Key.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Keys Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a set of keys of the JsonObject.

Syntax

```
Keys :=   JsonObject.Keys()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Return Value

Keys Type: List of [Text]

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Path Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the JSON path of the object relative to the root of its containing tree.

Syntax

```
Path :=   JsonObject.Path()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Return Value

Path Type: String The path of the object relative to its containing JSON tree. If the object is the root of the JSON tree, the path will be empty.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the string into a JsonObject variable.

Syntax

```
[Ok := ]  JsonObject.ReadFrom(String: String)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

String
  Type: String
The String object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the JSON data is malformed. If the operation succeeds, the JsonObject will be disconnected from its current JSON tree and the data contained by the JsonObject will be replaced with the new value. To delete the contents in a JsonObject variable use the Clear function.

```
Clear(JsonObject)
```

Example

This example shows how to read JSON data from a string into a JsonObject variable.

```
local procedure ReadJson(data : Text) result : JsonObject;
begin
    result.ReadFrom(data);
end;
```

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the stream into a JsonObject variable.

Syntax

```
[Ok := ]  JsonObject.ReadFrom(InStream: InStream)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

InStream
Type: InStream
The InStream object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the stream is in an invalid state or if the JSON data is malformed. If the operation succeeds, the JsonObject will be disconnected from its current JSON tree and the data contained by the JsonObject will be replaced with the new value. To delete the contents in a JsonObject variable use the Clear function.

```
Clear(JsonObject)
```

Example

This example shows how to read JSON data from a stream into a JsonObject variable.

```
local procedure ReadJson(source : InStream) result : JsonObject;
begin
    result.ReadFrom(source);
end;
```

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the property with the given key from the object.

Syntax

```
[Ok := ]  JsonObject.Remove(Key: String)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: JsonToken)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonToken

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: JsonArray)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonArray

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: JsonObject)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonObject

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: JsonValue)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: JsonValue

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Boolean)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Boolean

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Char)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Char

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Byte)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Byte

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Integer)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Integer

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Option)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Option

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: BigInteger)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: BigInteger

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Decimal)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Decimal

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Duration)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Duration

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Date)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Date

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: Time)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: Time

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: DateTime)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: DateTime

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the value of the property with the given key with the new value.

Syntax

```
[Ok := ]  JsonObject.Replace(Key: String, Value: String)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Key
Type: String

Value
Type: String

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.SelectToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a JsonToken using a JPath expression.

Syntax

```
[Ok := ]  JsonObject.SelectToken(Path: String, var Result: JsonToken)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Path
Type: String
A valid JPath expression.

Result
Type: JsonToken
A JsonToken variable that will contain the result if the operation is successful.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if more or less than 1 tokens are the result of evaluating the JPath.

Example

The following example shows how to select a value from a complex JSON Object. We build up a select expression in the query variable, we select the token corresponding to the salary of the employee with the given employeeId, and finally, we convert the token to a Decimal value.

We assume that the company token contains JSON data similar to the one below.

```
{ "company": {
    "employees": [
      { "id": "Marcy",
        "salary": 8.95
      },
      { "id": "John",
        "salary": 7
      },
      { "id": "Diana",
        "salary": 10.95
      }
    ]
  }
}
```

```
local procedure SelectEmployeeSalary(companyData : JsonToken; employeeId : Text) salary : Decimal;
var
    query : Text;
    salaryToken : JsonToken;
begin
    query := '$.company.employees[?(@.id=='''+employeeId+''')].salary';
    companyData.SelectToken(query, salaryToken);

    salary := salaryToken.AsValue().AsDecimal();
end;
```

NOTE

Ensure that the selected expression contains ' (single quotation mark) and not " (double quotation mark) to decorate the string value.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.Values Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a set of values of the JsonObject.

Syntax

```
Values :=   JsonObject.Values()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

Return Value

Values Type: List of [JsonToken]

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonObject to a given Text object.

Syntax

```
[Ok := ]  JsonObject.WriteTo(var String: Text)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

String
Type: Text
The Text object to which the JSON data will be written.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonObject.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonObject to a given OutStream object.

Syntax

```
[Ok := ]  JsonObject.WriteTo(OutStream: OutStream)
```

Parameters

JsonObject Type: JsonObject An instance of the JsonObject data type.

OutStream
Type: OutStream
The OutStream object to which the JSON data will be written.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonObject Data Type
Getting Started with AL
Developing Extensions

JsonToken Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a container for any well-formed JSON data. A default JsonToken object contains the JSON value of NULL.

The following methods are available on instances of the JsonToken data type.

METHOD NAME	DESCRIPTION
AsArray()	Converts the value in a JsonToken to a JsonArray data type.
AsObject()	Converts the value in a JsonToken to a JsonObject data type.
AsValue()	Converts the value in a JsonToken to a JsonValue data type.
Clone()	Creates a deep-copy of the JsonToken value.
IsArray()	Indicates whether a JsonToken represents a JSON array.
IsObject()	Indicates whether a JsonToken contains a JSON object.
IsValue()	Indicates whether a JsonToken contains a JSON value.
Path()	Retrieves the JSON path of the token relative to the root of its containing tree.
ReadFrom(String)	Reads the JSON data from the string into a JsonToken variable.
ReadFrom(InStream)	Reads the JSON data from the stream into a JsonToken variable.
SelectToken(String, var JsonToken)	Selects a JsonToken using a JPath expression.
WriteTo(var Text)	Serializes and writes the JSON data of the JsonToken to a given Text object.
WriteTo(OutStream)	Serializes and writes the JSON data of the JsonToken to a given OutStream object.

NOTE

For performance reasons all HTTP, JSON, TextBuilder, and XML types are reference types, not value types. Reference types holds a pointer to the data elsewhere in memory, whereas value types store its own data.

See Also

Getting Started with AL

JsonToken.AsArray Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonToken to a JsonArray data type.

Syntax

```
Array :=   JsonToken.AsArray()
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Array Type: JsonArray The returned JsonArray contains the same data as the JsonToken, but allows array-specific operations to be performed on it.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.AsObject Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonToken to a JsonObject data type.

Syntax

```
Object :=   JsonToken.AsObject()
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Object Type: JsonObject The returned JsonObject contains the same data as the JsonToken, but allows object-specific operations to be performed on it.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.AsValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonToken to a JsonValue data type.

Syntax

```
Value :=   JsonToken.AsValue()
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Value Type: JsonValue The returned JsonValue contains the same data as the JsonToken, but allows value-specific operations to be performed on the data.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.Clone Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a deep-copy of the JsonToken value.

Syntax

```
Clone :=   JsonToken.Clone()
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Clone Type: JsonToken The Result will be a full, deep-copy of the Value.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.IsArray Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether a JsonToken represents a JSON array.

Syntax

```
Ok :=   JsonToken.IsArray()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Ok Type: Boolean **true** if the JsonToken represents a JSON array; otherwise, **false**.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.IsObject Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether a JsonToken contains a JSON object.

Syntax

```
Ok :=   JsonToken.IsObject()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Ok Type: Boolean **true** if the JsonToken represents a JSON object; otherwise, **false**.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.IsValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether a JsonToken contains a JSON value.

Syntax

```
Ok :=   JsonToken.IsValue()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Ok Type: Boolean **true** if the JsonToken represents a JSON value; otherwise, **false**.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.Path Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the JSON path of the token relative to the root of its containing tree.

Syntax

```
Path :=   JsonToken.Path()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Return Value

Path Type: String

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the string into a JsonToken variable.

Syntax

```
[Ok := ]  JsonToken.ReadFrom(String: String)
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

String
  Type: String
The String object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the JSON data is malformed. If the operation succeeds, the JsonToken will be disconnected from its current JSON tree and the data contained by the JsonToken will be replaced with the new value. To delete the contents in a JsonToken variable use the Clear function.

```
Clear(JsonToken)
```

Example

This example shows how to read JSON data from a string into a JsonToken variable.

```
local procedure ReadJson(data : Text) result : JsonToken;
begin
    result.ReadFrom(data);
end;
```

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the stream into a JsonToken variable.

Syntax

```
[Ok := ]  JsonToken.ReadFrom(InStream: InStream)
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

InStream
Type: InStream
The InStream object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the stream is in an invalid state or if the JSON data is malformed. If the operation succeeds, the JsonToken will be disconnected from its current JSON tree and the data contained by the JsonToken will be replaced with the new value. To delete the contents in a JsonToken variable use the Clear function.

```
Clear(JsonToken)
```

Example

This example shows how to read JSON data from a stream into a JsonToken variable.

```
local procedure ReadJson(source : InStream) result : JsonToken;
begin
    result.ReadFrom(source);
end;
```

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.SelectToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a JsonToken using a JPath expression.

Syntax

```
[Ok := ]  JsonToken.SelectToken(Path: String, var Result: JsonToken)
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Path
Type: String
A valid JPath expression.

Result
Type: JsonToken
A JsonToken variable that will contain the result if the operation is successful.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The operation will fail if more or less than 1 tokens are the result of evaluating the JPath.

Example

The following example shows how to select a value from a complex JSON Object. We build up a select expression in the query variable, we select the token corresponding to the salary of the employee with the given employeeId, and finally, we convert the token to a Decimal value.

We assume that the company token contains JSON data similar to the one below.

```
{ "company": {
    "employees": [
      { "id": "Marcy",
        "salary": 8.95
      },
      { "id": "John",
        "salary": 7
      },
      { "id": "Diana",
        "salary": 10.95
      }
    ]
  }
}
```

```
local procedure SelectEmployeeSalary(companyData : JsonToken; employeeId : Text) salary : Decimal;
var
    query : Text;
    salaryToken : JsonToken;
begin
    query := '$.company.employees[?(@.id=='''+employeeId+''')].salary';
    companyData.SelectToken(query, salaryToken);

    salary := salaryToken.AsValue().AsDecimal();
end;
```

NOTE

Ensure that the selected expression contains ' (single quotation mark) and not " (double quotation mark) to decorate the string value.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonToken to a given Text object.

Syntax

```
[Ok := ]  JsonToken.WriteTo(var String: Text)
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

String
Type: Text
The Text object to which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonToken.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonToken to a given OutStream object.

Syntax

```
[Ok := ]  JsonToken.WriteTo(Data: OutStream)
```

Parameters

JsonToken Type: JsonToken An instance of the JsonToken data type.

Data
Type: OutStream
The OutStream object to which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonToken Data Type
Getting Started with AL
Developing Extensions

JsonValue Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a container for any well-formed fundamental JSON value. A default JsonValue is set to the JSON value of NULL.

The following methods are available on instances of the JsonValue data type.

METHOD NAME	DESCRIPTION
AsBigInteger()	Converts the value in a JsonValue to an BigInteger data type.
AsBoolean()	Converts the value in a JsonValue to a Boolean data type.
AsByte()	Converts the value in a JsonValue to a Byte data type.
AsChar()	Converts the value in a JsonValue to a Char data type.
AsCode()	Converts the value in a JsonValue to a Code data type.
AsDate()	Converts the value in a JsonValue to a Date data type.
AsDateTime()	Converts the value in a JsonValue to a DateTime data type.
AsDecimal()	Converts the value in a JsonValue to a Decimal data type.
AsDuration()	Converts the value in a JsonValue to a Duration data type.
AsInteger()	Converts the value in a JsonValue to an Integer data type.
AsOption()	Converts the value in a JsonValue to an Option data type.
AsText()	Converts the value in a JsonValue to a Text data type.
AsTime()	Converts the value in a JsonValue to a Time data type.
AsToken()	Converts the value in a JsonValue to a JsonToken data type.
Clone()	Creates a deep-copy of the JsonToken value.
IsNull()	Indicates whether the JsonValue contains the JSON value of NULL.
IsUndefined()	Indicates whether the JsonValue contains the JSON value of UNDEFINED.
Path()	Retrieves the JSON path of the value relative to its containing tree.

METHOD NAME	DESCRIPTION
ReadFrom(String)	Reads the JSON data into a JsonValue variable.
ReadFrom(InStream)	Reads the JSON data from the stream into a JsonValue variable.
SelectToken(String, var JsonToken)	Selects a JsonToken using a JPath expression.
SetValue(Boolean)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Char)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Byte)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Option)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Integer)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(BigInteger)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Decimal)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Duration)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Date)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(Time)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(DateTime)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValue(String)	Set the contents of the JsonValue variable to the JSON representation of the given value.
SetValueToNull()	Set the contents of the JsonValue variable to the JSON representation of NULL.
SetValueToUndefined()	Set the contents of the JsonValue variable to the JSON representation of UNDEFINED.
WriteTo(var Text)	Serializes and writes the JSON data of the JsonValue to a given object.
WriteTo(OutStream)	Serializes and writes the JSON data of the JsonValue to a given object.

> **NOTE**
>
> For performance reasons all HTTP, JSON, TextBuilder, and XML types are reference types, not value types. Reference types holds a pointer to the data elsewhere in memory, whereas value types store its own data.

See Also

Getting Started with AL
Developing Extensions

JsonValue.AsBigInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to an BigInteger data type.

Syntax

```
Result :=   JsonValue.AsBigInteger()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: BigInteger If the JsonValue does not contain number or a string which can be converted without loss of precision to an BigInteger, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsBoolean Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Boolean data type.

Syntax

```
Result :=   JsonValue.AsBoolean()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Boolean The operation will succeed if the value was created from a Boolean using SetValue or if the value was parsed from a string containing one of the values : **true** or **false**. The operation will fail with a run-time error otherwise.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsByte Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Byte data type.

Syntax

```
Result :=   JsonValue.AsByte()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Byte If the JsonValue does not contain a number which can be converted without loss of precision to a Byte, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsChar Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Char data type.

Syntax

```
Result :=   JsonValue.AsChar()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Char If the JsonValue does not contain a number which can be converted without loss of precision to a Char, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Code data type.

Syntax

```
Result :=   JsonValue.AsCode()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Code

Remarks

The operation will fail with a run-time error if the JsonValue contains NULL or UNDEFINED.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsDate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Date data type.

Syntax

```
Result :=   JsonValue.AsDate()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Date If the JsonValue does not contain a string of the format "yyyy-MM-dd" e.g. "2017-01-17" (2017-January-17), the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsDateTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a DateTime data type.

Syntax

```
Result :=   JsonValue.AsDateTime()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: DateTime

Remarks

If the JsonValue does not contain a string of the format "o" as specified here the operation will fail with a run-time error.

If there is no timezone specifier, the value will be treated as a UTC DateTime. If the timezone specifier is local, it will be treated as local to the server's timezone and converted to UTC. We recommend using UTC time to prevent unexpected behavior.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsDecimal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Decimal data type.

Syntax

```
Result :=   JsonValue.AsDecimal()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Decimal If the JsonValue does not contain a number or a string which can be converted without loss of precision to a Decimal, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsDuration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Duration data type.

Syntax

```
Result :=   JsonValue.AsDuration()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Duration If the JsonValue does not contain a number or a string which can be converted without loss of precision to a BigInteger, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to an Integer data type.

Syntax

```
Result :=   JsonValue.AsInteger()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Integer If the JsonValue does not contain a number which can be converted without loss of precision to an Integer, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsOption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to an Option data type.

Syntax

```
Result :=   JsonValue.AsOption()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Option If the JsonValue does not contain a number which can be converted without loss of precision to an Option, the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Text data type.

Syntax

```
Result :=   JsonValue.AsText()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Text

Remarks

The operation will fail with a run-time error if the JsonValue contains NULL or UNDEFINED.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a Time data type.

Syntax

```
Result :=   JsonValue.AsTime()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Result Type: Time If the JsonValue does not contain a string of the format "HH:mm:ss.FFFFFFF" (see Custom Date and Time Format Strings) the operation will fail with a run-time error.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.AsToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a JsonValue to a JsonToken data type.

Syntax

```
Token :=   JsonValue.AsToken()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Token Type: JsonToken The returned JsonToken contains the same data as the JsonValue, but allows for treating the data in a generic manner.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.Clone Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a deep-copy of the JsonToken value.

Syntax

```
Clone :=   JsonValue.Clone()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Clone Type: JsonToken The Result will be a full, deep-copy of the Value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.IsNull Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether the JsonValue contains the JSON value of NULL.

Syntax

```
Ok :=   JsonValue.IsNull()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Ok Type: Boolean **true** if the JsonValue contains the JSON value of NULL; otherwise, **false**

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.IsUndefined Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether the JsonValue contains the JSON value of UNDEFINED.

Syntax

```
Ok :=   JsonValue.IsUndefined()
```

NOTE

This method can be invoked using property access syntax.

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Ok Type: Boolean **true** if the JsonValue contains the JSON value of UNDEFINED; otherwise, **false**

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.Path Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves the JSON path of the value relative to its containing tree.

Syntax

```
Path :=   JsonValue.Path()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Return Value

Path Type: String The path of the value relative to its containing JSON tree. If the object is the root of the JSON tree, the path will be empty.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data into a JsonValue variable.

Syntax

```
[Ok := ]  JsonValue.ReadFrom(Data: String)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Data
Type: String
The String object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the JSON data is malformed. If the operation succeeds, the JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value. To delete the contents in a JsonValue variable use the Clear function.

```
Clear(JsonValue)
```

Example

This example shows how to read JSON data from a string into a JsonValue variable.

```
local procedure ReadJson(data : Text) result : JsonValue;
begin
    result.ReadFrom(data);
end;
```

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads the JSON data from the stream into a JsonValue variable.

Syntax

```
[Ok := ]  JsonValue.ReadFrom(Data: InStream)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Data
Type: InStream
The InStream object from which the JSON data will be read.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method can fail if the stream is in an invalid state or if the JSON data is malformed. If the operation succeeds, the JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value. To delete the contents in a JsonValue variable use the Clear function.

```
Clear(JsonValue)
```

Example

This example shows how to read JSON data from a stream into a JsonValue variable.

```
local procedure ReadJson(source : InStream) result : JsonValue;
begin
    result.ReadFrom(source);
end;
```

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SelectToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a JsonToken using a JPath expression.

Syntax

```
[Ok := ]  JsonValue.SelectToken(Path: String, var Result: JsonToken)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Path
Type: String

Result
Type: JsonToken

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Boolean)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Boolean

Remarks

- When Value is a Boolean type, it will be stored and serialized as a JSON Boolean value.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Char)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Char

Remarks

- When Value is a Char, Byte, Option, Integer type, the integral value will be stored and serialized as a JSON Number.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Byte)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Byte

Remarks

- When Value is a Char, Byte, Option, Integer type, the integral value will be stored and serialized as a JSON Number.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Option)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Option

Remarks

- When Value is a Char, Byte, Option, Integer type, the integral value will be stored and serialized as a JSON Number.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Integer)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Integer

Remarks

- When Value is a Char, Byte, Option, Integer type, the integral value will be stored and serialized as a JSON Number.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: BigInteger)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: BigInteger

Remarks

- When Value is BigInteger, Decimal, the value will be stored as a String to not lose precision.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Decimal)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Decimal

Remarks

- When Value is BigInteger, Decimal, the value will be stored as a String to not lose precision.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Duration)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Duration

Remarks

- When Value is Duration, its underlying value, representing a 64-bit integer, is stored and serialized as a BigInteger.
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Date)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Date

Remarks

- When Value is a Date type, it will be serialized in the format yyyy-MM-dd (as described in Custom Date and Time Format Strings).
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: Time)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: Time

Remarks

- When Value is a Time type, it will be serialized using the .NET format specifier HH:mm:ss.FFFFFFF (as described in Custom Date and Time Format Strings).
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: DateTime)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: DateTime

Remarks

- When Value is a DateTime type, it will be serialized using the .NET format specifier o (o as described in Standard Date and Time Format Strings).
- The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of the given value.

Syntax

```
JsonValue.SetValue(Value: String)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Value
Type: String

Remarks

The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValueToNull Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of NULL.

Syntax

```
JsonValue.SetValueToNull()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Remarks

The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.SetValueToUndefined Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Set the contents of the JsonValue variable to the JSON representation of UNDEFINED.

Syntax

```
JsonValue.SetValueToUndefined()
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Remarks

The JsonValue will be disconnected from its current JSON tree and the data contained by the JsonValue will be replaced with the new value.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonValue to a given object.

Syntax

```
[Ok := ]  JsonValue.WriteTo(var Data: Text)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Data
Type: Text
The Text object to which the JSON data will be written.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

JsonValue.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and writes the JSON data of the JsonValue to a given object.

Syntax

```
[Ok := ]  JsonValue.WriteTo(Data: OutStream)
```

Parameters

JsonValue Type: JsonValue An instance of the JsonValue data type.

Data
Type: OutStream
The OutStream object to which the JSON data will be written.

Return Value

Ok Type: Boolean **true** if the read was successful; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

JsonValue Data Type
Getting Started with AL
Developing Extensions

KeyRef Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Identifies a key in a table and the fields in this key.

The following methods are available on instances of the KeyRef data type.

METHOD NAME	DESCRIPTION
Active()	Indicates whether the key is enabled.
FieldCount()	Gets the number of fields that have been defined in a key. Returns an error if no key is selected.
FieldIndex(Integer)	Gets the FieldRef of the field that has this index in the key referred to by the KeyRef variable. Returns an error if no key is selected.
Record()	Returns a RecordRef for the current record referred to by the key.

See Also

Getting Started with AL
Developing Extensions

KeyRef.Active Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether the key is enabled.

Syntax

```
Ok :=   KeyRef.Active()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

KeyRef Type: KeyRef An instance of the KeyRef data type.

Return Value

Ok Type: Boolean **true** if the key is enabled; otherwise, **false**.

Example

The following example uses the `KeyRef.Active` method to determine whether a key in a record is enabled. The table with ID 18 (the Customer table) is open with a reference to table 18. The KeyIndex Method (RecordRef) method retrieves the first key in the record and the `varKeyRef.Active` method returns a Boolean value that indicates whether the retrieved key is enabled. The Boolean value is displayed in a message box.

```
var
    RecRef: RecordRef;
    varKeyRef: KeyRef;
    IsActive: Boolean;
begin
    RecRef.Open(18);
    varKeyRef := RecRef.KeyIndex(1);
    IsActive := varKeyRef.Active;
    Message('Is the key active =  %1 ', IsActive);
end;
```

See Also

KeyRef Data Type
Getting Started with AL
Developing Extensions

KeyRef.FieldCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of fields that have been defined in a key. Returns an error if no key is selected.

Syntax

```
No :=   KeyRef.FieldCount()
```

NOTE

This method can be invoked using property access syntax.

Parameters

KeyRef Type: KeyRef An instance of the KeyRef data type.

Return Value

No Type: Integer The number of fields that have been defined in the key.

Example

The following example retrieves the number of fields that are defined in a key in record. The table with ID 18 (the Customer table) is open with a reference to table 18. The KeyIndex Method (RecordRef) method retrieves the second key in the record and store the *KeyRef* in the varKeyRef variable. The FieldCount Method (KeyREF) is then used to return the number of fields defined in the key and displayed in a message box.

```
var
    RecRef: RecordRef;
    varKeyRef: KeyRef;
    VarCount: Integer;
begin
    RecRef.Open(18);
    varKeyRef := RecRef.KeyIndex(2);
    VarCount := varKeyRef.FieldCount;
    Message('The number of fields defined in the key is: %1', VarCount);
end;
```

See Also

KeyRef Data Type
Getting Started with AL
Developing Extensions

KeyRef.FieldIndex Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the FieldRef of the field that has this index in the key referred to by the KeyRef variable. Returns an error if no key is selected.

Syntax

```
Field :=   KeyRef.FieldIndex(Index: Integer)
```

Parameters

KeyRef Type: KeyRef An instance of the KeyRef data type.

Index
Type: Integer
The input index.

Return Value

Field Type: FieldRef The FieldRef that refers to the field with this index in the key.

Remarks

The first field in the index must have index 1, the second index 2, and so on. The last field must have index = FieldCount. If the index is out of the range supplied or if no table is selected, the method returns an error.

Example

The following example displays the caption of a field in a record. The table with ID 18 (the Customer table) is open with a reference. The *Keyref* for the record is retrieved by using the KeyIndex Method (RecordRef). The method retrieves the second key in the record and stores the value in the varKeyRef variable. The varKeyRef variable is then used to return the *FieldRef*. The varFieldRef variable is used to display the caption of the field.

```
var
    RecRef: RecordRef;
    varKeyRef: KeyRef;
    varFieldRef: FieldRef;
begin
    RecRef.Open(18);
    varKeyRef := RecRef.KeyIndex(2);
    varFieldRef := varKeyRef.FieldIndex(1);
    Message('The Field name is: %1' ,varFieldRef.Caption);
end;
```

See Also

KeyRef Data Type

KeyRef.Record Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a RecordRef for the current record referred to by the key.

Syntax

```
Record :=   KeyRef.Record()
```

Parameters

KeyRef Type: KeyRef An instance of the KeyRef data type.

Return Value

Record Type: RecordRef The RecordRef of the record that is currently selected referenced by the key. If a key is not selected, an error is returned.

The table with ID 18 (the Customer table) is open with a reference. The KeyRef Data Type for the record is retrieved by using the KeyIndex Method (RecordRef). The method retrieves the key that has an index of 1 in the record and stores the value in the varKeyRef variable. The varKeyRef variable is then used to return the RecordRef Data Type.

```
var
    RecRef: RecordRef;
    varKeyRef: KeyRef;
begin
    RecRef.Open(18);
    varKeyRef := RecRef.KeyIndex(1);
    RecRef := varKeyRef.Record;
end;
```

See Also

KeyRef Data Type
Getting Started with AL
Developing Extensions

Label Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Denotes a string constant that can be optionally translated into multiple languages.

Parameters

All of the parameters below are optional and the order is not enforced.

ATTRIBUTE	DESCRIPTION
Comment	It is used for general comments about the label, specifically about the placeholders in that label.
Locked	When Locked is set to **true**, the label should not be translated. Default value is **false**.
MaxLength	It determines how much of the label is used. ` If no maximum length is specified, the string can be any length.

Syntax example

```
var
a:Label'LabelText',Comment='Foo',MaxLength=999,Locked=true;
```

Remarks

The `Label` data type is used in .xlf files for translations. For more information, see Working with Translation Files.

For information about naming, see CodeCop Rule AA0074.

See Also

Getting Started with AL
Developing Extensions

List Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a strongly typed list of ordered objects that can be accessed by index. Contrary to the Array data type, a List is unbounded, such that its dimension does not need to be specified upon declaration.

The following methods are available on instances of the List data type.

METHOD NAME	DESCRIPTION
Add(T)	Adds a value to the end of the List.
AddRange(T [, T,...])	Adds the elements of the specified collection to the end of the list.
AddRange(List of [T])	Adds the elements of the specified collection to the end of the list.
Contains(T)	Determines whether an element is in the List.
Count()	Gets the number of elements contained in the List.
Get(Integer, var T)	Gets the element at the specified index.
Get(Integer)	Gets the element at the specified index. This method will raise an error if the index is outside the valid range.
GetRange(Integer, Integer)	Get a shallow copy of a range of elements in the source.
GetRange(Integer, Integer, var List of [T])	Get a shallow copy of a range of elements in the source.
IndexOf(T)	Searches for the specified value and returns the one-based index of the first occurrence within the entire List.
Insert(Integer, T)	Inserts an element into the List at the specified index.
LastIndexOf(T)	Searches for the specified value and returns the one-based index of the last occurrence within the entire List.
Remove(T)	Removes the first occurrence of a specified value from the List.
RemoveAt(Integer)	Removes the element at the specified index of the List.
RemoveRange(Integer, Integer)	Removes a range of elements from the List.
Reverse()	Reverses the order of the elements in the entire List.

METHOD NAME	DESCRIPTION
Set(Integer, T)	Sets the element at the specified index.
Set(Integer, T, var T)	Sets the element at the specified index.

Remarks

The List can only be used with simple types i.e. you can have a List of [Integer] but cannot have a List of [Blob]. Similarly, the List data type does not support holding instantiated records. For this purpose, use temporary tables.

Lists are 1-based indexed, that is, the indexing of a List begins with 1.

WARNING

Previously in C/AL, one would have typically used an in-memory temporary table to create an unbounded "array" data structure, as shown in the code below. In AL you use the List Data Type instead.

```
listRec.Value := ‘Some Value’;
listRec.Insert();
```

Example

In the following example, the variable `CustomerNames` is a list of Text values which represent customer names. The procedure `WorkWithListOfCustomers` displays how one would work with the List data type. The `Add` method is used to add the string `'John'` to the `CustomerNames` list. The `Contains` method is used to check whether the list contains the specified value, in this case, the string `'John'`. We continue by using the Message procedure to display a relevant message.

```
procedure WorkWithListOfCustomers();
var
    customerNames : List of [Text];
begin
    // Adding an element to the list
    customerNames.Add('John');

    // Checking if the list contains an element
    if customerNames.Contains('John') then
        Message('John is in the list')
    else
        Message('John is not in the list')
end;
```

See Also

Getting Started with AL
Developing Extensions
Dictionary Data Type

List.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a value to the end of the List.

Syntax

```
List.Add(Value: T)
```

Parameters

List Type: List An instance of the List data type.

Value
Type: T
The value to be added to the end of the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.AddRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the elements of the specified collection to the end of the list.

Syntax

```
List.AddRange(Value: T [, Values: T,...])
```

Parameters

List Type: List An instance of the List data type.

Value
Type: T
The value to be added to the end of the List.

Values
Type: T
The collection whose elements should be added to the end of the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.AddRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the elements of the specified collection to the end of the list.

Syntax

```
List.AddRange(Values: List of [T])
```

Parameters

List Type: List An instance of the List data type.

Values

Type: List of [T]

The collection whose elements should be added to the end of the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Contains Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether an element is in the List.

Syntax

```
Result :=   List.Contains(Value: T)
```

Parameters

List Type: List An instance of the List data type.

Value
Type: T
The value to locate in the List.

Return Value

Result Type: Boolean **true** if the List contains the value, otherwise **false**.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of elements contained in the List.

Syntax

```
Count :=   List.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

List Type: List An instance of the List data type.

Return Value

Count Type: Integer The number of elements contained in the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the element at the specified index.

Syntax

```
[Ok := ]  List.Get(Index: Integer, var Result: T)
```

Parameters

List Type: List An instance of the List data type.

Index
Type: Integer
The one-based index of the element to get.
Result
Type: T
The element at the specified index.

Return Value

Ok Type: Boolean **true** if an element exists at the given index, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the element at the specified index. This method will raise an error if the index is outside the valid range.

Syntax

```
Result :=   List.Get(Index: Integer)
```

Parameters

List Type: List An instance of the List data type.

Index

Type: Integer

The one-based index of the element to get.

Return Value

Result Type: T The element at the specified index.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.GetRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Get a shallow copy of a range of elements in the source.

Syntax

```
Result :=   List.GetRange(Index: Integer, Count: Integer)
```

Parameters

List Type: List An instance of the List data type.

Index

Type: Integer

The one-based List index at which the range starts.

Count

Type: Integer

The number of elements in the range.

Return Value

Result Type: List of [T] A shallow copy of a range of elements in the source List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.GetRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Get a shallow copy of a range of elements in the source.

Syntax

```
[Ok := ]  List.GetRange(Index: Integer, Count: Integer, var Result: List of [T])
```

Parameters

List Type: List An instance of the List data type.

Index
Type: Integer
The one-based List index at which the range starts.

Count
Type: Integer
The number of elements in the range.

Result
Type: List of [T]
A shallow copy of a range of elements in the source List.

Return Value

Ok Type: Boolean **true** if the range is a valid range, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Searches for the specified value and returns the one-based index of the first occurrence within the entire List.

Syntax

```
Index :=   List.IndexOf(Value: T)
```

Parameters

List Type: List An instance of the List data type.

Value
Type: T
The value to locate in the List.

Return Value

Index Type: Integer The one-based index at which the value is found or 0 if the value does not exist in the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts an element into the List at the specified index.

Syntax

```
[Ok := ]  List.Insert(Index: Integer, Value: T)
```

Parameters

List Type: List An instance of the List data type.

Index
Type: Integer
The one-based index at which the value should be inserted.

Value
Type: T
The value to be inserted.

Return Value

Ok Type: Boolean **true** if the index was within the valid range, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.LastIndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Searches for the specified value and returns the one-based index of the last occurrence within the entire List.

Syntax

```
Index :=   List.LastIndexOf(Value: T)
```

Parameters

List Type: List An instance of the List data type.

Value
Type: T
The value to locate in the List.

Return Value

Index Type: Integer The one-based index at which the value is found or 0 if the value does not exist in the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the first occurrence of a specified value from the List.

Syntax

```
[Removed := ]  List.Remove(Value: T)
```

Parameters

List Type: List An instance of the List data type.

Value
Type: T
The value to remove from the List.

Return Value

Removed Type: Boolean **true** if item is successfully removed; otherwise, **false**. This method also returns **false** if item was not found in the List.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.RemoveAt Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the element at the specified index of the List.

Syntax

```
[Ok := ]  List.RemoveAt(Index: Integer)
```

Parameters

List Type: List An instance of the List data type.

Index
Type: Integer
The one-based index of the element to remove.

Return Value

Ok Type: Boolean **true** if the index was within the valid range, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.RemoveRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes a range of elements from the List.

Syntax

```
[Ok := ]  List.RemoveRange(Index: Integer, Count: Integer)
```

Parameters

List Type: List An instance of the List data type.

Index
Type: Integer
The one-based starting index of the range of elements to remove.

Count
Type: Integer
The number of elements to remove.

Return Value

Ok Type: Boolean **true** if the range is a valid range, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Reverse Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reverses the order of the elements in the entire List.

Syntax

```
List.Reverse()
```

Parameters

List Type: List An instance of the List data type.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the element at the specified index.

Syntax

```
[Ok := ]  List.Set(Index: Integer, NewValue: T)
```

Parameters

List Type: List An instance of the List data type.

Index

Type: Integer

The one-based index of the element to set.

NewValue

Type: T

The new value associated with the specified index.

Return Value

Ok Type: Boolean **true** if the index was within the valid range, **false** otherwise. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

List.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the element at the specified index.

Syntax

```
[Ok := ]  List.Set(Index: Integer, NewValue: T, var OldValue: T)
```

Parameters

List Type: List An instance of the List data type.

Index
Type: Integer
The one-based index of the element to set.
NewValue
Type: T
The new value associated with the specified index.
OldValue
Type: T
The value previously associated with the specified index.

Return Value

Ok Type: Boolean **true** if the index was within the valid range, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

List Data Type
Getting Started with AL
Developing Extensions

Media Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Encapsulates media files, such as image .jpg and .png files, in application database tables. The Media data type can be used as a table field data type, but cannot be used as a variable or parameter. The Media data type enables you to import a media file to the application database and reference the file from records, making it possible to display the media file in the client user interface. You can also export media from the database to files and streams.

The following methods are available on instances of the Media data type.

METHOD NAME	DESCRIPTION
ExportFile(String)	Exports the media object (such as an image) that is currently used on record to a file on your computer or network. On the record, the media object is referenced in a Media data type field.
ExportStream(OutStream)	Exports the current media object (such as a JPEG image) that is used on record to an OUTSTREAM object. The OUTSTREAM object can be created from a BLOB field, a File or from a .NET Framework interoperability object. In the record, the media is referenced in a Media data type field.
HasValue()	Checks whether a Media data type field in a record has been initialized with a media object and that the specified media object exists in the database.
ImportFile(Text, Text [, Text])	Adds a media type, such as a JPEG image, from a file to a Media data type field of a record for displaying the media with the record in the client. The media file is imported to the application database, and a reference to the media is included in the Media data type field.
ImportStream(InStream, Text [, Text])	Adds a media type (MIME), such as jpeg image, from an InStream object to a Media data type field of a record for displaying the media in the client. The media file is imported to the application database and a reference to the media is included in the Media data type field.
ImportStream(InStream, Text, Text, Text)	Adds a media type (MIME), such as jpeg image, from an InStream object to a Media data type field of a record for displaying the media in the client. The media file is imported to the application database and a reference to the media is included in the Media data type field.
MediaId()	Gets the unique identifier of a media object on a record.

See Also

Getting Started with AL

Media.ExportFile Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exports the media object (such as an image) that is currently used on record to a file on your computer or network. On the record, the media object is referenced in a Media data type field.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Result := ]  Media.ExportFile(Filename: String)
```

Parameters

Media Type: Media An instance of the Media data type.

Filename
Type: String
Specifies the full path and name of the file to create for the exported media.

Return Value

Result Type: Boolean **true** if the media was successfully exported, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The exported media file will be of the same media type, such as JPEG (.jpg) or GIF (.gif), as it was when imported. For more information about the media types, see Supported Media Types.

If a file with the same name as the exported file already exists in the target folder and the current session has write access on the file, the existing file will be automatically replaced by the new file. If the export fails, the existing file will be erased.

Example

This example uses the ExportFile method to export media objects that are used on records in a sample table that is named **My Items**. Each media object is exported to a separate file in a folder on your computer.

The example assumes that the **My Items** table already exists. Also, the table contains a **Media** data type field that is named **Image**, and one or more records already include media. For information about importing media, see ImportFile Method (Media) or ImportStream Method (Media).

The code iterates over records in the **My Items** table. If a media object is referenced in the *Image* field, the media is exported to a file in the *C:\images* folder. The file is given a name in the format *ItemNN.jpg*, where *NN*

is the number assigned to item record in the table, as specified by the **No.** field.

```
 var
    myItemRec: Record "My Items";
    fileName: Text;
    count: Integer;
    Text000: Label '%1 media files were exported';
begin
    if myItemRec.FindFirst() then begin
        repeat begin
            fileName := 'C:\images\' + 'Item' + Format(myItemRec."No.") + '.jpg';
            if myItemRec.Image.ExportFile(fileName) then
                count := count + 1
        end until myItemRec.Next < 1;
        Message(Text000, count);
    end;
end;
```

See Also

Media Data Type
Getting Started with AL
Developing Extensions

Media.ExportStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exports the current media object (such as a JPEG image) that is used on record to an OUTSTREAM object. The OUTSTREAM object can be created from a BLOB field, a File or from a .NET Framework interoperability object. In the record, the media is referenced in a Media data type field.

Syntax

```
[Result := ]  Media.ExportStream(Stream: OutStream)
```

Parameters

Media Type: Media An instance of the Media data type.

Stream

Type: OutStream

The OutStream object that is created by the object that will receive the media content.

Return Value

Result Type: Boolean **true** if the media was successfully exported, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

This example uses the ExportStream to iterate over a sample table named **My Items** table and export any media that is used on records to an OutStream that is created on a file object.

This example assumes that the **My Item** table contains a **Media** data type field that is named **Image**, and that you have already imported some media on records. For information about importing media, see ImportFile Method (Media) or ImportStream Method (Media).

```
 var
    myItemRec: Record "My Items";
    fileName: Text;
    count: Integer;
    exportFile: File;
    dataOutStream: OutStream;
    Text000: Label '%1 media files were exported';
begin
    if myItemRec.FindFirst() then begin
        repeat begin
            if myItemRec.Image.HasValue then begin
              fileName := 'C:\images\export\' + 'ItemPictureFromStream' + Format(myItemRec."No.") + '.jpg';
              exportFile.Create(fileName);
              exportFile.CreateOutstream(dataOutStream);
              myItemRec.Image.ExportStream(dataOutStream);
              count := count + 1;
              exportFile.Close;
            end;
        end until myItemRec.Next < 1;
        Message(Text000, count);
    end;
end;
```

See Also

Media Data Type
Getting Started with AL
Developing Extensions

Media.HasValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether a Media data type field in a record has been initialized with a media object and that the specified media object exists in the database.

Syntax

```
HasValue :=   Media.HasValue()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Media Type: Media An instance of the Media data type.

Return Value

HasValue Type: Boolean **true** if the Media data type field in a record has been initialized with a media object and the specified media object exists in the database, otherwise **false**.

Example

This example uses the HasValue method to iterate over the **My Items** table to determine whether media objects are available on records in the table.

The example assumes that **My Items** table exists and contains a **Media** data type field that is named **Image**. For information about importing media, see ImportFile Method (Media).

The code returns a message if a record does not include a media object.

```
 var
    myItemRec: Record "My Items";
begin
    if myItemRec.FindFirst() then begin
        repeat begin
            if not myItemRec.Image.HasValue then
                Error('Item %1 does not have a valid image', myItemRec."No.");
        end until myItemRec.Next < 1;
    end;
end;
```

See Also

Media Data Type

Media.ImportFile Method

2/17/2021 • 4 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a media type, such as a JPEG image, from a file to a Media data type field of a record for displaying the media with the record in the client. The media file is imported to the application database, and a reference to the media is included in the Media data type field.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[ID := ]  Media.ImportFile(Filename: Text, Description: Text [, MimeType: Text])
```

Parameters

Media Type: Media An instance of the Media data type.

Filename
Type: Text
Specifies the full path and name of the media file to be added.

Description
Type: Text
Specifies text that can be used in the client to describe the media.

MimeType
Type: Text
Specifies the media content type. MIME type is used by browsers, and is an Internet standard to describe the contents of a file. The MimeType value must be a two-part string that consists of a type and subtype, such as image/jpeg or image/gif. If this parameter is not specified, the function will deduct the MIME type from the file extension. For example the MIME type for a .jpg file is image/jpeg.

Return Value

ID Type: Guid The unique ID that is assigned to the media object in the database. You can also get the ID by using the MediaId method. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You use this method to upload a media, which you want to associate with a record, to the database. For example, you can upload an JPEG image for a record in a table. When a media file is imported, it is assigned a unique identifier (GUID) and stored in the system table **2000000183 Tenant Media** of the application database. The GUID is then included in the **Media** data type field as a reference to the media file.

If you import a media file into a record that already has a media object, and a modify operation is performed, the original media object will be permanently deleted from the database. However, if there are other rows in the same table that reference the original media object from the same field index, the original media object is not deleted from the database. This behavior allows a row to be copied.

Example

This example uses the ImportFile method to import a JPEG image from file to a record in a database table. After the image is imported, it can be displayed on a list page that uses the table, when the page is opened and viewed in a brick layout.

Preparation:

To support the example code that follows, create the following objects:

- A table that is named **My Items** and has the following characteristics (as a minimum):
  - An **Integer** data type field that has the name **No.**.

    This field is used to give an item a number.
  - A **Media** data type field that has the name **Image**.

    This is the field on which you will import the media file.
  - A field group that has the name **Brick** and includes the **No.** and **Image** fields.

    The field group is used to display the image on a page in the brick layout. For more information, see Define fields for a drop-down control.
- A page that is named **My Items** and has the following characteristics:
  - List type page that uses the **My Items** table as its source.
  - A repeater control that contains the fields of the **My Items** table.

    > **NOTE**
    > It is not necessary to include the **Media** data type field on the page.

  Use the page to add one or more items to the table, assigning each item a number like 1,2,3, and so on.
- JPEG image files for one or more items in the **My Items** table.
  - Give each file a name that corresponds to an item number in the table, such as 1.jpg, 2.jpg, 3.jpg, and so on.
  - Save the files in the *C:\images* folder on the computer that is running Dynamics 365 Business Central service.

Code:

With the objects in place, you can add and run the following AL code to import the images. For this code example, create a codeunit and add the code to the **OnRun** trigger of the codeunit.

The example code iterates over records in the **My Items** table. For each record, it looks in the *C:\images* folder for a file whose name matches the **No.** field of the record. If there is a match, the file is imported and a message appears; otherwise, nothing happens.

```
 var
    myItemRec: Record "My Items";
    fileName: Text;
    imageID: GUID;
    Text000: Label 'An image with the following ID has been imported on item %1: %2';
begin
    if myItemRec.FindFirst() then begin
        repeat begin
            fileName := 'C:\images\' + Format(myItemRec."No.") + '.jpg';
            if File.Exists(fileName) then begin
              imageID := myItemRec.Image.ImportFile(fileName, 'Demo image for item ' +
Format(myItemRec."No."));
              myItemRec.Modify;
              Message(Text000, myItemRec."No.", imageID)
            end;
        end until myItemRec.Next < 1;
    end;
end;
```

See Also

Media Data Type
Getting Started with AL
Developing Extensions

Media.ImportStream Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a media type (MIME), such as jpeg image, from an InStream object to a Media data type field of a record for displaying the media in the client. The media file is imported to the application database and a reference to the media is included in the Media data type field.

Syntax

```
[ID := ]  Media.ImportStream(Stream: InStream, Description: Text [, MimeType: Text])
```

Parameters

Media Type: Media An instance of the Media data type.

Stream
Type: InStream
The InStream object that contains the media that you want to use on the record.

Description
Type: Text
Specifies text that can be used in the client to describe the media file.

MimeType
Type: Text
Specifies the media content type. MIME type is used by browsers, and is an Internet standard to describe the contents of a file. The MimeType value must be a two-part string that consists of a type and subtype, such as image/jpeg or image/gif. If this parameter is not specified, the function will deduct the MIME type from the file extension. For example the MIME type for a .jpg file is image/jpeg.

Return Value

ID Type: Guid The unique ID that is assigned to the media object in the database. You can also get the ID by using the MediaId method. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use this method to import media into the database, and associate the media with a record. For example, you can upload an image of all items in table **27 Item**. When media is imported, the object is stored in the system table **2000000184 Tenant Media** of the application database.

If you import media on a record that already has a media object, and a modify operation is performed, the original media object will be permanently deleted from the database. However, if there are other rows in the same table that reference the original media object from the same field index, the original media object is not deleted from the database. This behavior allows a row to be copied.

Example

Preparation:
To support the example code that follows, create the following objects:

- A table that is named **My Items** and has the following characteristics (as a minimum):
  - An **Integer** data type field that has the name **No.**.

    This field is used to give an item a number.
  - A **Media** data type field that has the name **Image**.

    This is the field on which you will import the media file.
  - A field group that has the name **Brick** and includes the **No.** and **Image** fields.

    The field group is used to display the image on a page in the brick layout. For more information, see Define fields for a drop-down control.
- A page that is named **My Items** and has the following characteristics:
  - List type page that uses the **My Items** table as its source.
  - A repeater control that contains the fields of the **My Items** table.

    > **NOTE**
    > It is not necessary to include the **Media** data type field on the page.

  Use the page to add one or more items to the table, assigning each item a number like 1,2,3, and so on.
- JPEG image files for one or more items in the **My Items** table.
  - Give each file a name that corresponds to an item number in the table, such as 1.jpg, 2.jpg, 3.jpg, and so on.
  - Save the files in the *C:\images* folder on the computer that is running Dynamics 365 Business Central service.

Code
With the objects in place, you can add and run the following AL code to import the images. For this code example, create a codeunit and add the code to the **OnRun** trigger of the codeunit.

This code iterates over records in the **My Items** table. For each record, it looks in the *C:\images* folder for a file whose name matches the **No.** field of the record. If there is a match the file, an InStream object is created for the file, the media is imported into the record, and a confirmation message is returned.

```
 var
    myItemRec: Record "My Items";
    fileName: Text;
    importFile: File;
    imageInStream: InStream;
    imageID: GUID;
    Text000: Label 'An image with the following ID has been imported on item %1: %2';
begin
    if  myItemRec.FindFirst() then begin
        repeat begin
            fileName := 'C:\images\' + Format(myItemRec."No.") + '.jpg';

            if File.Exists(fileName) then begin
                importFile.Open(fileName);
                importFile.CreateInstream(imageInstream);
                imageID := myItemRec.Image.ImportStream(imageInstream, 'Demo image for item ' + Format(
myItemRec."No."));
                myItemRec.Modify;
                Message(Text000, myItemRec."No.", imageID);
                importFile.Close;
            end;
        end until myItemRec.Next < 1;
    end;
end;
```

See Also

Media Data Type
Getting Started with AL
Developing Extensions

Media.ImportStream Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Adds a media type (MIME), such as jpeg image, from an InStream object to a Media data type field of a record for displaying the media in the client. The media file is imported to the application database and a reference to the media is included in the Media data type field.

NOTE

This method is supported only in Business Central on-premises.

Syntax

```
[ID := ]  Media.ImportStream(Stream: InStream, Description: Text, MimeType: Text, FileName: Text)
```

Parameters

Media Type: Media An instance of the Media data type.

Stream

Type: InStream

The InStream object that contains the media that you want to use on the record.

Description

Type: Text

Specifies text that can be used in the client to describe the media file.

MimeType

Type: Text

Specifies the media content type. MIME type is used by browsers, and is an Internet standard to describe the contents of a file. The MimeType value must be a two-part string that consists of a type and subtype, such as image/jpeg or image/gif. If this parameter is not specified, the function will deduct the MIME type from the file extension. For example the MIME type for a .jpg file is image/jpeg.

FileName

Type: Text

Specifies the file name to associate with the Media object.

Return Value

ID Type: Guid The unique ID that is assigned to the media object in the database. You can also get the ID by using the MediaId method. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use this method to import media into the database, and associate the media with a record. For example,

you can upload an image of all items in table **27 Item**. When media is imported, the object is stored in the system table **2000000184 Tenant Media** of the application database.

If you import media on a record that already has a media object, and a modify operation is performed, the original media object will be permanently deleted from the database. However, if there are other rows in the same table that reference the original media object from the same field index, the original media object is not deleted from the database. This behavior allows a row to be copied.

Example

Preparation:
To support the example code that follows, create the following objects:

- A table that is named **My Items** and has the following characteristics (as a minimum):
  - An **Integer** data type field that has the name **No.**.

    This field is used to give an item a number.
  - A **Media** data type field that has the name **Image**.

    This is the field on which you will import the media file.
  - A field group that has the name **Brick** and includes the **No.** and **Image** fields.

    The field group is used to display the image on a page in the brick layout. For more information, see Define fields for a drop-down control.
- A page that is named **My Items** and has the following characteristics:
  - List type page that uses the **My Items** table as its source.
  - A repeater control that contains the fields of the **My Items** table.

    > **NOTE**
    > It is not necessary to include the **Media** data type field on the page.

  Use the page to add one or more items to the table, assigning each item a number like 1,2,3, and so on.
- JPEG image files for one or more items in the **My Items** table.
  - Give each file a name that corresponds to an item number in the table, such as 1.jpg, 2.jpg, 3.jpg, and so on.
  - Save the files in the *C:\images* folder on the computer that is running Dynamics 365 Business Central service.

Code
With the objects in place, you can add and run the following AL code to import the images. For this code example, create a codeunit and add the code to the **OnRun** trigger of the codeunit.

This code iterates over records in the **My Items** table. For each record, it looks in the *C:\images* folder for a file whose name matches the **No.** field of the record. If there is a match the file, an InStream object is created for the file, the media is imported into the record, and a confirmation message is returned.

```
 var
    myItemRec: Record "My Items";
    fileName: Text;
    importFile: File;
    imageInStream: InStream;
    imageID: GUID;
    Text000: Label 'An image with the following ID has been imported on item %1: %2';
begin
    if  myItemRec.FindFirst() then begin
        repeat begin
            fileName := 'C:\images\' + Format(myItemRec."No.") + '.jpg';

            if File.Exists(fileName) then begin
                importFile.Open(fileName);
                importFile.CreateInstream(imageInstream);
                imageID := myItemRec.Image.ImportStream(imageInstream, 'Demo image for item ' + Format(
myItemRec."No."));
                myItemRec.Modify;
                Message(Text000, myItemRec."No.", imageID);
                importFile.Close;
            end;
        end until myItemRec.Next < 1;
    end;
end;
```

See Also

Media Data Type
Getting Started with AL
Developing Extensions

Media.MediaId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the unique identifier of a media object on a record.

Syntax

```
MediaId :=   Media.MediaId()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Media Type: Media An instance of the Media data type.

Return Value

MediaId Type: Guid The GUID of the Media object in the database.

Remarks

When a media is imported on the Media data type field of table record, the media is given a GUID and stored in the system table **2000000184 Tenant Media** of the application database. The GUID is then included in the **Media** data type field as a reference to the media in the database.

Example

This example uses the MediaId method to get the GUID of the media object that is used on item number **1** in the a the table named **My Items**.

The example assumes that the **My Items** table already exists and has a **Media** data type field named **Image**.

```
 var
    myItemRec: Record "My Items";
    imageID: GUID;
    Text000: Label 'Item %1 has a media object with the following ID: %2';
begin
    myItemRec.Get('1');
    mediaGuid := myItemRec.Image.MediaId;
    Message(Text000, myItemRec."No.", imageID);
end;
```

See Also

Media Data Type

MediaSet Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Encapsulates media, such as images, in application database tables.

The following methods are available on instances of the MediaSet data type.

METHOD NAME	DESCRIPTION
Count()	Gets the number of media objects that are included in the MediaSet of a record.
ExportFile(String)	Exports the media objects in the current media set of a record to individual files on your computer or network. In the record, the media set is referenced in a MediaSet data type field.
ImportFile(String, String [, String])	Adds a media, such as a JPEG image, to the MediaSet data type field of a record for displaying the media in the client. The media is imported to the database and included in a MediaSet for the record.
ImportStream(InStream, String [, String])	Adds a media file, such as a JPEG image, from an InStream object to the MediaSet of record for displaying in the client. The media is imported to the database and included in a MediaSet for the record.
Insert(Guid)	Adds a media object that already exists in the database to a MediaSet of a record.
Item(Integer)	Gets the unique identifier (GUID) of a media object that is assigned to a MediaSet on a record.
MediaId()	Gets the unique identifier that is assigned to a MediaSet of a record. The MediaSet is a collection of media objects that are used on the record that can be displayed in the client.
Remove(Guid)	Removes a media object from a MediaSet of a record.

See Also

Getting Started with AL
Developing Extensions
Working With Media on Records

MediaSet.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of media objects that are included in the MediaSet of a record.

Syntax

```
Count :=   MediaSet.Count()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

Return Value

Count Type: Integer The number of media objects that are included in the MediaSet of a record.

Example

This example counts the number of media objects that are available for item No. 1000 in table **27 Item** of the CRONUS demonstration database. In this example, the field in the **Item** table that is used for the MediaSet data type is **Picture**.

```
 var
    itemRec: Record Item;
    count: Integer;
    Text000: Label 'The number of media files: %1';
begin
    itemRec.Get('1000');
        count := (itemRec.Picture.Count);
        Message(Text000,count);
end;
```

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

MediaSet.ExportFile Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exports the media objects in the current media set of a record to individual files on your computer or network. In the record, the media set is referenced in a MediaSet data type field.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Count := ]  MediaSet.ExportFile(FilenamePrefix: String)
```

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

FilenamePrefix
Type: String
Specifies the location and name of the exported media files. Each exported media file is given a name that consists of a prefix that you specify, plus an index number that is automatically assigned. The file name has the format prefix-index.type, for example, Image-1.jpg, Image-2.jpg, and Image-3.jpg. To set the parameter value, use the format: path\prefix.type.

- path is the folder path where you want to store the files.
- prefix is the text that you want before the index number.
- type is the media type extension.

Return Value

Count Type: Integer The number of exported elements. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The exported media files will be of the same media file type as when they were imported. For more information about the media types, see Supported Media Types.

The method has the following behavior:

- If a file that has the same name as the exported file already is located in the target folder and the current session has write access on the file, the existing file will be automatically replaced by the new file.
- If the export fails, the existing file will be erased.
- If a media in the media set cannot be found in the database, no file will be generated for this object.

Example

This example first imports two media files (JPEG image files) from a local folder to the media set of a record in the table **27 Item** of the CRONUS demonstration database. Then, using the EXPORTFile method, the media objects are exported to files again in another local folder.

For using media sets on records, the **Item** table includes a **MediaSet** data type field that is named **Picture**.

The code imports the JPEG image files (.jpg) from the folder *C:\images* to record *1000* in the **Item** table, and then exports the media files to the folder *C:\images\export*.

```
var
    itemRec: Record Item;
    count: Boolean;
    Text000: Label '%1 media files were exported.';
begin
    // Import image files the C:\images folder.
    itemRec.Get('1000');
    itemRec.Picture.ImportFile('C:\images\1000-v1.jpg', 'Demo image for item ' + Format(itemRec."No."));
    itemRec.Picture.ImportFile('C:\images\1000-v2.jpg', 'Demo image for item ' + Format(itemRec."No."));
    itemRec.Modify;
    Commit;

    // Export the MediaSet to two separate image files in the c:\images\export folder.
    itemRec.Get('1000');
    count := itemRec.Picture.ExportFile('C:\images\export\' + 'Item1000Image.jpg');
    Message('%1 files exported.', count);
end;
```

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

MediaSet.ImportFile Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a media, such as a JPEG image, to the MediaSet data type field of a record for displaying the media in the client. The media is imported to the database and included in a MediaSet for the record.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[ID := ]  MediaSet.ImportFile(Filename: String, Description: String [, MimeType: String])
```

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

Filename
Type: String
Specifies the full path and name of the media file to be imported.

Description
Type: String
Specifies text that can be used in the client to describe the media.

MimeType
Type: String
Specifies the media content type. MIME type is used by browsers, and is an Internet standard to describe the contents of a file. The MimeType value must be a two-part string that consists of a type and subtype, such as image/jpeg or image/gif. If this parameter is not specified, the function will deduct the MIME type from the file extension. For example the MIME type for a .jpg file is image/jpeg.

Return Value

ID Type: Guid The unique ID that is assigned to the MediaSet of the record. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You use this method to upload a media file as part of a collection of media objects that you want to associate with a record. The method is similar to the ImportFile Method (Media) except that this method enables you to import multiple media files for the same record. For example, you can add multiple images for an item in table **27 Item**.

When a media file is imported, a media object is created and stored in the system table **2000000181 Tenant Media** of the application database. The media object is assigned a unique identifier (GUID).

In addition, the media object is assigned to a MediaSet which also has a specific GUID. This GUID is included in the MediaSet data type field as a reference to the media objects. The MediaSet and its GUID are created with the first media that is imported, and the information is stored in table **2000000183 Tenant Media Set**. All additional media objects for the record are then associated with the same MediaSet GUID.

Example

This example uses the IMPORTFile method to add images to records in table **27 Item** of the CRONUS demonstration database.

In support of the example code, you alsmust complete these tasks:

- Create two sample image files that you want to use on item no. 1000 in table **27 Item**.

  Save the images as JPEG type, and give them the names 1000-v1.jpg and 1000-v2.jpg. Save the files in the *C:\images* folder on the computer that is running Dynamics 365 Business Central service instance.

- Verify that table **27 Item** has a field that is called **Picture** and has the data type **MediaSet**.

  This is field on which you will add the images. If the field is not present, then add it.

With these tasks in place, you can add the following AL code for importing the images. For this code example, create a codeunit, and add the code to the **OnRun** trigger.

```
 var
    itemRec: Record Item;
    count: Integer;
    mediasetId: GUID;
    Text000: Label 'The files have been imported. Item %1 has %2 pictures in MediaSet: %3';
begin
    itemRec.Get('1000');
    itemRec.Picture.ImportFile('C:\images\1000-v1.jpg', 'Demo image for item ' + Format(itemRec."No."));
    itemRec.Picture.ImportFile('C:\images\1000-v2.jpg', 'Demo image for item ' + Format(itemRec."No."));
    count := (itemRec.Picture.Count);
    mediasetId := itemRec.Picture.MediaId;
    Message(Text000,itemRec."No.", count, mediasetId);
end;
```

If you run system table **2000000181 Tenant Media**, you should see the new images in the list.

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

MediaSet.ImportStream Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a media file, such as a JPEG image, from an InStream object to the MediaSet of record for displaying in the client. The media is imported to the database and included in a MediaSet for the record.

Syntax

```
[ID := ]  MediaSet.ImportStream(Stream: InStream, Description: String [, MimeType: String])
```

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

Stream
Type: InStream
Specifies the InStream object that contains the media that you want to use on the record.

Description
Type: String
Specifies text that can be used in the client to describe the media files.

MimeType
Type: String
Specifies the media content type. MIME type is used by browsers, and is an Internet standard to describe the contents of a file. The MimeType value must be a two-part string that consists of a type and subtype, such as image/jpeg or image/gif. If this parameter is not specified, the function will deduct the MIME type from the file extension. For example the MIME type for a .jpg file is image/jpeg.

Return Value

ID Type: Guid The unique ID that is assigned to the MediaSet of the record. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use this method to upload media that you want to associate with a record to the database. For example, you can upload media to items in table **27 Item**. The method is similar to the ImportStream Method (Media) except that this method enables you to import multiple media objects on the same record.

When a media is imported, it is assigned a unique identifier (GUID) and stored in the system table **2000000181 Tenant Media** of the application database.

In addition, the media object is assigned to a MediaSet which also has a specific GUID. This GUID is included in the MediaSet data type field as a reference to the media objects. The MediaSet and its GUID are created with the first media that is imported, and the information is stored in table **2000000183 Tenant Media Set**. All additional media objects for the record are then associated with the same MediaSet GUID.

Example

This example uses the ImportStream method to add images to records in table **27 Item** of the CRONUS demonstration database.

To support the example code that follows, you also have to complete these tasks:

- Create two sample image files that you want to use on item no. 1000 in table **27 Item**.

  Save the images as JPEG type, and give them the names 1000-v1.jpg and 1000-v2.jpg. Save the files in the *C:\images* folder on the computer that is running Dynamics 365 Business Central service instance.

- Verify that table **27 Item** has a field that is called **Picture** and has the data type **MediaSet**.

  This is field on which you will add the images. If the field is not present, then add it.

With these tasks in place, you can add and run the following AL code to import the images. For this code example, create a codeunit and add the code to the OnRun trigger.

```
 var
    itemRec: Record Item;
    count: Integer;
    fileName: Text;
    inStreamObject: InStream;
    importFile: File;
    mediasetId: GUID;
    Text000: Label 'The files have been imported. Item %1 has %2 pictures in MediaSet: %3';
begin
    itemRec.Get('1000');

    fileName := 'C:\images\1000-v1.jpg';
    importFile.Open(fileName);
    importFile.CreateInStream(inStreamObject);
    itemRec.Picture.ImportStream(inStreamObject, 'Demo image for item ' + Format(itemRec."No."));
    itemRec.Modify;
    importFile.Close;

    fileName := 'C:\images\1000-v2.jpg';
    importFile.Open(fileName);
    importFile.CreateInStream(inStreamObject);
    itemRec.Picture.ImportStream(inStreamObject, 'Demo image for item ' + Format(itemRec."No."));
    itemRec.Modify;
    importFile.Close;

    count := (itemRec.Picture.Count);
    mediasetId := itemRec.Picture.MediaId;
    Message(Text000,itemRec."No.",count,mediasetId);
end;
```

If you run system table **2000000181 Tenant Media**, you should see the new images in the list.

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

MediaSet.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a media object that already exists in the database to a MediaSet of a record.

Syntax

```
[Result := ]  MediaSet.Insert(MediaId: Guid)
```

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

MediaId
Type: Guid
Specifies the unique ID that is assigned to the media object that you want to insert. Existing media objects are stored in the system table 2000000184 Tenant Media of the application database. There are different ways of obtaining the GUID of a media object. You could identify the media object ID by looking in the table. Or programmatically, you can use either the Item function on a MediaSet data type field of a record or the MEDIAID function on Media data type field of a record.

Return Value

Result Type: Boolean **true** if the media is successfully added to the MediaSet, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

When media object is inserted in a MediaSet, it is assigned in index number. For more information, see Indexing of media objects in a media set.

Example

This example uses the Insert method and Item Method (MediaSet) to take a media object that is already in the database and assigned to a record in a table (TableA), and add it to the MediaSet of a record in another table (TableB). This example assumes the following:

- TableA and TableB already exist
- Each table has a MediaSet data type field called **Images**
- Each table contains the record number '1000'.
- There is at least 1 media object in the MediaSet of record 1000 in TableA.

```
 var
    recA: Record TableA;
    recB: Record TableB;
    mediasetId: GUID;
    Text000: Label 'Media %1 was added to MediaSet %2.';
    Text000: Label 'The media was not added to MediaSet %1.';
begin
    // Retrieves the GUID of the first media object (index number 1) in the MediaSet of record 1000 in
TableA
    recA.Get('1000');
    MediaId := recA.Images.Item(1);

    // Adds media object to the MediaSet of record 1000 in TableB based on the media object GUID
    recB.Get('1000');
    if recB.Images.Insert(mediaId) then begin
        recB.Modify;
        Message(Text000, mediaId, recB.Images.MediaId);
    end else begin
       Message(Text001);
    end;
end;
```

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

MediaSet.Item Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the unique identifier (GUID) of a media object that is assigned to a MediaSet on a record.

Syntax

```
MediaId :=   MediaSet.Item(Index: Integer)
```

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

Index

Type: Integer

Specifies the index number that is assigned to the media object in the MediaSet.

Return Value

MediaId Type: Guid The unique identifier (GUID) of the media object.

Example

For an example of using the Item method, see this example.

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

MediaSet.MediaId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the unique identifier that is assigned to a MediaSet of a record. The MediaSet is a collection of media objects that are used on the record that can be displayed in the client.

Syntax

```
MediaId :=   MediaSet.MediaId()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

Return Value

MediaId Type: Guid The GUID of MediaSet on the record.

Remarks

When you import media on a table record by using either the ImportFile Method (MediaSet) or ImportStream Method (MediaSet), the media is assigned to a MediaSet GUID in the system table **2000000183 Tenant Media Set** of the application database. You can use the MediaId method to retrieve the MediaSet GUID. Note that the imported media object is also assigned a GUID. To get the media object's GUID, you can use the MediaId Method (Media).

Example

This example is gets the GUID of the MediaSet that is used on item No. 1000 in the **Item** table. The field in the **Item** table that is used for the MediaSet data type is **Picture**.

```
 var
    item: Record Item;
    mediasetId: GUID;
    Text000: Label 'The GUID of the MediaSet is: %1';
begin
    item.Get('1000');
    mediasetId := item.Picture.MediaId;
    Message(Text000, mediasetId);
end;
```

See Also

MediaSet.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes a media object from a MediaSet of a record.

Syntax

```
[Result := ]  MediaSet.Remove(MediaId: Guid)
```

Parameters

MediaSet Type: MediaSet An instance of the MediaSet data type.

MediaId
Type: Guid
Specifies the unique ID that is assigned to the media object that you want to remove from the MediaSet. Existing media objects are stored in the system table 2000000184 Tenant Media of the application database. There are different ways of obtaining the GUID of a media object. You could identify the media object ID by looking in the table. Or programmatically, you can use either the Item function on a MediaSet data type field of a record or the MEDIAID function on Media data type field of a record.

Return Value

Result Type: Boolean **true** if the object was removed or **false** if a media object with the given ID was not present in the set. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The Remove method disassociates the media object from the MediaSet. It does not delete the media object from the database.

Example

This example uses the REMOVE method and Item Method (MediaSet) to remove a media object from the MediaSet for record '1000' in the table called TableA. This example assumes the following about TableA:

- It has a MediaSet data type field called **Images**
- It contains the record number '1000'.
- Record '1000' has at least 1 media object in the MediaSet.

```
 var
    recA: Record TableA;
    mediasetId: GUID;
    Text000: Label 'Media %1 was removed from MediaSet %2.';
    Text001: Label 'The media was not removed from MediaSet %1.';
begin
    // Retrieves the GUID of the first media object (index number 1) in the MediaSet of record 1000 in
TableA
    recA.Get('1000');
    mediaId := recA.Images.Item(1);

    // Removes the media object from the MediaSet of record 1000
    if recA.Images.Remove(mediaId) then begin
        recA.Modify;
        Message(Text000, mediaId, recA.Images.MediaId);
    end else begin
        Message(Text001);
    end;
end;
```

See Also

MediaSet Data Type
Getting Started with AL
Developing Extensions

ModuleDependencyInfo Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides information about a dependent module.

The following methods are available on instances of the ModuleDependencyInfo data type.

METHOD NAME	DESCRIPTION
Id()	Gets the app ID of the specified app.
Name()	Gets the name of the specified application.
Publisher()	Gets the publisher of the specified application.

See Also

Getting Started with AL
Developing Extensions

ModuleDependencyInfo.Id Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the app ID of the specified app.

Syntax

```
Id :=   ModuleDependencyInfo.Id()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ModuleDependencyInfo Type: ModuleDependencyInfo An instance of the ModuleDependencyInfo data type.

Return Value

Id Type: Guid The application ID.

See Also

ModuleDependencyInfo Data Type
Getting Started with AL
Developing Extensions

ModuleDependencyInfo.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of the specified application.

Syntax

```
Name :=   ModuleDependencyInfo.Name()
```

NOTE

This method can be invoked using property access syntax.

Parameters

ModuleDependencyInfo Type: ModuleDependencyInfo An instance of the ModuleDependencyInfo data type.

Return Value

Name Type: String The application name.

See Also

ModuleDependencyInfo Data Type
Getting Started with AL
Developing Extensions

ModuleDependencyInfo.Publisher Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the publisher of the specified application.

Syntax

```
Publisher :=   ModuleDependencyInfo.Publisher()
```

NOTE

This method can be invoked using property access syntax.

Parameters

ModuleDependencyInfo Type: ModuleDependencyInfo An instance of the ModuleDependencyInfo data type.

Return Value

Publisher Type: String The application publisher.

See Also

ModuleDependencyInfo Data Type
Getting Started with AL
Developing Extensions

ModuleInfo Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents information about an application consumable from AL.

The following methods are available on instances of the ModuleInfo data type.

METHOD NAME	DESCRIPTION
AppVersion()	Gets the version of the specified application's metadata.
DataVersion()	Gets the version of the specified application's data in the context of a given tenant. This indicates the last version that was installed or successfully upgraded to and will not match the application version if the tenant is in a data upgrade pending state.
Dependencies()	Gets the collection of application dependencies.
Id()	Gets the ID of the specified application.
Name()	Gets the name of the specified application.
Publisher()	Gets the publisher of the specified application.

See Also

Getting Started with AL
Developing Extensions

ModuleInfo.AppVersion Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the version of the specified application's metadata.

Syntax

```
AppVersion :=   ModuleInfo.AppVersion()
```

NOTE

This method can be invoked using property access syntax.

Parameters

ModuleInfo Type: ModuleInfo An instance of the ModuleInfo data type.

Return Value

AppVersion Type: Version The version of the specified application's metadata.

See Also

ModuleInfo Data Type
Getting Started with AL
Developing Extensions

ModuleInfo.DataVersion Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the version of the specified application's data in the context of a given tenant. This indicates the last version that was installed or successfully upgraded to and will not match the application version if the tenant is in a data upgrade pending state.

Syntax

```
DataVersion :=   ModuleInfo.DataVersion()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ModuleInfo Type: ModuleInfo An instance of the ModuleInfo data type.

Return Value

DataVersion Type: Version The version

See Also

ModuleInfo Data Type
Getting Started with AL
Developing Extensions

ModuleInfo.Dependencies Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the collection of application dependencies.

Syntax

```
Dependencies :=   ModuleInfo.Dependencies()
```

NOTE

This method can be invoked using property access syntax.

Parameters

ModuleInfo Type: ModuleInfo An instance of the ModuleInfo data type.

Return Value

Dependencies Type: List of [ModuleDependencyInfo] Collection of application dependencies.

See Also

ModuleInfo Data Type
Getting Started with AL
Developing Extensions

ModuleInfo.Id Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the ID of the specified application.

Syntax

```
Id :=   ModuleInfo.Id()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ModuleInfo Type: ModuleInfo An instance of the ModuleInfo data type.

Return Value

Id Type: Guid The ID of the specified application.

See Also

ModuleInfo Data Type
Getting Started with AL
Developing Extensions

ModuleInfo.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of the specified application.

Syntax

```
Name :=   ModuleInfo.Name()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

ModuleInfo Type: ModuleInfo An instance of the ModuleInfo data type.

Return Value

Name Type: String The name of the specified application.

See Also

ModuleInfo Data Type
Getting Started with AL
Developing Extensions

ModuleInfo.Publisher Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the publisher of the specified application.

Syntax

```
Publisher :=   ModuleInfo.Publisher()
```

NOTE

This method can be invoked using property access syntax.

Parameters

ModuleInfo Type: ModuleInfo An instance of the ModuleInfo data type.

Return Value

Publisher Type: String The publisher of the specified application.

See Also

ModuleInfo Data Type
Getting Started with AL
Developing Extensions

NavApp Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides information about a NavApp.

The following methods are available on the NavApp data type.

METHOD NAME	DESCRIPTION
DeleteArchiveData(Integer)	Deletes the archived data for a specified table of an extension during installation.
GetArchiveRecordRef(Integer, var RecordRef)	Returns a RecordRef for the specified table.
GetArchiveVersion()	Returns the version of the extension that the specified table is part of.
GetCallerModuleInfo(var ModuleInfo)	Gets information about the extension that contains the method that called the currently running method. For example, if method 1 (in extension A) calls method 2 (in extension B), which calls GetCallerModuleInfo, then GetCallerModuleInfo will return information about extension A.
GetCurrentModuleInfo(var ModuleInfo)	Gets information about the application that contains the AL object that is currently running.
GetModuleInfo(Guid, var ModuleInfo)	Gets information about the specified AL application.
IsInstalling()	Returns **true** if the application that contains the AL object that is currently running is being installed, otherwise it returns **false**.
LoadPackageData(Integer)	Loads default, or starting, table data into the specified table of an extension during installation.
RestoreArchiveData(Integer [, Boolean])	Restores archived data for a specified table of an extension during installation.

See Also

Getting Started with AL
Developing Extensions

NavApp.DeleteArchiveData Method

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0 until version 6.0 where it was deprecated for the following reason: "The features related to data migration from V1 to V2 extensions are being deprecated."*

Deletes the archived data for a specified table of an extension during installation.

Syntax

```
NavApp.DeleteArchiveData(TableNo: Integer)
```

Parameters

TableNo

Type: Integer

The ID of the table for which to delete archived data.

If you omit this optional return value and if archived data cannot be deleted for the specified table, then a run-time error occurs. If you include a return value, then it is assumed that you will handle any errors and no run-time error occurs, even though the archived data is not deleted.

Remarks

You use this method as part of the upgrade code for an extension, where it is called from the `OnNavAppUpgradePerDatabase()` or `OnNavAppUpgradePerCompany()` system methods. When an extension is uninstalled, the data in application tables of the extension is automatically stored into a set of special tables so that the data is still preserved. With the DeleteARCHIVEDATA method, you can delete the archived data from the application table of the new version of an extension when it is installed.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.GetArchiveRecordRef Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 6.0 where it was deprecated for the following reason: "The features related to data migration from V1 to V2 extensions are being deprecated."*

Returns a RecordRef for the specified table.

Syntax

```
[Ok := ]  NavApp.GetArchiveRecordRef(TableNo: Integer, var RecordRef: RecordRef)
```

Parameters

TableNo

Type: Integer

Specifies the table ID.

RecordRef

Type: RecordRef

Specifies the reference.

Return Value

Ok Type: Boolean **true**, if a record was found; other **false**

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.GetArchiveVersion Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 6.0 where it was deprecated for the following reason: "The features related to data migration from V1 to V2 extensions are being deprecated."*

Returns the version of the extension that the specified table is part of.

Syntax

```
Version :=   NavApp.GetArchiveVersion()
```

Return Value

Version Type: String The version

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.GetCallerModuleInfo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Gets information about the extension that contains the method that called the currently running method. For example, if method 1 (in extension A) calls method 2 (in extension B), which calls GetCallerModuleInfo, then GetCallerModuleInfo will return information about extension A.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  NavApp.GetCallerModuleInfo(var Info: ModuleInfo)
```

Parameters

Info
Type: ModuleInfo
A value containing information about the calling application.

Return Value

Ok Type: Boolean **true** if the information could be retrieved, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.GetCurrentModuleInfo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets information about the application that contains the AL object that is currently running.

Syntax

```
[Ok := ]  NavApp.GetCurrentModuleInfo(var Info: ModuleInfo)
```

Parameters

Info

Type: ModuleInfo

A value containing information about the currently running application.

Return Value

Ok Type: Boolean **true** if the information could be retrieved, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.GetModuleInfo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets information about the specified AL application.

Syntax

```
[Ok := ]  NavApp.GetModuleInfo(AppId: Guid, var Info: ModuleInfo)
```

Parameters

AppId

Type: Guid

The ID of the application for which to retrieve information.

Info

Type: ModuleInfo

A value containing information about the application with the given ID.

Return Value

Ok Type: Boolean **true** if the information could be retrieved, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.IsInstalling Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns **true** if the application that contains the AL object that is currently running is being installed, otherwise it returns **false**.

Syntax

```
Result :=   NavApp.IsInstalling()
```

Return Value

Result Type: Boolean **true** if the application that contains the AL object that is currently running is being installed, otherwise **false**.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.LoadPackageData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads default, or starting, table data into the specified table of an extension during installation.

Syntax

```
NavApp.LoadPackageData(TableNo: Integer)
```

Parameters

TableNo

Type: Integer

The ID of the table for which to load package data.

Remarks

You use this method as part of the upgrade code for an extension, where it is called from the `OnNavAppUpgradePerDatabase()` or `OnNavAppUpgradePerCompany()` system methods. With the LOADPACKAGEDATA method, you can populate a new table in your extension with data to help users get started using your extension.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

NavApp.RestoreArchiveData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 6.0 where it was deprecated for the following reason: "The features related to data migration from V1 to V2 extensions are being deprecated."*

Restores archived data for a specified table of an extension during installation.

Syntax

```
[Ok := ]  NavApp.RestoreArchiveData(TableNo: Integer [, RunTrigger: Boolean])
```

Parameters

TableNo
Type: Integer
The ID of the table for which to restore achived data.
RunTrigger
Type: Boolean
true if the table triggers should run, otherwise **false**.

Return Value

Ok Type: Boolean **true**, if the archived data was restored for the specified table; otherwise **false** If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You use this method as part of the upgrade code for an extension, where it is called from the `OnNavAppUpgradePerDatabase()` or `OnNavAppUpgradePerCompany()` system methods. When an extension is uninstalled, the data in application tables of the extension is automatically stored into a set of special tables so that the data is still preserved. With the RESTOREARCHIVEDATA method, you can restore the archived data to the application table of the new version of an extension when it is installed.

See Also

NavApp Data Type
Getting Started with AL
Developing Extensions

None Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is used implicitly when a method does not return a value.

See Also

Getting Started with AL
Developing Extensions

Notification Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides a programmatic way to send non-intrusive information to the user interface (UI) in the Business Central Web client.

The following methods are available on instances of the Notification data type.

METHOD NAME	DESCRIPTION
AddAction(String, Integer, String)	Specifies an action for the notification.
GetData(String)	Retrieves data that was passed to a notification instance as specified by a SetData method call.
HasData(String)	Checks if data was passed to a notification instance as specified by a SetData method call.
Id([Guid])	Specifies the identifier for a notification.
Message([String])	Specifies the content of the notification.
Recall()	Recall a sent notification.
Scope([NotificationScope])	Specifies the context in which the notification appears in the client.
Send()	Sends the notification to the client, where it will display in the UI.
SetData(String, String)	Specifies a data property value for the notification. The data is specified as text in a key-value pair.

See Also

Notifications
Getting Started with AL
Developing Extensions

Notification.AddAction Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies an action for the notification.

Syntax

```
Notification.AddAction(Caption: String, CodeunitID: Integer, MethodName: String)
```

Parameters

Notification Type: Notification An instance of the Notification data type.

Caption

Type: String

The text string that appears as the caption of the action in the notification UI. The string can be a text constant that is enabled for multilanguage functionality.

CodeunitID

Type: Integer

The ID of the Codeunit to run when the action is initiated from the notification UI. The codeunit should contain at least one global method to be called by the notification action. The global method must have a Notification data type parameter for accepting the notification object.

MethodName

Type: String

The name of the method in the Codeunit, which is specified by the CodeunitID parameter, that you want to run for the action.

Remarks

An action provides a way for you to add an interactive notification that enables users to respond to or take action on the notification. The method that is called by the action contains logic that you want to run for the action.

For more information and a detailed example, see Notifications.

Example

The following code creates two actions for a notification. The actions call the **RunAction1** and **RunAction2** methods in the codeunit **Action Handler**.

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.AddAction('Action 1',CodeUnit::"Action Handler",'RunAction1');
MyNotification.AddAction('Action 2',CodeUnit::"Action Handler",'RunAction2');
MyNotification.Send;
```

To handle the actions, the **Action Handler** codeunit has two global methods that have a **Notification** data type parameter:

```
procedure RunAction1@1(MyNotification@1000 : Notification);
begin
    Message('This is RunAction1');
end;

procedure RunAction2@2(MyNotification@1000 : Notification);
begin
 Message('This is RunAction2');
end;
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.GetData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves data that was passed to a notification instance as specified by a SetData method call.

Syntax

```
Value :=   Notification.GetData(Name: String)
```

Parameters

Notification Type: Notification An instance of the Notification data type.

Name
Type: String
The name of the data item that is specified by the SetData method call.

Return Value

Value Type: String The data retrieved

Remarks

You use the SetData Method (Notification) and GetData methods for transferring data in a notification. The methods are typically needed for handling actions on the notification. The SetData Method (Notification) method is called from the source of the notification, while the GetData method is called from the action code.

For more information and a detailed example, see Notifications.

Example

The following code sets the data for a notification:

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.SetData('Created',Format(CurrentDateTime,0,9));
MyNotification.SetData('ID',Format(CreateGUID,0,9));
MyNotification.AddAction('Action 1',CodeUnit::"Action Handler",'RunAction1');
MyNotification.AddAction('Action 2',CodeUnit::"Action Handler",'RunAction2');
MyNotification.Send;
```

The following code gets the data for a notification:

```
MyNotification.GetData('Created');
MyNotification.GetData('ID');
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.HasData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks if data was passed to a notification instance as specified by a SetData method call.

Syntax

```
Value :=   Notification.HasData(Name: String)
```

Parameters

Notification Type: Notification An instance of the Notification data type.

Name

Type: String

The name of the data item that is specified by the SetData method call.

Return Value

Value Type: Boolean **true**, if there is data; otherwise **false**.

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.Id Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the identifier for a notification.

Syntax

```
[Id := ]  Notification.Id([Id: Guid])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Notification Type: Notification An instance of the Notification data type.

Id
Type: Guid

Return Value

Id Type: Guid

Remarks

If left unassigned the notification will be assigned an ID when the Send method is called. For more information, see Send Method (Notification).

Example

The following code creates a notification and sends it if NewBalance is greater than the credit limit. If it is lower than the credit limit, it recalls the notification.

The example uses a pre-defined ID so that the notification can be recalled.

```
MyNotification.ID := '00000000-0000-0000-0000-000000000001';
IF NewBallance > Rec. "Credit Limit" THEN begin
  MyNotification.Message := 'The customer's current balance exceeds their credit limit.';
  MyNotification.Scope := NotificationScope::LocalScope;
  MyNotification.AddAction('Fix it.', 50001, 'FixCustomerCreditLimit');
  MyNotification.SetData('CustomerNo.', Rec."No.");
  MyNotification.Send;
END ELSE
  MyNotification.Recall;
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.Message Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the content of the notification.

Syntax

```
[Message := ]  Notification.Message([Message: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Notification Type: Notification An instance of the Notification data type.

Message

Type: String

Displays a text string in a message window.

Return Value

Message Type: String The message

Remarks

The Message method defines the notification. You use the Send Method (Notification) to send the notification to the client, where it will be displayed.

For more information and a detailed example, see Notifications.

Example

The following code creates a notification and sends it in the local scope.

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.Send;
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.Recall Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Recall a sent notification.

Syntax

```
[Ok := ]  Notification.Recall()
```

Parameters

Notification Type: Notification An instance of the Notification data type.

Return Value

Ok Type: Boolean **true** if it succeeds in sending a recall request to the client; otherwise **false**. The same notification can be recalled more than once, without failing. Also, a notification can be recalled successfully even if it hasn't been sent. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

A typical reason that the RECALL method returns **false** is a failure in the communication with the client. Another reason could be that the code tries to recall a notification for which there is no instance.

Example

The following code creates a notification and sends it if NewBalance is greater than the credit limit. If it is lower than the credit limit, it recalls the notification.

```
MyNotification.ID := '00000000-0000-0000-0000-000000000001';
IF NewBalance > Rec. "Credit Limit" then begin
  MyNotification.Message := 'The customer's current balance exceeds their credit limit.';
  MyNotification.Scope := NotificationScope::LocalScope;
  MyNotification.AddAction('Fix it.', 50001, 'FixCustomerCreditLimit');
  MyNotification.SetData('CustomerNo.', Rec."No.");
  MyNotification.Send;
end else
  MyNotification.Recall;
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.Scope Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the context in which the notification appears in the client.

Syntax

```
[Scope := ]  Notification.Scope([Scope: NotificationScope])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Notification Type: Notification An instance of the Notification data type.

Scope

Type: NotificationScope

The scope in which the notification appears in the client

Return Value

Scope Type: NotificationScope The scope of the current notification.

Remarks 1

For more information and a detailed example, see Notifications.

Example 1

The following code creates a notification and sends it in the local scope.

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.Send;
```

Remarks 2

The data that is specified by the SetData method can be retrieved by the GetData Method. The SetData and GetData methods are typically used for actions with actions on the notification. The SetData method is called from the source is the notification, while the GetData method is called from the action code.

You can use multiple SetData method calls to specify different data items. The data remains available for the life of the notification instance. The data is cleared once the notification instance has been dismissed or an action is taken.

For more information and a detailed example, see Notifications.

Example 2

The following code sets the data for a notification:

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.SetData('Created',Format(CurrentDateTime,0,9));
MyNotification.SetData('ID',Format(CreateGUID,0,9));
MyNotification.AddAction('Action 1',CodeUnit::"Action Handler",'RunAction1');
MyNotification.AddAction('Action 2',CodeUnit::"Action Handler",'RunAction2');
MyNotification.Send;
```

The following code gets the data for a notification:

```
MyNotification.GetData('Created');
MyNotification.GetData('ID');''
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions''

Notification.Send Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sends the notification to the client, where it will display in the UI.

Syntax

```
[Ok := ]  Notification.Send()
```

Parameters

Notification Type: Notification An instance of the Notification data type.

Return Value

Ok Type: Boolean **true** if the notification was sent; otherwise **false** If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The Send method displays the content of the notification that is specified by the Message Method.

For more information and a detailed example, see Notifications.

Example

The following code creates a notification and sends it to the client in the local scope.

```
MyNotification.Message := 'This is a notification';
MyNotification.Scope := NotificationScope::LocalScope;
MyNotification.Send;
```

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

Notification.SetData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies a data property value for the notification. The data is specified as text in a key-value pair.

Syntax

```
Notification.SetData(Name: String, Value: String)
```

Parameters

Notification Type: Notification An instance of the Notification data type.

Name

Type: String

The text string to use as a unique identifier for the data item.

Value

Type: String

The text string that represents the data.

See Also

Notification Data Type
Getting Started with AL
Developing Extensions

NumberSequence Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Is a complex data type for creating and managing number sequences in the database.

The following methods are available on the NumberSequence data type.

METHOD NAME	DESCRIPTION
Current(String [, Boolean])	Gets the current value from the number sequence, without doing any increment. The value is retrieved out of transaction. The value will not be returned on transaction rollback.
Delete(String [, Boolean])	Deletes a specific number sequence.
Exists(String [, Boolean])	Checks whether a specific number sequence exists.
Insert(String [, BigInteger] [, BigInteger] [, Boolean])	Creates a number sequence in the database, with the given parameters.
Next(String [, Boolean])	Retrieves the next value from the number sequence.

See Also

Getting Started with AL
Developing Extensions

NumberSequence.Current Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the current value from the number sequence, without doing any increment. The value is retrieved out of transaction. The value will not be returned on transaction rollback.

Syntax

```
Current :=   NumberSequence.Current(Name: String [, CompanySpecific: Boolean])
```

Parameters

Name
Type: String
Specifies the name of the number sequence.
CompanySpecific
Type: Boolean
Specifies if the number sequence is company-specific. Default is true.

Return Value

Current Type: BigInteger Returns the current value from number sequence.

Example

The following example gets the current value for the number sequence `MyNumberSequence`, which is not company specific.

```
number := NumberSequence.Current('MyNumberSequence', false);
```

See Also

NumberSequence Data Type
Getting Started with AL
Developing Extensions

NumberSequence.Delete Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Deletes a specific number sequence.

Syntax

```
NumberSequence.Delete(Name: String [, CompanySpecific: Boolean])
```

Parameters

Name

Type: String

Specifies the name of the number sequence.

CompanySpecific

Type: Boolean

Specifies if the number sequence is company-specific. Default is true.

Example

The following example checks whether the number sequence `MyNumberSequence` exists, and if so, it deletes it.

```
if NumberSequence.Exists('MyNumberSequence', false) then
    NumberSequence.Delete('MyNumberSequence', false);
```

See Also

NumberSequence Data Type
Getting Started with AL
Developing Extensions

NumberSequence.Exists Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Checks whether a specific number sequence exists.

Syntax

```
Exists :=   NumberSequence.Exists(Name: String [, CompanySpecific: Boolean])
```

Parameters

Name

Type: String

Specifies the name of the number sequence.

CompanySpecific

Type: Boolean

Specifies if the number sequence is company-specific. Default is true.

Return Value

Exists Type: Boolean Returns true if the number sequence exists.

Example

The following example checks whether the number sequence `MyNumberSequence` exists, and if so, it deletes it.

```
if NumberSequence.Exists('MyNumberSequence', false) then
    NumberSequence.Delete('MyNumberSequence', false);
```

See Also

NumberSequence Data Type
Getting Started with AL
Developing Extensions

NumberSequence.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Creates a number sequence in the database, with the given parameters.

Syntax

```
 NumberSequence.Insert(Name: String [, Seed: BigInteger] [, Increment: BigInteger] [, CompanySpecific:
Boolean])
```

Parameters

Name

Type: String

Specifies the name of the number sequence.

Seed

Type: BigInteger

Specifies the first value coming from the number sequence. Default is 0.

Increment

Type: BigInteger

The increment value used for the number sequence. Default is 1.

CompanySpecific

Type: Boolean

Specifies if the number sequence is company-specific. Default is true.

Example

The following example creates the number sequence `MyNumberSequence` that starts at zero and increments by a value of ten. The number series is not company specific.

```
NumberSequence.Insert('MyNumberSequence', 0, 10, false);
```

See Also

NumberSequence Data Type
Getting Started with AL
Developing Extensions

NumberSequence.Next Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Retrieves the next value from the number sequence.

Syntax

```
Next :=   NumberSequence.Next(Name: String [, CompanySpecific: Boolean])
```

Parameters

Name

Type: String

Specifies the name of the number sequence.

CompanySpecific

Type: Boolean

Specifies if the number sequence is company-specific. Default is true.

Return Value

Next Type: BigInteger Returns the next value from number sequence.

Example

The following example gets the next value in the number sequence `MyNumberSequence`. The number series is not company specific.

```
number := NumberSequence.Next('MyNumberSequence', false);
```

See Also

NumberSequence Data Type
Getting Started with AL
Developing Extensions

SessionInformation Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 4.0.*

Is a complex data type for exposing Session information into AL.

The following methods are available on the SessionInformation data type.

METHOD NAME	DESCRIPTION
SqlRowsRead()	Gets the amount of SQL rows read on the session, since the session started.
SqlStatementsExecuted()	Gets the amount of SQL statements executed on the session, since the session started.

See Also

Getting Started with AL
Developing Extensions

SessionInformation.SqlRowsRead Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the amount of SQL rows read on the session, since the session started.

Syntax

```
SqlStatementsExecuted :=   SessionInformation.SqlRowsRead()
```

NOTE

This method can be invoked using property access syntax.

Return Value

SqlStatementsExecuted Type: BigInteger The amount of SQL rows read on the session, since the session started.

Remarks

AL debugger also lets you monitor the number of SQL rows read. For more information, see Debugging SQL behavior.

Example

The following code gets the number of SQL rows read for the session and displays the number in a message.

```
var
    SqlRowsRead := BigText;

begin

    SqlRowsRead := SessionInformation.SqlRowsRead();
    Message(Format(SqlRowsRead));
end;
```

See Also

SessionInformation Data Type
Getting Started with AL
Developing Extensions

SessionInformation.SqlStatementsExecuted Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the amount of SQL statements executed on the session, since the session started.

Syntax

```
SqlStatementsExecuted :=   SessionInformation.SqlStatementsExecuted()
```

> **NOTE**
> This method can be invoked using property access syntax.

Return Value

SqlStatementsExecuted Type: BigInteger The amount of SQL statements executed on the session, since the session started.

Remarks

AL debugger also lets you to monitor the number of SQL statements executed. For more information, see Debugging SQL behavior.

Example

The following code gets the number of SQL statements executed and displays the number in a message.

```
var
    SqlStatementsExecuted : BigText;

begin

    SqlStatementsExecuted := SessionInformation.SqlStatementsExecuted();
    Message(Format(SqlStatementsExecuted));
end;
```

See Also

SessionInformation Data Type
Getting Started with AL
Developing Extensions

Option Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes an option value. In the code snippet below, you can see how the Option data type is declared.

Syntax example

```
procedure HelloWithOptions(OptionParameter : Option Alpha, "Bra-vo")
    var
        OptionVariable : Option C, "or D";
    begin
        Message('%1',OptionParameter::Alpha);
        Message('%1',OptionVariable::C);
    end;
```

NOTE

It is not possible to reference the members of the `OptionParameter` from outside the body of the procedure.

Remarks

In the OptionString Property of the field or variable, you can enter the option values as a comma-separated list. The Option type is a zero-based enumerator type, which means that the option values are assigned to sequential numbers, starting with 0. You can convert option data types to integers.

Example

This example shows how you can use the value of an option field as a constant in your AL code.

```
PurchHeaderRec."Document Type" := PurchHeaderRec."Document Type"::Invoice;
```

See Also

Getting Started with AL
Developing Extensions

OutStream Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a generic stream object that you can use to write to files and BLOBs.

The following methods are available on instances of the OutStream data type.

METHOD NAME	DESCRIPTION
Write(Variant [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Boolean [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Byte [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Char [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Integer [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(BigInteger [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Decimal [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Guid [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Text [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Code [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Label [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(TextConst [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(BigText [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Date [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.

METHOD NAME	DESCRIPTION
Write(Time [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(DateTime [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(DateFormula [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Duration [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Option [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Record [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(RecordId [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(String [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
Write(Any [, Integer])	Writes a specified number of bytes to the stream. Data is written in binary format.
WriteText([String] [, Integer])	Writes text to an OutStream object.

See Also

Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Variant [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Variant
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Var: Variant;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(Var);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Boolean [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Boolean
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Byte [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Byte
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Char [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Char
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Integer [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Integer
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(33);
    recBinaries.Modify();
end;
```

See Also

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: BigInteger [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: BigInteger
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Decimal [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Decimal
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

This example requires that you create the following variables.

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(0.33);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Guid [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Guid
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Guid: GUID;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(Guid);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Text [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Text
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Txt: Text;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(Txt);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Code [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Code
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Label [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Label
Contains the data to be written.
Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Lbl: Label 'Hello World';
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(lbl);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: TextConst [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: TextConst
Contains the data to be written.
Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

This example requires that you create the following variables.

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    TxtConst: Label 'Hello World';
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(TxtConst);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: BigText [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: BigText
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Date [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Date
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(19960106D);
    recBinaries.Modify();
end;
```

See Also

Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Time [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Time
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

This example requires that you create the following variables.

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Tme: Time;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(Tme);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: DateTime [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: DateTime
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    DateTme: DateTime;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(DateTme);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: DateFormula [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: DateFormula
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    DateForm: DateFormula;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(DateForm);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Duration [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Duration
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Write adds a zero byte at the end of the stream. This is differs from WriteText, which does not. For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Dur: Duration;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(Dur);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Option [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Option
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Opt: Option;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(Opt);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Record [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Record
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    Rec: Record Customer;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(recCust);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: RecordId [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: RecordId
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
    RecId: RecordID;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write(RecId);
    recBinaries.Modify();
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: String [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: String
Contains the data to be written.
Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
 var
    recBinaries: Record "Company Information";
    OStream: OutStream;
begin
    recBinaries.Find('-');
    recBinaries.Picture.CreateOutstream(OStream);
    OStream.Write('Hello World');
    recBinaries.Modify();
end;
```

See Also

OutStream.Write Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes a specified number of bytes to the stream. Data is written in binary format.

Syntax

```
[Written := ]  OutStream.Write(Value: Any [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Value
Type: Any
Contains the data to be written.

Length
Type: Integer
The number of bytes to be written. In the case of data types other than string, code, and binary, if you specify a length that differs from the size of the variable, an error message is displayed.

Return Value

Written Type: Integer The number of bytes that were written. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the optional return value, *Written*, is not specified and it was not possible to write all the data, an error message is displayed.

If the return value is present, you must verify that all the data was streamed.

Example

```
recBinaries.Find('-');
recBinaries.Data.CreateOutstream(OutStream);
OutStream.Write('Secretary');
OutStream.Write('Alice');
OutStream.Write('Hart');
OutStream.Write(19960106D);
recBinaries.Modify();
```

See Also

OutStream Data Type

OutStream.WriteText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Writes text to an OutStream object.

Syntax

```
[Written := ]  OutStream.WriteText([Text: String] [, Length: Integer])
```

Parameters

OutStream Type: OutStream An instance of the OutStream data type.

Text
Type: String
The text to write. If you do not specify this, a carriage return and a line feed are written.

Length
Type: Integer
The number of characters to be written.

Return Value

Written Type: Integer If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

For more information about how zero bytes and line endings are written and read, see Write, WriteText, Read, and ReadText Method Behavior Regarding Line Endings and Zero Terminators.

Example

This example also requires that the c:\TestFiles folder exists.

```
var
    MyHTMLFile: File;
    TestOutStream: OutStream;
begin
    MyHTMLFile.Create('c:\TestFiles\main.html');
    MyHTMLFile.CreateOutstream(TestOutStream);
    TestOutStream.WriteText('<html>');
    TestOutStream.WriteText;
    TestOutStream.WriteText('<head>');
    TestOutStream.WriteText('<title>My Page</title>');
    TestOutStream.WriteText('</head>');
    TestOutStream.WriteText;
    TestOutStream.WriteText('<P>Hello world!</p>');
    TestOutStream.WriteText;
    TestOutStream.WriteText('</html>');
    FileMyHTML.Close;
end;
```

See Also

OutStream Data Type
Getting Started with AL
Developing Extensions

Page Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Contains a number of simpler elements called controls. Controls are used to display information to the user or receive information from the user.

The following methods are available on the Page data type.

METHOD NAME	DESCRIPTION
GetBackgroundParameters()	Gets the page background task input parameters.
Run(Integer [, Record] [, Any])	Creates and launches a page that you specify. You can use Clear method to remove the page.
Run(Integer, Record, Integer)	Creates and launches a page that you specify. You can use Clear method to remove the page.
RunModal(Integer [, Record] [, Any])	Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.
RunModal(Integer, Record, Integer)	Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.
RunModal(Integer, Record, FieldRef)	Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.
SetBackgroundTaskResult(Dictionary of [Text, Text])	Sets the page background task result as a dictionary. When the task is completed, the OnPageBackgroundCompleted trigger will be invoked on the page with this result dictionary.

The following methods are available on instances of the Page data type.

METHOD NAME	DESCRIPTION
Activate([Boolean])	Activates the current page on the client if possible. The data on the page will not be refreshed.
CancelBackgroundTask(Integer)	Attempt to cancel a page background task.
Caption([String])	The caption shown in the title bar. For example, the default value in English (United States) is the same as the name of the page.
Close()	Closes the current page.

METHOD NAME	DESCRIPTION
Editable([Boolean])	Gets or sets the default editability of the page.
EnqueueBackgroundTask(var Integer, Integer [, var Dictionary of [Text, Text]] [, Integer] [, PageBackgroundTaskErrorLevel])	Creates and queues a background task that runs the specified codeunit (without a UI) in a read-only child session of the page session. If the task completes successfully, the **OnPageBackgroundTaskCompleted** trigger is invoked. If an error occurs, the **OnPageBackgroundTaskError** trigger is invoked. If the page is closed before the task completes, or the page record ID on the task changed, the task is cancelled.
GetRecord(var Record)	Gets the current record of the page.
LookupMode([Boolean])	Gets or sets the default lookup mode for the page.
ObjectId([Boolean])	Returns a string in the "Page xxx" format, where xxx is the caption or ID of the application object.
Run()	Creates and launches a page that you specify. You can use Clear method to remove the page.
RunModal()	Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.
SaveRecord()	Saves the current record as if performed by the client. If the record does not exist it is inserted, otherwise it is modified.
SetRecord(var Record)	Sets the current record for the page.
SetSelectionFilter(var Record)	Notes the records that the user has selected on the page, marks those records in the table specified, and sets the filter to "marked only".
SetTableView(var Record)	Applies the table view on the current record as the table view for the page, report, or XmlPort.
Update([Boolean])	Saves the current record and then updates the controls on the page. If you set the SaveRecord parameter to false, this method will not save the record before the page is updated.

See Also

Getting Started with AL
Developing Extensions

Page.GetBackgroundParameters Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the page background task input parameters.

Syntax

```
Parameters :=   Page.GetBackgroundParameters()
```

Return Value

Parameters Type: Dictionary of [Text, Text] The input parameters of the page background task.

Remarks

You use this method in a page background task codeunit, which is the codeunit that runs in a page background task. When a page background task is enqueued by the EnqueueBackgroundTask, it can include a set of parameters (a collection of key and value pairs) that can be used in the computations done in the background task codeunit. These parameters are passed as a dictionary of text to the codeunit's OnRun trigger when the page background task session is started. You use the GetBackgroundParameters method to retrieve these parameters.

Use the Evaluate method to convert parameter values to the data types required for calculations.

Example

The following code is an example of a page background task codeunit that uses the GetBackgroundParameters method to retrieve the value of the key named `Wait`, which is passed to the OnRun trigger from the calling page when the page background task is enqueued. For more details about this example, see Page Background Tasks.

```
codeunit 50100 PBTWaitCodeunit
{
    trigger OnRun()
    var
        Result: Dictionary of [Text, Text];
        StartTime: Time;
        WaitParam: Text;
        WaitTime: Integer;
        EndTime: Time;
    begin
        if not Evaluate(WaitTime, Page.GetBackgroundParameters().Get('Wait')) then
            Error('Could not parse parameter WaitParam');

        StartTime := System.Time();
        Sleep(WaitTime);
        EndTime := System.Time();

        Result.Add('started', Format(StartTime));
        Result.Add('waited', Format(WaitTime));
        Result.Add('finished', Format(EndTime));

        Page.SetBackgroundTaskResult(Result);
    end;
}
```

See Also

Page Background Tasks
Page Data Type
Getting Started with AL
Developing Extensions

Page.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates and launches a page that you specify. You can use Clear method to remove the page.

Syntax

```
Page.Run(Number: Integer [, Record: Record] [, Field: Any])
```

Parameters

Number

Type: Integer

The number of the page that you want to run. If you enter zero (0), the system displays the default lookup window for the current page. If the page you specify does not exist, a run-time error occurs.

Record

Type: Record

The record last displayed on the page. For each object, the system stores information about the most recently displayed record and the attached key and filters. Use this optional parameter to select a specific record to display on the page. The record must be of the same type as the table attached to the window. When the record is displayed, the key and filters attached to the record are used.

Field

Type: Any

Use this optional parameter to select a specific field on which focus will be placed.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates and launches a page that you specify. You can use Clear method to remove the page.

Syntax

```
Page.Run(Number: Integer, Record: Record, FieldNo: Integer)
```

Parameters

Number

Type: Integer

The number of the page that you want to run. If you enter zero (0), the system displays the default lookup window for the current page. If the page you specify does not exist, a run-time error occurs.

Record

Type: Record

The record last displayed on the page. For each object, the system stores information about the most recently displayed record and the attached key and filters. Use this optional parameter to select a specific record to display on the page. The record must be of the same type as the table attached to the window. When the record is displayed, the key and filters attached to the record are used.

FieldNo

Type: Integer

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.

Syntax

```
[Action := ]  Page.RunModal(Number: Integer [, Record: Record] [, Field: Any])
```

Parameters

Number

Type: Integer

The number of the page that you want to run.

Record

Type: Record

By default, this method shows the record that was last displayed on the page. For each object, information is stored about the most recently shown record and the attached key and filters. Use this optional parameter to select a specific record to display on the page. The record must be of the same type as the table that is attached to the page. When the record is displayed, the key and filters that are attached to the record are used.

Field

Type: Any

Use this optional parameter to select a specific field which will be in focus.

Return Value

Action Type: Action Specifies what action the user took on the page.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.

Syntax

```
[Action := ]  Page.RunModal(Number: Integer, Record: Record, FieldNo: Integer)
```

Parameters

Number

Type: Integer

The number of the page that you want to run.

Record

Type: Record

By default, this method shows the record that was last displayed on the page. For each object, information is stored about the most recently shown record and the attached key and filters. Use this optional parameter to select a specific record to display on the page. The record must be of the same type as the table that is attached to the page. When the record is displayed, the key and filters that are attached to the record are used.

FieldNo

Type: Integer

Use this optional parameter to select a specific field on which focus will be put.

Return Value

Action Type: Action Specifies what action the user took on the page.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.

Syntax

```
[Action := ]  Page.RunModal(Number: Integer, Record: Record, FieldRef: FieldRef)
```

Parameters

Number

Type: Integer

The number of the page that you want to run.

Record

Type: Record

By default, this function shows the record that was last displayed on the page. For each object, information is stored about the most recently shown record and the attached key and filters. Use this optional parameter to select a specific record to display on the page. The record must be of the same type as the table that is attached to the page. When the record is displayed, the key and filters that are attached to the record are used.

FieldRef

Type: FieldRef

Use this parameter to select a specific field on which focus will be put.

Return Value

Action Type: Action Specifies what action the user took on the page.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.SetBackgroundTaskResult Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Sets the page background task result as a dictionary. When the task is completed, the OnPageBackgroundCompleted trigger will be invoked on the page with this result dictionary.

Syntax

```
Page.SetBackgroundTaskResult(Results: Dictionary of [Text, Text])
```

Parameters

Results

Type: Dictionary of [Text, Text]

Specifies the dictionary of results for the page background task.

Remarks

You use this method in a page background task codeunit to pass the results of the page background task codeunit to the calling page. Before calling this method, use the Add method to populate the `Results` dictionary with the key-value pairs that you want to pass to calling page's OnPageBackgroundTaskCompleted trigger. The `OnPageBackgroundCompleted` trigger can then handle the results on the calling page, such as updating the record in the UI or database.

Example

The following code is an example of a page background task codeunit that uses the SetBackgroundTaskResult method to set a dictionary of results that will be passed to the calling page. For more details about this example, see Page Background Tasks.

```
codeunit 50100 PBTWaitCodeunit
{
    trigger OnRun()
    var
        Result: Dictionary of [Text, Text];
        StartTime: Time;
        WaitParam: Text;
        WaitTime: Integer;
        EndTime: Time;
    begin
        if not Evaluate(WaitTime, Page.GetBackgroundParameters().Get('Wait')) then
            Error('Could not parse parameter WaitParam');

        StartTime := System.Time();
        Sleep(WaitTime);
        EndTime := System.Time();

        Result.Add('started', Format(StartTime));
        Result.Add('waited', Format(WaitTime));
        Result.Add('finished', Format(EndTime));

        Page.SetBackgroundTaskResult(Result);
    end;
}
```

See Also

Page Background Tasks
Page Data Type
Getting Started with AL
Developing Extensions

Page.Activate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Activates the current page on the client if possible. The data on the page will not be refreshed.

Syntax

```
[Ok := ]  Page.Activate([Refresh: Boolean])
```

Parameters

Page Type: Page An instance of the Page data type.

Refresh

Type: Boolean

If set to **true**, the data on the page will be refreshed.

Return Value

Ok Type: Boolean

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.CancelBackgroundTask Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Attempt to cancel a page background task.

Syntax

```
[Ok := ]  Page.CancelBackgroundTask(TaskId: Integer)
```

Parameters

Page Type: Page An instance of the Page data type.

TaskId
Type: Integer
Specifies the ID of the page background task to cancel. The ID is assigned to the task when it is queued by the EnqueueBackgroundTask method.

Return Value

Ok Type: Boolean **true** if the page background task was marked for cancellation; otherwise **false**.

Example

The following example uses the CancelBackgroundTask method to cancel an existing page background task, based on its task ID.

```
var
  WaitTaskId: Integer;

trigger OnAfterGetRecord()
var
  TaskParameters: Dictionary of [Text, Text];
begin
  if (WaitTaskId > 0) then
    Currpage.CancelBackgroundTask(WaitTaskId);
    TaskParameters.Add('Wait', '1000');
    CurrPage.EnqueueBackgroundTask(WaitTaskId, 50100, TaskParameters, 1000,
PageBackgroundTaskErrorLevel::Warning);
end;
```

See Also

Page Background Tasks
Page Data Type
Getting Started with AL
Developing Extensions

Page.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

The caption shown in the title bar. For example, the default value in English (United States) is the same as the name of the page.

Syntax

```
[Caption := ]  Page.Caption([NewCaption: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Page Type: Page An instance of the Page data type.

NewCaption
Type: String
The new caption text.

Return Value

Caption Type: String The text used for the caption.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes the current page.

Syntax

```
Page.Close()
```

Parameters

Page Type: Page An instance of the Page data type.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.Editable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the default editability of the page.

Syntax

```
[Editable := ]  Page.Editable([NewEditable: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Page Type: Page An instance of the Page data type.

NewEditable

Type: Boolean

The new default editability of the page.

Return Value

Editable Type: Boolean Indicates the editability of the page.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.EnqueueBackgroundTask Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Creates and queues a background task that runs the specified codeunit (without a UI) in a read-only child session of the page session. If the task completes successfully, the **OnPageBackgroundTaskCompleted** trigger is invoked. If an error occurs, the **OnPageBackgroundTaskError** trigger is invoked. If the page is closed before the task completes, or the page record ID on the task changed, the task is cancelled.

Syntax

```
[Ok := ]  Page.EnqueueBackgroundTask(var TaskId: Integer, CodeunitId: Integer [, var Parameters: Dictionary
of [Text, Text]] [, Timeout: Integer] [, ErrorLevel: PageBackgroundTaskErrorLevel])
```

Parameters

Page Type: Page An instance of the Page data type.

TaskId

Type: Integer

Specifies the ID of the new page background task. The ID is assigned to the TaskId variable after the task is queued successfully. This parameter is passed by reference to the method.

CodeunitId

Type: Integer

Specifies the ID of the codeunit to run when the task is started.

Parameters

Type: Dictionary of [Text, Text]

Specifies a collection of keys and values that are passed to the OnRun trigger of the codeunit that runs when the page background task session is started.

Timeout

Type: Integer

Specifies the number of milliseconds that the page background task can run before it is automatically cancelled.

ErrorLevel

Type: PageBackgroundTaskErrorLevel

Specifies the level of error handling on page background task level.

Return Value

Ok Type: Boolean **true** if the page background task is successfully queued for execution; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The enqueued page background task stores the record ID of the current page. If the current record ID on the page changes, or the page is closed, the task is canceled. Typically, you call the EnqueueBackgroundTask method from a page trigger. The ID of the current record of the page must remain static after the call is made and while the background task is running. Otherwise, the task will be canceled. For this reason, we recommend that you

don't enqueue the background task from the `OnOpenPage` trigger. Instead, use the `OnAfterGetRecord` or `OnAfterGetCurrRecord` triggers.

The **Child Session Max Concurrency** setting of the Business Central Server controls how many page background tasks can be run simultaneously for a parent session. The setting has a default value of 5. If this number is exceeded, then they'll be queued and run when a slot becomes available as other tasks are finished. Enqueuing the task will fail if the total number of enqueued tasks exceed the **Child Sessions Max Queue Length** server configuration setting. For more information, see Configuring Business Central Server - Asynchronous Processing.

Timeout

When the value of the *Timeout* parameter is exceeded, the background task is canceled and an error with error code ChildSessionTaskTimeout occurs. On the page, the error will appear as a notification.

The Business Central Server instance includes two configuration settings related to the page back ground task timeout: PageBackgroundTaskDefaultTimeout and PageBackgroundTaskMaxTimeout.

- The PageBackgroundTaskDefaultTimeout (which has a default value of 00:02:00) determines the timeout if the *Timeout* parameter isn't given a value.
- The PageBackgroundTaskMaxTimeout specifies the maximum amount of time that a page background task can run. It doesn't matter what the *Timeout* parameter value is. If the *Timeout* value is greater than the PageBackgroundTaskMaxTimeout, which has a default value of 00:10:00, the PageBackgroundTaskMaxTimeout value determines the timeout.

For more information these settings, see Configuring Business Central Server.

It's possible to enqueue the task again in the completion trigger or error trigger, but this pattern isn't recommended as it can lead to an endless loop. For more information, see Page Background Tasks.

Example

The following code extends the **Customer Card** page with a page background task by using the EnqueueBackgroundTask method. For more information about this example, see Page Background Tasks.

```
pageextension 50100 CustomerCardExt extends "Customer Card"
{
    layout
    {
        addlast(General)
        {

            field(Before1; before1)
            {
                ApplicationArea = All;
                Caption = 'Before 1';
                Editable = false;
            }

            field(Duration1; duration1)
            {
                ApplicationArea = All;
                Caption = 'Duration 1';
                Editable = false;
            }

            field(After1; after1)
            {
                ApplicationArea = All;
                Caption = 'After 1';
                Editable = false;
            }
        }
    }

    var
        // Global variable used for the TaskID
        WaitTaskId: Integer;

        // Variables for the three fields on the page
        before1: Text;
        duration1: Text;
        after1: Text;

    trigger OnAfterGetRecord();
    var
        //Defines a variable for passing parameters to the background task
        TaskParameters: Dictionary of [Text, Text];
    begin
        // Adds a key-value pair to the parameters dictionary
        TaskParameters.Add('Wait', '1000');

        //Enqueues the page background task
        CurrPage.EnqueueBackgroundTask(WaitTaskId, 50100, TaskParameters, 10000,
PageBackgroundTaskErrorLevel::Warning);
    end;
```

See Also

Page Background Tasks
Page Data Type
Getting Started with AL
Developing Extensions

Page.GetRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current record of the page.

Syntax

```
Page.GetRecord(var Record: Record)
```

Parameters

Page Type: Page An instance of the Page data type.

Record

Type: Record

The Record variable that will contain the current record associated with the page.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.LookupMode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the default lookup mode for the page.

Syntax

```
[LookupMode := ]  Page.LookupMode([NewLookupMode: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Page Type: Page An instance of the Page data type.

NewLookupMode
Type: Boolean
The new default lookup mode for the page.

Return Value

LookupMode Type: Boolean The current default lookup mode for the page

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.ObjectId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a string in the "Page xxx" format, where xxx is the caption or ID of the application object.

Syntax

```
String :=   Page.ObjectId([UseNames: Boolean])
```

Parameters

Page Type: Page An instance of the Page data type.

UseNames
Type: Boolean
If **true**, the page caption is returned, else the page ID as text.

Return Value

String Type: String The text of the object

Example

If you add the following code to a page method or trigger, then the returned string is displayed in a message window.

```
Message(CurrPage.ObjectId(true));
```

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates and launches a page that you specify. You can use Clear method to remove the page.

Syntax

```
Page.Run()
```

Parameters

Page Type: Page An instance of the Page data type.

Remarks

If, at design time, you do not know the specific page you want to run, then use this method or the Page.RunModal Method and specify the page in the *Number* parameter.

If you do know which page you want to run, then you can create a Page variable, set the subtype of the variable to a specific page, and then use the Run Method (Page) or RunModal Method (Page) on the Page variable.

When you want to close the page, use CurrPage.Close. CurrPage is a predefined system variable.

Example

```
Page.Run(4711)
```

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates, opens, and closes a page that you specify. When a page is run modally, no input, such as a keyboard or mouse click, can occur except for objects on the modal page.

Syntax

```
[Action := ]  Page.RunModal()
```

Parameters

Page Type: Page An instance of the Page data type.

Return Value

Action Type: Action Specifies what action the user took on the page.

Specifies what action the user took on the page. The following table shows the possible return values for the different user actions.

RETURN VALUE	DESCRIPTION
OK	To close the page window, the user does one of the following: - Chooses the **Close** button. - Chooses the **X** button when there is no **Cancel** button on the window.
Cancel	To close the page window, the user does one of the following: - Chooses the **Cancel** button. - Chooses the **X** button when there is a **Cancel** button on the window.
LookupOK	To close a lookup window, the user chooses the **OK** button.
LookupCancel	To close a lookup window, the user chooses the **Cancel** button.
Yes	To close a confirmation window, the user selects **Yes**.
No	To close a confirmation window, the user does one of the following: - Chooses the **No** button. - Chooses the **X** button.

RETURN VALUE	DESCRIPTION
RunObject	The user selected an option that ran another object.
RunSystem	The user selected an option that ran an external program.

Remarks

If you know the specific page that you want to run when you are designing your application, then you can create a Page variable, set the Subtype of the variable to a specific page, and then use this method or the Run Method (Page).

If you do not know the specific page that you want to run, then use the Run Method (Page) or the **RunModal Method (Page)** and specify the page in the *Number* parameter.

After you define the page variable, you can use it before and after you run the page. If you use the Run Method (Page), then you can only use the variable before you run the page.

Example

This example shows how to use this method. Assume that the *SomePage* variable has been defined as `Page 1`.

```
Clear(SomePage);
SomePage.XXX; // Any user-defined method
SomePage.SetTableView(MyRecord);
SomePage.SetRecord(MyRecord);
if SomePage.RunModal = Action::LookupOK then
  SomePage.GetRecord(MyRecord)...
```

NOTE

This code example includes the Clear Method to make sure that the variable has been cleared.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.SaveRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the current record as if performed by the client. If the record does not exist it is inserted, otherwise it is modified.

Syntax

```
Page.SaveRecord()
```

Parameters

Page Type: Page An instance of the Page data type.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.SetRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current record for the page.

Syntax

```
Page.SetRecord(var Record: Record)
```

Parameters

Page Type: Page An instance of the Page data type.

Record
Type: Record
The record to set as the current record. You cannot use a temporary record for the Record parameter.

Remarks

You can use this method to set the record to display when the user opens the page.

Example

The following example retrieves the record that has a primary key value of '30000' from the Customer table. If the record is found, it is stored in the MyRecord variable. The **SetRecord** method uses the retrieved record as the current record and sets record for MyPage, which is a Customer Card page. When the code unit is run, the record is displayed on the MyPage page. If the record is not found, a message box displays a message that indicates that the record was not found.

```
 var
    MyPage: Page "Customer Card";
    MyRecord: Record Customer;
    Text000: Label 'The record was not found';
begin
    if MyRecord.Get('30000') then begin
      MyPage.SetRecord(MyRecord);
      MyPage.Run;
    end else begin
      Message(Text000);
    end;
end;
```

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.SetSelectionFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Notes the records that the user has selected on the page, marks those records in the table specified, and sets the filter to "marked only".

Syntax

```
Page.SetSelectionFilter(var Record: Record)
```

Parameters

Page Type: Page An instance of the Page data type.

Record
Type: Record

Remarks

If all records are selected, marks will not be used.

If only the current record is selected on the page, then SetSelectionFilter does the following:

- Sets the current filter group to `0` on the destination record
- Adds filters on the primary key fields that point to the current record of the page

If more than one record is selected on the page, then SetSelectionFilter does the following:

- Copies the current key from the page source table to the destination record
- Copies the current sort order from the table to the destination record
- Copies the current filters that are set in all filter groups
- Copies the current filter group
- Marks the selected records and sets the "marked only" filter

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.SetTableView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies the table view on the current record as the table view for the page, report, or XmlPort.

Syntax

```
Page.SetTableView(var Record: Record)
```

Parameters

Page Type: Page An instance of the Page data type.

Record

Type: Record

The record that has a table view that you want to apply to the page or data item.

Remarks

The table view is the view of the table that you present to the user. You determine what records that the user can see by setting filters, determining the sorting order, and selecting the keys.

This method only narrows the view of the table that was set through the SourceTableView Property of the page or through the DataItemTableView Property of the data item.

> **IMPORTANT**
>
> SetTableView is not supported for setting views on subpages from code on table headers. For example, you cannot set a table view on the SalesOrder subpage from the SalesHeader.

Example

This example is based on the Sales Header table and shows how SetTableView is used for a page object.

```
 var
    SalesHeader: Record "Sales Header";
    SomePage: Page "Sales List";
begin
    SalesHeader.SetCurrentKey("Document Type");
    SalesHeader.SetRange("Document Type",SalesHeader."Document Type"::Order);
    SomePage.SetTableView(SalesHeader); // Only view sales orders.
    SomePage.Run;
end;
```

The page that is reference by the SomePage variable can be any page object that has Sales Header as the value of the SourceTable Property.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

Page.Update Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the current record and then updates the controls on the page. If you set the SaveRecord parameter to false, this method will not save the record before the page is updated.

Syntax

```
Page.Update([SaveRecord: Boolean])
```

Parameters

Page Type: Page An instance of the Page data type.

SaveRecord
Type: Boolean
Set this parameter to true if you want to save the current record. Set this parameter to false if you want to update without saving the current record.

Remarks

The `SaveRecord` default value is true. The default value of SaveRecord, however, depends on whether SourceTable is specified or not. If SourceTable is not defined (== null), then the default value is `false`. If SourceTable is specified then the default value is `true`.

See Also

Page Data Type
Getting Started with AL
Developing Extensions

ProductName Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

An application can have a full name, marketing name, and short name. The PRODUCTNAME functions enable you to retrieve these name variations.

The following methods are available on the ProductName data type.

METHOD NAME	DESCRIPTION
Full()	FULL returns a text string that contains the application's full name.
Marketing()	MARKETING returns a text string that contains the application's marketing name.
Short()	SHORT returns a text string that contains the application's short name.

Remarks

You define the different name variations for an application in the `navsettings.json` configuration file. For more information, see Configuring Business Central Web Server Instances.

See Also

Getting Started with AL
Developing Extensions

ProductName.Full Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

FULL returns a text string that contains the application's full name.

Syntax

```
ProductName :=   ProductName.Full()
```

Return Value

ProductName Type: String Text of the product's full name.

Remarks

This method is useful when you have to include the application name in UI text. Instead of using static text for the name, you use one of the PRODUCTNAME methods. This lets you reuse the same text string across different applications, and makes it easier if the application is ever renamed.

You define the different name variations for an application in the `navsettings.json` configuration file. For more information, see Configuring Business Central Web Server Instances.

See Also

ProductName Data Type
Getting Started with AL
Developing Extensions
Configuring Business Central Web Server Instances

ProductName.Marketing Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

MARKETING returns a text string that contains the application's marketing name.

Syntax

```
ProductNameMarketing :=   ProductName.Marketing()
```

Return Value

ProductNameMarketing Type: String Text of the product's marketing name.

Remarks

This method is useful when you have to include the application name in UI text. Instead of using static text for the name, you use one of the ProductName methods. This lets you reuse the same text string across different applications, and makes it easier if the application is ever renamed.

You define the different name variations for an application in the `navsettings.json` configuration file. For more information, see Configuring Business Central Web Server Instances.

See Also

ProductName Data Type
Getting Started with AL
Developing Extensions
Configuring Business Central Web Server Instances

ProductName.Short Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

SHORT returns a text string that contains the application's short name.

Syntax

```
ProductNameShort :=   ProductName.Short()
```

Return Value

ProductNameShort Type: String Text of the product's short name.

Remarks

This method is useful when you have to include the application name in UI text. Instead of using static text for the name, you use one of the ProductName methods. This lets you reuse the same text string across different applications, and makes it easier if the application is ever renamed.

You define the different name variations for an application in the `navsettings.json` configuration file. For more information, see Configuring Business Central Web Server Instances.

See Also

ProductName Data Type
Getting Started with AL
Developing Extensions
Configuring Business Central Web Server Instances

Query Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Enables you to retrieve data from multiple tables and combine the data in single dataset.

The following methods are available on the Query data type.

METHOD NAME	DESCRIPTION
SaveAsCsv(Integer, String [, Integer] [, String])	Saves the resulting data set of a query as a comma-separated values (CSV) file.
SaveAsCsv(Integer, OutStream [, Integer] [, String])	Saves the resulting data set of a query as a comma separated values (CSV) file.
SaveAsXml(Integer, String)	Saves the resulting data set of a query as an .xml file.
SaveAsXml(Integer, OutStream)	Saves the resulting data set of a query as an .xml file.

The following methods are available on instances of the Query data type.

METHOD NAME	DESCRIPTION
Close()	Closes a query data set and returns the query instance to the initialized state. The following code shows the syntax of the Close method. Query is a variable of the Query data type that specifies the query object.
ColumnCaption(Any)	Returns the current caption of a query column as a text string.
ColumnName(Any)	Returns the name of a query column as a text string.
ColumnNo(Any)	Returns the ID that is assigned to a query column in the query definition.
GetFilter(Any)	Returns the filters that are set on the field of a specified column in the query. The following code shows the syntax of the GetFilter method. Query is a variable of the Query data type that specifies the query object.
GetFilters()	Returns the filters that are applied to all columns in the query. The following code shows the syntax of the GetFilterS method. Query is a variable of the Query data type that specifies the query object.
Open()	Runs a query object and generates a data set that can be read. The following code shows the syntax of the Open method. Query is a variable of the Query data type that specifies the query object.

METHOD NAME	DESCRIPTION
Read()	Reads data from a row in the resulting data set of a query.
SaveAsCsv(String [, Integer] [, String])	Saves the resulting data set of a query as comma separated values (CSV)
SaveAsCsv(OutStream [, Integer] [, String])	Saves the resulting data set of a query as comma separated values (CSV)
SaveAsXml(String)	Saves the resulting data set of a query as XML
SaveAsXml(OutStream)	Saves the resulting data set of a query as XML
SecurityFiltering([SecurityFilter])	Gets or sets how security filters are applied to the query.
SetFilter(Any, String [, Any,...])	Sets a filter on a column of a query to limit the records in the resulting data set of a query.
SetRange(Any [, Any] [, Any])	Sets a filter on a range of values on a column of a query data set.
TopNumberOfRows([Integer])	Specifies the maximum number of rows to include in the resulting data set of a query.

See Also

Getting Started with AL
Developing Extensions

Query.SaveAsCsv Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as a comma-separated values (CSV) file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Query.SaveAsCsv(Number: Integer, FileName: String [, Format: Integer] [, FormatArgument: String])
```

Parameters

Number
Type: Integer
The ID of the query object. If the query that you specify does not exist, then a run-time error occurs.

FileName
Type: String
The path and name of the file that you want to save the query data set to.

Format
Type: Integer
Specifies whether the columns of the resulting data set are at fixed positions in the CSV file or separated only by a delimiter.

FormatArgument
Type: String
You set the FormatArgument parameter based on the setting of the Format parameter. If the Format parameter is set to 0, then the FormatArgument parameter specifies the starting position of each column in the data set. The value is a comma separated string of integers that includes an integer for every column. In a CSV file, each line is evenly divided into positions for holding characters. The first integer corresponds to the starting position of the first column, the second integer corresponds to the starting position of the second column, and so on.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

A CSV file stores data in a plain text format. When you save a query dataset as a CSV file, each row of the dataset is stored on a separate line in the file, and each column in a row is separated by a comma or another delimiter of your choice. The first line of the file will contain the column names of the query. A query column

name is specified by its Name Property.

When the **SaveAsCSV** method is called the query dataset is generated and then saved in CSV format to the file that is designated by the *FileName* parameter.

To apply filters with the **SetFilter** and **SetRange** methods, the SaveAsCSV method must be called after **SetFilter** and **SetRange** methods, as shown in the following example.

```
// Sets a filter on the Quantity column of the query
Query.SetFilter(Quantity, '>50');

// Opens a new query that is filtered by the SetFilter method and saves the dataset
Query.SaveAsCSV('c:\test.csv');
```

The **SaveAsCSV** method can be called at any place in the code and does not require that the **Close**, **Open** or **Read** methods are called before it. When the **SaveAsCSV** method is called, a new instance of the query is created. The query is implicitly opened, read, and closed. If there is currently a dataset in the opened state when the **SaveAsCSV** method is called, then that instance is closed. This means that the following code is illegal because the query is not open on the second **Read** call.

```
Query.Open;
Query.Read;
Query.SaveAsCSV('c:\test.csv');
Query.Read;
```

The correct code for this example is as follows.

```
Query.Open;
Query.Read;
Query.SaveAsCSV('c:\test.csv');
Query.Open;
Query.Read;
```

Example

The following example shows how to save the dataset of a query with the name **My Customers Query** as a CSV file. The file is given the name **mycustomers.csv** and is saved on the c: drive of the computer running Dynamics 365 Business Central service. The query consists of three columns: No., Name, and City. The file is set to place the columns at the following positions: 1, 10, and 40.

This example requires that you create a query called **My Customer Query** that is based on table **18 Customer** and contains the **No**, **Name**, and **City** columns.

```
var
  MyCustomerQuery: Query "My Customer Query";
  OK: Boolean;
  Text000: Label 'Query was not saved.';
begin
    OK := MyCustomerQuery.SaveAsCSV('c:\mycustomers.csv', 0, '1,10,40');
    if not OK then
      Error(Text000);
end;
```

The following code shows an example of the content of the saved file.

```
No       Name                          City
01121212 Spotsmeyer's Furnishings      Miami
01445544 Progressive Home Furnishings  Chicago
01454545 New Concepts Furniture        Atlanta
01905893 Candoxy Canada Inc.           Thunder Bay
01905899 Elkhorn Airport               Elkhorn
01905902 London Candoxy Storage Campus London
```

If for some reason the file cannot be saved, then the message **Query was not saved.** appears.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsCsv Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as a comma separated values (CSV) file.

Syntax

```
[Ok := ]  Query.SaveAsCsv(Number: Integer, OutStream: OutStream [, Format: Integer] [, FormatArgument:
String])
```

Parameters

Number

Type: Integer

The ID of the query object. If the query that you specify does not exist, then a run-time error occurs.

OutStream

Type: OutStream

The stream that you want to save the query as CSV to.

Format

Type: Integer

Specifies whether the columns of the resulting data set are at fixed positions in the CSV file or separated only by a delimiter.

FormatArgument

Type: String

You set the FormatArgument parameter based on the setting of the Format parameter. If the Format parameter is set to 0, then the FormatArgument parameter specifies the starting position of each column in the data set. The value is a comma separated string of integers that includes an integer for every column. In a CSV file, each line is evenly divided into positions for holding characters. The first integer corresponds to the starting position of the first column, the second integer corresponds to the starting position of the second column, and so on.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

A CSV file stores data in a plain text format. When you save a query dataset as a CSV file, each row of the dataset is stored on a separate line in the file, and each column in a row is separated by a comma or another delimiter of your choice. The first line of the file will contain the column names of the query. A query column name is specified by its Name Property.

When the **SaveAsCSV** method is called the query dataset is generated and then saved in CSV format to the file that is designated by the *FileName* parameter.

To apply filters with the **SetFilter** and **SetRange** methods, the SaveAsCSV method must be called after **SetFilter** and **SetRange** methods, as shown in the following example.

```
// Sets a filter on the Quantity column of the query
Query.SetFilter(Quantity, '>50');

// Opens a new query that is filtered by the SetFilter method and saves the dataset
Query.SaveAsCSV('c:\test.csv');
```

The **SaveAsCSV** method can be called at any place in the code and does not require that the **Close**, **Open** or **Read** methods are called before it. When the **SaveAsCSV** method is called, a new instance of the query is created. The query is implicitly opened, read, and closed. If there is currently a dataset in the opened state when the **SaveAsCSV** method is called, then that instance is closed. This means that the following code is illegal because the query is not open on the second **Read** call.

```
Query.Open;
Query.Read;
Query.SaveAsCSV('c:\test.csv');
Query.Read;
```

The correct code for this example is as follows.

```
Query.Open;
Query.Read;
Query.SaveAsCSV('c:\test.csv');
Query.Open;
Query.Read;
```

Example

The following example shows how to save the dataset of a query with the name **My Customers Query** as a CSV file. The file is given the name **mycustomers.csv** and is saved on the c: drive of the computer running Dynamics 365 Business Central service. The query consists of three columns: No., Name, and City. The file is set to place the columns at the following positions: 1, 10, and 40.

This example requires that you create a query called **My Customer Query** that is based on table **18 Customer** and contains the **No**, **Name**, and **City** columns.

```
  var
    MyCustomerQuery: Query "My Customer Query";
    OK: Boolean;
    Text000: Label 'Query was not saved.';
begin
    OK := MyCustomerQuery.SaveAsCSV('c:\mycustomers.csv', 0, '1,10,40');
    if not OK then
          Error(Text000);
end;
```

The following code shows an example of the content of the saved file.

```
No       Name                          City
01121212 Spotsmeyer's Furnishings      Miami
01445544 Progressive Home Furnishings  Chicago
01454545 New Concepts Furniture        Atlanta
01905893 Candoxy Canada Inc.           Thunder Bay
01905899 Elkhorn Airport               Elkhorn
01905902 London Candoxy Storage Campus London
```

If for some reason the file cannot be saved, then the message **Query was not saved.** appears.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as an .xml file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Query.SaveAsXml(Number: Integer, FileName: String)
```

Parameters

Number
Type: Integer
The ID of the query object that you want to save as an .xml file. If the query that you specify does not exist, then a run-time error occurs.

FileName
Type: String
The path and name of the file that you want to save the query to.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

When the **SaveAsXML** method is called, the dataset is generated and then saved in XML format in the file and location that is designated by the *FileName* parameter.

The **SaveAsXML** method can be called at any place in the code and does not require that the **Close**, **Open**, or **Read** methods are called before it. When the **SaveAsXML** method is called, a new instance of the query is created. The query is implicitly opened, read, and closed. If there is currently a dataset in the opened state when the **SaveAsXML** method is called, then that instance is closed. This means that the following code is unauthorized because the query is not open on the second **Read** call.

```
Query.Open;
Query.Read;
Query.SaveAsXML('c:\test.xml');
Query.Read;
```

The correct code for this example is as follows.

```
Query.Open;
Query.Read;
Query.SaveAsXML('c:\test.xml');
Query.Open;
Query.Read;
```

Example

The following example shows how to save a query with the name **My Customer Query** as an .xml file. The file is given the name **myquery.xml** and is saved on the c: drive of the computer running Dynamics 365 Business Central service.

```
var
    MyCustomerQuery: Query "My Customer Query";
    OK: Boolean;
    Text000: Label 'Query was not saved.';
begin
    OK := MyCustomerQuery.SaveAsXML('c:\myquery.xml');
    if not OK then
      Error(Text000);
end;
```

If the file cannot be saved, then the follow message appears:

Query not saved.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as an .xml file.

Syntax

```
[Ok := ]  Query.SaveAsXml(Number: Integer, OutStream: OutStream)
```

Parameters

Number

Type: Integer

The ID of the query object that you want to save as an .xml file. If the query that you specify does not exist, then a run-time error occurs.

OutStream

Type: OutStream

The stream that you want to save the query as XML to.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

When the **SaveAsXML** method is called, the dataset is generated and then saved in XML format in the file and location that is designated by the *FileName* parameter.

The **SaveAsXML** method can be called at any place in the code and does not require that the **Close**, **Open**, or **Read** methods are called before it. When the **SaveAsXML** method is called, a new instance of the query is created. The query is implicitly opened, read, and closed. If there is currently a dataset in the opened state when the **SaveAsXML** method is called, then that instance is closed. This means that the following code is unauthorized because the query is not open on the second **Read** call.

```
Query.Open;
Query.Read;
Query.SaveAsXML('c:\test.xml');
Query.Read;
```

The correct code for this example is as follows.

```
Query.Open;
Query.Read;
Query.SaveAsXML('c:\test.xml');
Query.Open;
Query.Read;
```

Example

The following example shows how to save a query with the name **My Customer Query** as an .xml file. The file is given the name **myquery.xml** and is saved on the c: drive of the computer running Dynamics 365 Business Central service.

```
var
    MyCustomerQuery: Query "My Customer Query";
    OK: Boolean;
    Text000: Label 'Query was not saved.';
begin
    OK := MyCustomerQuery.SaveAsXML('c:\myquery.xml');
    if not OK then
          Error(Text000);
end;
```

If the file cannot be saved, then the follow message appears:

Query not saved.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes a query data set and returns the query instance to the initialized state. The following code shows the syntax of the Close method. Query is a variable of the Query data type that specifies the query object.

Syntax

```
Query.Close()
```

Parameters

Query Type: Query An instance of the Query data type.

Remarks

The **Close** method does not clear any filters that are set by the SetFilterS method. If you want to clear the filters, then you must call the Clear method.

In most cases, you do not have to call the **Close** method explicitly. The **Close** method is called implicitly when the following conditions are true:

- When the query variable goes out of scope.
- When the **Open** method is called on a dataset that is currently open.
- When the SetFilter Method (Query) or SetRange Method (Query) are called on a dataset that is currently open.

Example

The following example demonstrates how to use the **Close** method on a query. The example code sets filters on the query, opens the query, and then reads the dataset. For each row in the dataset, a message box is displayed that contains the values of the columns in the row.

This example requires that you create a query called **Customer_SalesQuantity** that is links table **18 Customer** with table **37 Sales Lines** from the CRONUS International Ltd. demonstration database. Include columns for the **Name** and **No.** fields from the Customer table and the **Quantity** field from Sales Lines table.

The following AL code opens the query, reads each row of the dataset, and then closes the query. You can add the code to a codeunit, and then run the codeunit to see the results.

```
var
  MyQuery: Query "Customer SalesQuantity";
  Text000: Label 'Customer name = %1, Quantity = %2';
begin
    // Sets a filter to display only sales quantities greater than 20.
    MyQuery.SetFilter(Quantity, '>20');
    // Runs the query.
    MyQuery.Open;
    // Reads each row in the dataset and displays a message with column values.
    // Stops reading when there are no more rows remaining in the dataset (Read is False).
    while MyQuery.Read do
    begin
      Message(Text000, MyQuery.Name, MyQuery.Quantity);
    end;
    MyQuery.Close;
end;
```

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.ColumnCaption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the current caption of a query column as a text string.

Syntax

```
Caption :=   Query.ColumnCaption(Column: Any)
```

Parameters

Query Type: Query An instance of the Query data type.

Column
Type: Any
Refers to the name of the query column. The name of a query column is specified by the Name Property.

Return Value

Caption Type: String The query column caption.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.ColumnName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the name of a query column as a text string.

Syntax

```
Name :=   Query.ColumnName(Column: Any)
```

Parameters

Query Type: Query An instance of the Query data type.

Column
Type: Any
Refers to the name of the query column. The name of a query column is specified by the Name Property.

Return Value

Name Type: String The query column name.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.ColumnNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the ID that is assigned to a query column in the query definition.

Syntax

```
Number :=   Query.ColumnNo(Column: Any)
```

Parameters

Query Type: Query An instance of the Query data type.

Column
Type: Any
Refers to the name of the query column. The name of a query column is specified by the Name Property.

Return Value

Number Type: Integer The ID of the query column.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.GetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the filters that are set on the field of a specified column in the query. The following code shows the syntax of the GetFilter method. Query is a variable of the Query data type that specifies the query object.

Syntax

```
Filter :=   Query.GetFilter(Column: Any)
```

Parameters

Query Type: Query An instance of the Query data type.

Column
Type: Any
The name of the column in the query. A column name is defined by the Name Property.

Return Value

Filter Type: String The filters of the column.

Remarks

The **GetFilter** method returns the filters that are currently set for a data column or filter row by the SetFilter Method (Query) method, SetRange Method (Query) method, and the column's ColumnFilter Property. The **GetFilter** method does not return filters that are set on a column's source field by the DataItemTableFilter Property or global filters that are set by the **FilterGroup** method.

You can call the **GetFilter** method multiple times and at any point in the code. If you call the **GetFilter** method before the **SetFilter** or **SetRange** method, then the **GetFilter** method returns only filters on the column that are set by the column's ColumnFilter property.

Filters that are set by the **SetFilter** method and **SetRange** method are applied to a query when the query is opened with a call to the **Open**, **SaveAsXML**, or **SaveAsCSV** methods. You must consider the location of the **GetFilterS** method with respect to these methods to obtain the desired results. For example, in the following two code examples, the **GetFilter** method will return the filter that is set by the **SetFilter** method. However, in the first example, the filter has been applied to the query dataset; in the second example, the filter has not been applied.

```
Query.SetFilter(Column, String);
Query.Open;
Query.GetFilter(Column);
Query.Read;
```

```
Query.Open;
Query.SetFilter(Column, String);
Query.GetFilter(Column);
Query.Read;
```

Example

The following AL code example demonstrates how to use the **GetFilter** method on a query. The example code sets a filter on a query column, and then displays a message when the query is run that indicates the filter on the column.

This example requires that you create a query called **Customer_SalesQuantity** that has the following characteristics:

- Links table **18 Customer** with table **37 Sales Lines** from the CRONUS International Ltd. demonstration database.
- Includes columns for the **Name** and **No.** fields from the **Customer** table and the **Quantity** field from **Sales Lines** table.
- The ColumnFilter property of the **Quantity** column is set to include values greater than 5.

The following AL code runs the query and displays a message that contains the filter that is set on a query column. You can add the code to a codeunit, and then run the codeunit to see the results.

```
var
    MyQuery: Query "Customer SalesQuantity";
    MyFilter: Text;
    Text000: Label 'The filter is: %1';
begin
    // Sets a filter to display only sales quantities greater than 10. This overwrites the ColumnFilter
property.
    MyQuery.SetFilter(Quantity, '>10');
    // Runs the query and applies the filter.
    MyQuery.Open;
    // Returns the filter on the Quantity column and displays the filter in a message.
    MyFilter := MyQuery.GetFilter(Quantity);
    Message(Text000, MyFilter);
end;
```

Running the code returns the following message:

The filter is: Quantity > 10

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.GetFilters Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the filters that are applied to all columns in the query. The following code shows the syntax of the GetFilterS method. Query is a variable of the Query data type that specifies the query object.

Syntax

```
Filter :=   Query.GetFilters()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Query Type: Query An instance of the Query data type.

Return Value

Filter Type: String All filters of the query

Remarks

The **GetFilterS** method returns the filters that are currently set for all data columns and filter rows by the SetFilter Method (Query) method, SetRange Method (Query)method, and the ColumnFilter Property. The **GetFilter** method does not return filters that are set on a column's source field by the DataItemTableFilter Property or global filters that are set by the **FilterGroup** method.

You can call the **GetFilterS** method multiple times and at any point in the code. If you call the **GetFilterS** method before the first **SetFilter** or **SetRange** method call, then the **GetFilterS** method returns only filters that are set by the **ColumnFilter** property of the columns.

Filters that are set by the **SetFilter** method and **SetRange** method are applied to a query when the query is opened by using a call to the **Open**, the **SaveAsXML**, or **SaveAsCSV** methods. You must consider the location of the **GetFilterS** method with respect to these methods to obtain the results that you want. For example, in the following two code examples, the **GetFilterS** method will return the filter set by the **SetFilter** method call. However, in the first example, the filter has been applied to the query dataset; in the second example, the filter has not been applied.

```
Query.SetFilter(Column, String);
Query.Open;
Query.GetFilterS;
Query.Read;
```

```
Query.Open;
Query.SetFilter(Column, String);
Query.GetFilterS;
Query.Read;
```

Example

The following AL code example demonstrates how to use the **GetFilterS** method on a query. The example code sets filters on a query column, and then displays a message when the query is run that indicates the filter on the column.

This example requires that you create a query called **Customer_SalesQuantity** that has the following characteristics:

- Links table **18 Customer** with table **37 Sales Lines** from the CRONUS International Ltd. demonstration database.
- Includes columns for the **Name** and **No.** fields from the **Customer** table and the **Quantity** field from **Sales Lines** table.
- The **ColumnFilter** property of the **Quantity** column is set with a filter that includes values greater than 10.

The following AL code runs the query and displays a message that contains the filter that is set on a query column. You can add the code to the OnRun trigger of a codeunit, and then run the codeunit to see the results.

```
var
    MyQuery: Query "Customer SalesQuantity";
    MyFilter: Text;
    Text000: Label 'The filters are as follows: %1';
begin
    // Sets a filter to display only sales quantities greater than 10. This overwrites the value of
ColumnFilter property.
    MyQuery.SetFilter(Quantity, '>10');
    // Sets a filter to display the columns with the value Selangorian Ltd. only.
    MyQuery.SetFilter(Name, 'Selangorian Ltd.');
    // Runs the query and applies the filter.
    MyQuery.Open;
    // Returns the filters that are on the Quantity column and displays the filters in a message.
    MyFilters := MyQuery.GetFilterS;
    Message(Text000, MyFilters);
end;
```

Running the code returns the following message:

The filters are as follows: Quantity > 10, Name = Selangorian Ltd.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.Open Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs a query object and generates a data set that can be read. The following code shows the syntax of the Open method. Query is a variable of the Query data type that specifies the query object.

Syntax

```
[Ok := ]  Query.Open()
```

Parameters

Query Type: Query An instance of the Query data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

When the **Open** method is called, the query is executed and the *Query* variable is put in a state in which the resulting dataset can be read by the **Read** method. The **Open** method has the following behavior:

- To apply filters for the **SetFilterS** method or **SetRange** method, you call the **SetFilterS** method or **SetRange** method before the **Open** method.
- To read a row from the dataset, you must call the **Open** method before the **Read** method, as shown in the following code example.

  ```
  Query.SetFilter(Column1, String);
  Query.Open;
  Query.Read;
  ```

- To close an open query and return it to the initialized state, you can call the **Close** method. However, you can call the **Open** method multiple times without calling the **Close** method because the **Open** method implicitly calls the **Close** method if the query dataset is currently in an opened state.
- If the **Open** method is called on a query that is already in the opened or in the reading state, then the query dataset is closed, and then the query is executed again. To continue to loop through the dataset, the **Read** method must be called again. The next **Read** method call returns the first row in the dataset, as shown in the following code example.

```
// Opens the query and generates a dataset.
Query.Open;
Query.Read;
// Closes the query and reopens it.
Query.Open;
// Reads the first row in the new dataset.
Query.Read;
```

- **Open** method does not clear any filters that were set by the **SetFilter** or **SetRange** methods on a previous **Open** call. If you want to clear the filters, then you must call the **Clear** method on the query variable.

```
Query.SetFilter(Column1, String);
Query.Open;
Query.Read;
Clear(query);
Query.Open;
Query.Read;
```

- You are required to call the **Open** method before the SaveAsXML Method or SaveAsCSV Method. The **SaveAsXML** and **SaveAsCSV** methods automatically close the current query dataset and initialize a new instance of the query.

Example

The following example demonstrates how to use the **Open** method on a query. The example code sets filters on the query, opens the query, and then reads the dataset. For each row in the dataset, a message box is displayed that contains the values of the columns in the row.

This example requires that you create a query called **Customer_SalesQuantity** that links table **18 Customer** with table **37 Sales Lines** from the CRONUS International Ltd. demonstration database. Include columns for the **Name** and **No.** fields from the Customer table and the **Quantity** field from Sales Lines table.

The following AL code opens the query, reads each row of the dataset, and then displays a message that uses the content of the row. You can add the code to a codeunit, and then run the codeunit to see the results.

```
var
    MyQuery: Query "Customer SalesQuantity";
    Text000: Label 'Customer name = %1, Quantity = %2';
begin
    // Sets a filter to display only sales quantities greater than 20.
    MyQuery.SetFilter(Quantity, '>20');
    // Runs the query.
    MyQuery.Open;
    // Reads each row in the dataset and displays a message with column values.
    // Stops reading when there are no more rows remaining in the dataset (Read is False).
    while MyQuery.Read do
    begin
      Message(Text000, MyQuery.Name, MyQuery.Quantity);
    end;
    MyQuery.Close;
end;
```

When the code is run, a message that resembles the following appears for each row in the dataset:

Customer name = The Device Shop, Quantity = 30

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.Read Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads data from a row in the resulting data set of a query.

Syntax

```
[Ok := ]  Query.Read()
```

Parameters

Query Type: Query An instance of the Query data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

When the **Read** method is called, the next row in the dataset query is retrieved. While in the reading state, values of columns in the row can be accessed by calling *Query.ColumnName*, where *Query* is the query type variable that specifies the query object and *ColumnName* is the column of the query.

To read a row in the dataset, you must call the **Open** method before the **Read** method. After the **Read** method call, columns can be accessed as shown in the following code example.

```
query.Open;
query.Read;
query.ColumnName
```

You can call the **Read** method multiple times after the **Open** method to read consecutive rows in the dataset. The first **Read** method call retrieves the first row from the resulting dataset and each subsequent **Read** method retrieves the next row from the dataset. For example, the second **Read** method call retrieves the second row, the third **Read** method call retrieves the third row, and so on.

```
Query.Open;
// Reads the first row in the dataset.
Query.Read;
// Accesses a column in the first row of the dataset.
Query.ColumnName
// Reads the second row in the dataset.
Query.Read;
// Accesses a column in the first row of the dataset.
Query.ColumnName
```

> **NOTE**
>
> If the **Read** method is called and there are no more rows in the dataset, then the **Read** method returns **false**.

Example

The following example demonstrates how to use the **Read** method on a query. The example code sets filters on the query, opens the query, and then reads the dataset. For each row in the dataset, a message box is displayed that contains the values of the columns in the row.

This example requires that you create a query called **Customer_SalesQuantity** that is links table **18 Customer** with table **37 Sales Lines** from the CRONUS International Ltd. demonstration database. Include columns for the **Name** and **No.** fields from the Customer table and the **Quantity** field from Sales Lines table.

The following AL code opens the query, reads each row of dataset, and then displays a message that has the content of the row. You can add the code to a codeunit, and then run the codeunit to see the results.

```
var
  MyQuery: Query "Customer SalesQuantity";
  Text000: Label 'Customer name = %1, Quantity = %2';
begin
    // Sets a filter to display only sales quantities greater than 20.
    MyQuery.SetFilter(Quantity, '>20');
    // Runs the query.
    MyQuery.Open;
    // Reads each row in the dataset and displays message with column values.
    // Stops reading when there are no more rows remaining in the dataset (Read is False).
    while MyQuery.Read do
    begin
      Message(Text000, MyQuery.Name, MyQuery.Quantity);
    end;
    // Closes the query.
    MyQuery.Close;
end;
```

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsCsv Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as comma separated values (CSV)

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Query.SaveAsCsv(FileName: String [, Format: Integer] [, FormatArgument: String])
```

Parameters

Query Type: Query An instance of the Query data type.

FileName
Type: String
The path and name of the file that you want to save the query to.

Format
Type: Integer
Specifies whether the columns of the resulting data set are at fixed positions in the CSV file or separated only by a delimiter.

FormatArgument
Type: String
You set the FormatArgument parameter based on the setting of the Format parameter. If the Format parameter is set to 0, then the FormatArgument parameter specifies the starting position of each column in the data set. The value is a comma separated string of integers that includes an integer for every column. In a CSV file, each line is evenly divided into positions for holding characters. The first integer corresponds to the starting position of the first column, the second integer corresponds to the starting position of the second column, and so on.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsCsv Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as comma separated values (CSV)

Syntax

```
[Ok := ]  Query.SaveAsCsv(OutStream: OutStream [, Format: Integer] [, FormatArgument: String])
```

Parameters

Query Type: Query An instance of the Query data type.

OutStream
Type: OutStream
The stream that you want to save the query as CSV to.

Format
Type: Integer
Specifies whether the columns of the resulting data set are at fixed positions in the CSV file or separated only by a delimiter.

FormatArgument
Type: String
You set the FormatArgument parameter based on the setting of the Format parameter. If the Format parameter is set to 0, then the FormatArgument parameter specifies the starting position of each column in the data set. The value is a comma separated string of integers that includes an integer for every column. In a CSV file, each line is evenly divided into positions for holding characters. The first integer corresponds to the starting position of the first column, the second integer corresponds to the starting position of the second column, and so on.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as XML

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Query.SaveAsXml(FileName: String)
```

Parameters

Query Type: Query An instance of the Query data type.

FileName
Type: String
The path and name of the file that you want to save the query to.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as XML

Syntax

```
[Ok := ]  Query.SaveAsXml(OutStream: OutStream)
```

Parameters

Query Type: Query An instance of the Query data type.

OutStream
Type: OutStream
The stream that you want to save the query as XML to.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SecurityFiltering Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets how security filters are applied to the query.

Syntax

```
[SecurityFiltering := ]  Query.SecurityFiltering([NewSecurityFiltering: SecurityFilter])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Query Type: Query An instance of the Query data type.

NewSecurityFiltering
Type: SecurityFilter
The new security filter for the query

Return Value

SecurityFiltering Type: SecurityFilter The security filter applied to the query.

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SetFilter Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a filter on a column of a query to limit the records in the resulting data set of a query.

Syntax

```
Query.SetFilter(Column: Any, String: String [, Value: Any,...])
```

Parameters

Query Type: Query An instance of the Query data type.

Column
Type: Any
The name of the column in the query that you want to filter. The name is defined by the column's Name Property.

String
Type: String
The filter expression. A valid expression consists of alphanumeric characters and one or more of the following operators: <, >, \, &, |, and =. You can use replacement fields (%1, %2, and so on) to insert values at run-time.

Value
Type: Any
Replacement values to insert in replacement fields in the filter expression. The data type of Value must match the data type of field that is referred to by the ColumnName.

Remarks

To apply filters to a dataset, the **SetFilter** method must be called before the **Open**, **SaveAsXML**, and **SaveAsCSV** methods, as shown in the following example. To remove filters from query, you call the Clear Method.

```
Query.SetFilter(Column1, String);
Query.Open;
Query.Read;
Clear(Query);
```

A call to the **SetFilter** method automatically closes a query dataset that is currently open. Therefore, the following code is unauthorized and will fail because there is no open dataset for the **Read** method to read.

```
Query.Open;
Query.Read;
Query.SetFilter(Column2, String);
Query.Read;
```

You can have multiple calls to the **SetFilter** method. If **SetFilter** method calls set filters on different columns, then the filters are combined and applied to the dataset. If consecutive **SetFilter** method calls set filters on the same column, then the last **SetFilter** method call is applied to the column.

In addition to the **SetFilter** method, you can apply filters to a query using the SetRange Method (Query) method, the **FilterGroup** method, and the DataItemTableFilter Property and ColumnFilter Property.

IF THE SETFILTER METHOD...	THEN...
Sets a filter on the same field as the **DataItemTableFilter** property	The two filters are joined into a resulting filter.
Sets a filter on the same field as the **ColumnFilter** property	The **SetFilter** method overwrites the **ColumnFilter** property, so the filter that is set by the **SetFilter** method that is applied to the dataset.
Sets a filter on the same field as the **SetRange** method	The method that is called last is applied to the dataset.
Sets a filter on a field that has global filters that are applied by the **FilterGroup(1)** method	The filters of the **SetFilter** method are added to the global filters.

For example, a query has the following filters set on the **Quantity** column:

- **DataItemTableFilter** property: Quantity=Filter(<100)
- **ColumnFilter** property: Quantity=Filter(<>50)

`Query.SetFilter ("Quantity", '>1')` will result in a filter that is equivalent to: 1<Quantity <100.

Example

The following AL code example demonstrates how to use the **SetFilter** method on a query. The example code sets a filter on a query column, and then displays a message when the query is run that indicates the filter on the column.

This example requires that you create a query called **Customer_SalesQuantity** that has the following characteristics:

- Links table 18, Customer with table 37, Sales Lines from the CRONUS International Ltd. demonstration database.
- Includes columns for the **Name** and **No.** fields from the **Customer** table and the **Quantity** field from **Sales Lines** table.

```
var
  MyQuery: Query "Customer SalesQuantity";
  Text000: Label 'Customer name = %1, Quantity = %2';
begin
    // Sets a filter to display only sales quantities greater than 10.
    MyQuery.SetFilter(Quantity, '>10');
    // Sets a filter to display the columns with the value Selangorian Ltd. only.
    MyQuery.SetFilter(NAME, 'Selangorian Ltd.');
    // Runs the query.
    MyQuery.Open;
    // Reads each row in the dataset and displays message with column values.
    // Stops reading when there are no more rows remaining in the dataset (Read is False).
    while MyQuery.Read do
    begin
      Message(Text000, MyQuery.Name, MyQuery.Quantity);
    end;
    // Saves the resulting dataset as a CSV file.
    MyQuery.SaveAsCSV('c:\temp\CustomerSales.csv');
    // Closes the query.
    Myquery.Close;
end;
```

When the code is run, a message that resembles the following appears for each row in the dataset:

Customer name = Selangorian Ltd., Quantity = 30

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.SetRange Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a filter on a range of values on a column of a query data set.

Syntax

```
Query.SetRange(Column: Any [, FromValue: Any] [, ToValue: Any])
```

Parameters

Query Type: Query An instance of the Query data type.

Column
Type: Any
The name of the column in the query that you want to filter. The name is defined by the column's Name Property.

FromValue
Type: Any
The lower limit of the range. The data type of this parameter must match the data type of Column. If you set only the FromValue parameter, then the ToValue parameter is set to the same value as FromValue.

ToValue
Type: Any
The upper limit of the range. The data type of this parameter must match the data type of Column. If you omit the ToValue parameter, then the only the value that is specified for FromValue is used, which enables you to filter on a single column value instead of a range. If you omit both the FromValue and ToValue parameters, then the method removes all filters that are already set on the column.

Remarks

SetRange is a quick way to set a simple filter on a field. The SetRange method is functionally equivalent to calling *Query*.SetFilter(ColumnName, 'FromValue..ToValue').

To apply filters to a dataset, the **SetRange** method must be called before the **Open**, **SaveAsXML**, and **SaveAsCSV** methods, as shown in the following example. To remove filters, you call the Clear Method or **SetRange** without values for the *FromValue* and *ToValue* parameters.

```
Query.SetRange(Column1, FromValue, ToValue);
Query.Open;
Query.Read;
Clear(Query);
```

A call to the **SetRange** method automatically closes a query dataset that is currently open. Therefore, the following code is unauthorized and will fail because there is no open dataset for the **Read** method to read.

```
Query.Open;
Query.Read;
Query.SetRange(Column1, FromValue, ToValue);
Query.Read;
```

In addition to the **SetRange** method, you can apply filters to a query using the **SetFilter** method and the DataItemTableFilter Property and ColumnFilter Property.

IF THE SETRANGE METHOD...	THEN...
Sets a filter on the same field as the **DataItemTableFilter** property	The two filters are joined into a resulting filter.
Sets a filter on the same field as the **ColumnFilter** property	The **SetRange** method overwrites the **ColumnFilter** property, so the filter that is set by the **SetRange** method that is applied to the dataset.
Sets a filter on the same field as the **SetFilter** method	The method that is called last is applied to the dataset.
Sets a filter on a field that has global filters that are applied by the **FilterGroup(1)** method	The filters of the **SetRange** method are added to the global filters.

For example, a query has the following filters set on the **Quantity** column:

- **DataItemTableFilter** property: Quantity=Filter(<>10)
- **ColumnFilter** property: Quantity=Filter(<>5)

`Query.setRange(Quantity, 1, 15)` will result in a filter that is equivalent to: 1 < Quantity <15, except for 10.

Example

The following AL code example demonstrates how to use the **SetRange** method on a query. The example code sets a filter on a query column and saves the resulting dataset as a CSV file. A message also displays when the query is run that indicates the filter on the column.

This example requires that you do the following:

1. Create a query called **Customer_SalesQuantity** that has the following characteristics:
   - Links table **18 Customer** with table **37 Sales Lines** from the CRONUS International Ltd. demonstration database.
   - Includes columns for the **Name** and **No.** fields from the Customer table and the **Quantity** field from Sales Lines table.
2. Create the following AL variables and text constant in the object that will run the query, such as a codeunit.

   ```
   var
      MyQuery: Query "Customer SalesQuantity";
      Text000: Label 'Customer name = %1, Quantity = %2';
   ```

The following AL code uses the **SetRange** method to filter a query dataset over a range of values on the **Quantity** column. You can add the code to a codeunit, and then run the codeunit to see the results.

```
// Sets a filter to display only sales quantities greater than 10.
MyQuery.SetRange(Quantity, '10', '50');
// Sets a filter to display the columns with the value Selangorian Ltd. only.
MyQuery.SetFilter(Name, 'Selangorian Ltd.');
// Runs the query.
MyQuery.Open;
// Reads each row in the dataset and displays message with column values.
// Stops reading when there are no more rows remaining in the dataset (Read is False).
while MyQuery.Read do
begin
  Message(Text000, MyQuery.Name, MyQuery.Quantity);
end;
// Saves the resulting dataset as a CSV file.
MyQuery.SaveAsCSV('c:\temp\CustomerSales.csv');
MyQuery.Close;
```

When the code is run, a message that resembles the following appears for each row in the dataset:

Customer name = Selangorian Ltd., Quantity = 30

See Also

Query Data Type
Getting Started with AL
Developing Extensions

Query.TopNumberOfRows Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the maximum number of rows to include in the resulting data set of a query.

Syntax

```
[CurrentRows := ]  Query.TopNumberOfRows([NewRows: Integer])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Query Type: Query An instance of the Query data type.

NewRows
Type: Integer
The number of rows to include in the resulting data set. If you do not set the NewRows parameter, then the resulting data set will include all rows. If you set the value to 0, then there is no limit and all rows of the data set are returned.

Return Value

CurrentRows Type: Integer Gets the current maximum number of rows included in the resulting data set

Remarks

You use the **TopNumberOfRows** method to limit the resulting dataset to the first set of rows that are generated for the query. For example, you can include only the first 10 or first 100 rows in the resulting dataset. The **TopNumberOfRows** method is useful for key performance indicators such as the top number of customers or sales.

You can also specify the number of rows to include in the dataset by setting the TopNumberOfRows Property. The **TopNumberOfRows** method will overwrite the **TopNumberOfRows** property setting.

Example

This code example demonstrates how to use the **TopNumberOfRows** method on a query to return the top 10 customer sales orders based on the quantity of items.

The following query object links table **18 Customer** and table **37 Sales Line** and uses the **TopNumberOfRows** property to get top 5 customer sales orders based on the quantity of items.

```
query 50123 "Customer_Sales_Quantity"
{
    QueryType = Normal;
    // Sets the resultS to include the top 5 the results in descending order
    TopNumberOfRows = 5;
    OrderBy = descending(Qty);

    elements
    {
        dataitem(C; Customer)
        {
            column(Customer_Number; "No.")
            {
            }

            column(Customer_Name; Name)
            {
            }

            dataitem(SL; "Sales Line")
            {
                DataItemLink = "Sell-to Customer No." = c."No.";
                SqlJoinType = InnerJoin;

                column(Qty; Quantity)
                {
                }
            }
        }
    }
}
```

The following codeunit runs the query, saves it as a CSV file, and displays a message that states the number of rows that are returned in the resulting dataset.

```
codeunit 50100 MyQueryTop10
{
    trigger OnRun()
    begin

        // Overwrites the TopNumberOfRows property and returns the first 10 rows in the dataset.
        //MyQuery.TopNumberOfRows(10);
        // Opens the query.
        MyQuery.Open;
        // Reads each row of the dataset and counts the number of rows.
        while MyQuery.Read do begin
            Counter += 1;
        end;
        // Saves the dataset as a CSV file.
        MyQuery.SaveAsCsv('c:\temp\CustomerSales.csv');
        // Displays a message that shows the number of rows.
        Message(Text000, counter);

    end;

    var
        MyQuery: Query "Customer_Sales_Quantity";
        Counter: Integer;
        Text000: Label 'count %1.';
}
```

See Also

RecordId Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Contains the table number and the primary key of a table.

The following methods are available on instances of the RecordId data type.

METHOD NAME	DESCRIPTION
GetRecord()	Gets a RecordRef that refers to the record identified by the RecordID.
TableNo()	Gets the table number of the table that is identified by RecordID. This function returns an error if the record is blank.

See Also

Getting Started with AL
Developing Extensions

RecordId.GetRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a RecordRef that refers to the record identified by the RecordID.

Syntax

```
RecordRef :=   RecordId.GetRecord()
```

Parameters

RecordId Type: RecordId An instance of the RecordId data type.

Return Value

RecordRef Type: RecordRef The RecordRef of the record.

Remarks

No data is read from the database when you run this method and therefore, no other fields in the record are set. Furthermore, no filters are set on the record.

Example

The following example opens table number 18 (Customer table) and sets a reference to the table. The FindLast Method (RecordRef) selects the last record in the table. The RecordId Method (RecordRef) retrieves the ID of the currently selected record. In this case, it is the last record in the table. The GetRecord Method (RecordId) uses the retrieved record ID to determine the RecordRef of the selected record (the last record). This example requires that you create the following global variables and text constant.

```
var
    RecRef: RecordRef;
    RecID: RecordID;
```

```
RecRef.Open(18);
RecRef.FindLast;
RecID := RecRef.RecordId;
RecRef := RecID.GetRecord;
```

See Also

RecordId Data Type
Getting Started with AL
Developing Extensions

RecordId.TableNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the table number of the table that is identified by RecordID. This function returns an error if the record is blank.

Syntax

```
No :=   RecordId.TableNo()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordId Type: RecordId An instance of the RecordId data type.

Return Value

No Type: Integer The ID of the table.

Remarks

In previous versions of Dynamics 365, if a RecordID referred to a temporary table, then the table number value of the RecordID was the run-time generated sequence ID, which is from the base value of 2000100000. You could use the table number to determine if a RecordID referred to a temporary table. In Dynamics 365, the table number value of a RecordID always contains the ID of the originating physical table and not the run-time generated sequence ID. If you previously used the TableNo Method (RecordID) to test for the sequence number and determine if the RecordID was temporary, then you use the IsTemporary Method (RecordRef) in Dynamics 365 Business Central instead.

Example

The following example opens the Customer table with the record reference variable named RecRef and finds the first record in the Customers table. If a record is found, the RecordId Method (RecordRef) gets the ID of the first record in the table. The TableNo method then uses the RecID variable to retrieve the number of the table that contains the record. The table number is then displayed in a message box.

```
var
    RecRef: RecordRef;
    RecID: RecordID;
    varTableNumber: Integer;
begin
    RecRef.Open(Database::Customer);
    if RecRef.Find('-') then begin
      RecID := RecRef.RecordId;
      varTableNumber := RecID.TableNo;
      Message('The Customer table is number: %1', varTableNumber);
    end else begin
    Message('No records found in the table');
    end;
end;
```

See Also

RecordId Data Type
Getting Started with AL
Developing Extensions

RecordRef Data Type

2/17/2021 • 8 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

References a record in a table.

The following methods are available on instances of the RecordRef data type.

METHOD NAME	DESCRIPTION
AddLink(String [, String])	Adds a link to a record in a table.
AddLoadFields([Integer,...])	Specifies additional fields to be initially loaded when the record is retrieved from its data source. Subsequent calls to AddLoadFields will not overwrite fields already selected for the initial load.
AreFieldsLoaded(Integer,...)	Checks whether the specified fields are all initially loaded.
Ascending([Boolean])	Changes or checks the order in which a search through the table that is referred to by RecordRef will be performed.
Caption()	Gets the caption of the table that is currently selected. Returns an error if no table is selected.
ChangeCompany([String])	Redirects references to table data from one company to another.
ClearMarks()	Removes all the marks from a record.
Close()	Closes the current page or table.
Copy(var Record [, Boolean])	Copies a specified record's filters, views, automatically calculated FlowFields, marks, fields, and keys that are associated with the record from a table or creates a reference to a record.
Copy(RecordRef [, Boolean])	Copies a specified record referece's filters, views, automatically calculated FlowFields, marks, fields, and keys that are associated with the record from a table or creates a reference to a record.
CopyLinks(Record)	Copies all the links from a particular record.
CopyLinks(RecordRef)	Copies all the links from a particular record.
CopyLinks(Variant)	Copies all the links from a particular record.
Count()	Counts the number of records that are in the filters that are currently applied to the table referred to by the RecordRef.
CountApprox()	Gets an approximate count of the number of records in the table
CurrentCompany()	Gets the current company of a database table referred to by a RecordRef.
CurrentKey()	Gets the current key of the table referred to by the RecordRef. The current key is returned as a string.

METHOD NAME	DESCRIPTION
CurrentKeyIndex([Integer])	Gets or sets the current key of the table referred to by the RecordRef. The current key is set or returned as a number. This first key = 1, and so on. If RecordRef does not have an active record, CurrentKeyIndex will return -1. If this value is then passed to KeyIndex, an index out of bounds error will occur. Therefore it is important to implement a check of the RecordRef parameter.
Delete([Boolean])	Deletes a record in a table.
DeleteAll([Boolean])	Deletes all records in a table that fall within a specified range.
DeleteLink(Integer)	Deletes a specified link from a record in a table.
DeleteLinks()	Deletes all of the links that have been added to a record.
Duplicate()	Duplicates the table that contains the RecordRef.
Field(Integer)	Gets a FieldRef for the field that has the number FieldNo in the table that is currently selected. If no field has this number, the method returns an error.
FieldCount()	Gets the number of fields in the table that are currently selected or returns the number of fields that have been defined in a key. Returns an error if no table or no key is selected.
FieldExist(Integer)	Determines if the field that has the number FieldNo exists in the table that is referred to by the RecordRef. Returns an error if no table is currently selected.
FieldIndex(Integer)	Gets the FieldRef of the field that has the specified index in the table that is referred to by the RecordRef.
FilterGroup([Integer])	Changes the filter group that is being applied to the table. You can also use this method to return the number of the current filtergroup. You cannot return the number of the filtergroup and set a new filtergroup at the same time.
Find([String])	Finds a record in a table based on the values stored in the key fields.
FindFirst()	Finds the first record in a table based on the current key and filter.
FindLast()	Finds the last record in a table based on the current key and filter.
FindSet([Boolean] [, Boolean])	Finds a set of records in a table based on the current key and filter. FindSet can only retrieve records in ascending order.
Get(RecordId)	Gets a record based on the ID of the record.
GetBySystemId(Guid)	Gets a record based on the ID of the record. The RecordRef must already be opened.
GetFilters()	Determines which filters have been applied to the table referred to by the RecordRef.
GetPosition([Boolean])	Gets a string that contains the primary key of the current record.
GetTable(Record)	Gets the table of a Record variable and causes the RecordRef to refer to the same table.

METHOD NAME	DESCRIPTION
GetView([Boolean])	Returns a string that describes the current sort order, key, and filters on a table.
HasFilter()	Determines whether a filter has been applied to the table that the RecordRef refers to.
HasLinks()	Determines whether a record contains any links.
Init()	Initializes a record in a table.
Insert()	Inserts a record into a table without executing the code in the OnInsert trigger.
Insert(Boolean)	Inserts a record into a table.
Insert(Boolean, Boolean)	Inserts a record into a table.
IsDirty()	Gets a boolean value that indicates whether the current in-memory instance of a record or filtered set of records has changed since being retrieved from the database.
IsEmpty()	Determines whether any records exist in a filtered set of records in a table.
IsTemporary()	Determines whether a RecordRef refers to a temporary table.
KeyCount()	Gets the number of keys that exist in the table that is referred to by the RecordRef. Returns an error if no table is selected.
KeyIndex(Integer)	Gets the KeyRef of the key that has the index specified in the table that is currently selected. The key can be composed of fields of any supported data type. Data types that are not supported include BLOBs, FlowFilters, variables, and functions. If the sorting key is set to a field that is not part of a key, then the KeyIndex is -1.
LoadFields(Integer,...)	Accesses the table's corresponding data source and loads the values of the specified fields on the record.
LockTable([Boolean] [, Boolean])	Locks a table to protect it from write transactions that conflict with each other.
Mark([Boolean])	Marks a record. You can also use this method to determine whether a record is marked.
MarkedOnly([Boolean])	Activates a special filter. After you use this function, your view of the table includes only records marked by this function.
Modify([Boolean])	Modifies a record in a table.
Name()	Identifies the name of the table
Next([Integer])	Steps through a specified number of records and retrieves a record.
Number()	Gets the table ID (number) of the table that contains the record that was referred to by the RecordRef.
Open(Integer [, Boolean] [, String])	Causes a RecordRef variable to refer to a table, which is identified by its number in a particular company.
ReadConsistency()	Gets a value indicating whether read consistency is enabled.

METHOD NAME	DESCRIPTION
ReadPermission()	Determines if you can read from a table.
RecordId()	Gets the RecordID of the record that is currently selected in the table. If no table is selected, an error is generated.
RecordLevelLocking()	Gets a value indicating whether record level locking is enabled.
Rename(Any [, Any,...])	Changes the value of a primary key in a table.
Reset()	Removes all filters, including any special filters set by the MarkedOnly method (Record), changes fields select for loading back to all, and changes the current key to the primary key. Also removes any marks on the record and clears any AL variables defined on its table definition.
SecurityFiltering([SecurityFilter])	Gets or sets how security filters are applied to the RecordRef.
SetLoadFields([Integer,...])	Sets the fields to be initially loaded when the record is retrieved from its data source. This will overwrite fields previously selected for initial load.
SetPermissionFilter()	Applies the user's security filter to the referenced record. The security filter is combined with any other filters that are placed on the record with SetFilter or SetRange. The combined filter will not include any records outside the range of the security filter and this will prevent a runtime permission error from occuring when the record is read. If the permission filter is not set, an error can occur if you attempt to read a record that is outside the range of the user's security filter.
SetPosition(String)	Sets the fields in a primary key on a record to the values specified in the String parameter. The remaining fields are not changed.
SetRecFilter()	Sets a filter on a record that is referred to by a RecordRef.
SetTable(Record)	Sets the table to which a Record variable refers as the same table as a RecordRef variable.
SetView(String)	Sets the current sort order, key, and filters on a table.
SystemCreatedAtNo()	Gets the field number that is used by the SystemCreatedAt field. The SystemCreatedAt field is a system field that the platform adds to all table objects.
SystemCreatedByNo()	Gets the field number that is used by the SystemCreatedBy field. The SystemCreatedBy field is a system field that the platform adds to all table objects.
SystemIdNo()	Gets the field number that is used by the SystemId field. The SystemId field is a system field that the platform adds to all table objects.
SystemModifiedAtNo()	Gets the field number that is used by the SystemModifiedAt field. The SystemModifiedAt field is a system field that the platform adds to all table objects.
SystemModifiedByNo()	Gets the field number that is used by the SystemModifiedBy field. The SystemModifiedBy field is a system field that the platform adds to all table objects.
WritePermission()	Determines if you can write to a table.

Remarks

The RecordRef object can refer to any table in the database. Use the Open method to use the table number to select the table that you want to access, or use the GetTable method to use another record variable to select the table that you want to access.

If one RecordRef variable is assigned to another RecordRef variable, then they both refer to the same table instance.

See Also

Getting Started with AL
Developing Extensions

RecordRef.AddLink Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a link to a record in a table.

Syntax

```
[ID := ]  RecordRef.AddLink(URL: String [, Description: String])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

URL
Type: String
The link that you want to add to the record.
Description
Type: String
Optional description of the link.

Return Value

ID Type: Integer The ID of the URL that you want to add to the record. Every time that you add a link to a page or a table, an entry is created in the Record Link system table. Each entry is given an ID.

Remarks

The URL can be a link to a Web site, a file stored on the local or on a remote computer, or a link to a Dynamics 365 page. You can then view the link in the **Links** FactBox on pages that display the record.

Example

The following example adds a link to a record in the Customer table. The code starts by opening table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for the first field (No.). Next, `MyFieldRef.Value` is set to record 01121212. The Find Method (RecordRef) method searches the records for record no. 01121212. If the record is found, then the AddLink method adds a link to the record. The link is assigned a link ID, which is stored in the LinkID variable. The link ID is displayed in a message box. You can view the link you added in the **Links** FactBox on the Customer List or Customer Card pages.

```
var
    CustomerNum: Code;
    varLink: Text;
    CustomerRecref: RecordRef;
    MyFieldRef: FieldRef;
    LinkID: Integer;
    Text000: Label 'The link with ID %1 has been added.';
    Text001: Label 'The customer cannot be found.';
begin
    CustomerNum := '01121212';
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value := CustomerNum;
    if CustomerRecref.Find('=') then begin
      LinkID := CustomerRecref.AddLink(varLink);
      Message(Text000, LinkID);
    end else
      Message(Text001);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.AddLoadFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Specifies additional fields to be initially loaded when the record is retrieved from its data source. Subsequent calls to AddLoadFields will not overwrite fields already selected for the initial load.

Syntax

```
[Ok := ]  RecordRef.AddLoadFields([Fields: Integer,...])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Fields
Type: Integer
The FieldNo's of the fields to be loaded.

Return Value

Ok Type: Boolean **true** if all fields are selected for subsequent loads; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Calling SetLoadFields on a record without passing any fields will reset the fields selected to load to the default, where all readable normal fields are selected for load.

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

Example

This code example uses the AddLoadFields method to make sure that if a record is a **Currency**, then the **Currency Factor** field is loaded. This code would have to be called before a database operation is executed on the RecordRef.

```
procedure AlwaysNeededFields(VAR MyRecordRef: RecordRef)
var
        Currency: Record Currency;
begin
if (MyecordRef.Number = Database::Currency) then
// We always want the Currency."Currency Factor"
MyRecordRef.AddLoadFields(Currency.FieldNo(Currency."Currency Factor"));
end;
```

See Also

RecordRef.AreFieldsLoaded Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Checks whether the specified fields are all initially loaded.

Syntax

```
Ok :=   RecordRef.AreFieldsLoaded(Fields: Integer,...)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Fields

Type: Integer

The FieldNo's of the fields to check.

Return Value

Ok Type: Boolean **true** if all the fields specified by the Fields parameter are currently loaded; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

Example

This code example shows how you could use the AreFieldsLoaded method to determines how many fields are currently loaded for a given recordRef. Note that, because the platform might require more fields be loaded than specified by calls to SetLoadFields and AddLoadFields, the result might be larger than expected.

```
procedure GetLoadedFieldCount(MyRecordRef: RecordRef): Integer
var
    MyFieldRef: FieldRef;
    LoadedFields: Integer;
    Idx: Integer;
begin
    for Idx := 0 to MyRecordRef.FieldCount do begin
        MyFieldRef := MyRecordRef.FieldIndex(idx);
        if (MyFieldRef.Active() AND MyRecordRef.AreFieldsLoaded(MyFieldRef.Number)) then
            LoadedFields += 1;
    end;

    exit(LoadedFields);
end;
```

See Also

Using Partial Records
RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Ascending Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Changes or checks the order in which a search through the table that is referred to by RecordRef will be performed.

Syntax

```
[IsAscending := ]  RecordRef.Ascending([SetAscending: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

SetAscending

Type: Boolean

If this parameter is true, it will search in ascending order. If this parameter is false, it will search in descending order. If you do not specify this parameter, it will check the search order.

Return Value

IsAscending Type: Boolean Specifies the order in which a search will be performed.

Remarks

This method works just like the Ascending Method (Record).

Example 1

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The SetView Method (RecordRef) sets a filter that includes sorting the data in ascending order. The Ascending method then checks whether the sort order is ascending, stores the return value in the IsAscending variable and displays **True** in a message box.

```
var
    IsAscending: Boolean;
    CustomerRecref: RecordRef;
    Text000: Label 'Is the sort order ascending?  %1';
begin
    CustomerRecref.Open(18);
    CustomerRecref.SetView('Sorting(Name) Order(Ascending) Where(No.=Const(10000..20000))');
    IsAscending := CustomerRecref.Ascending;
    Message(Text000, IsAscending);
end;
```

Example 2

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The SetView Method (RecordRef) sets a filter that includes sorting the data in descending order. The Ascending method then checks whether the sort order is ascending, stores the return value in the IsAscending variable and displays **False** in a message box because the sort order is descending. The Ascending method changes the sort order to ascending by setting the *SetAscending* parameter to **true**. The Ascending method checks the sort order again. This time **True** is displayed.

```
CustomerRecref.Open(18);
CustomerRecref.SetView('Sorting(Name) Order(Descending) Where(No.=Const(10000..20000))');
IsAscending := CustomerRecref.Ascending;
Message(Text000, IsAscending);
IsAscending := CustomerRecref.Ascending(True);
Message(Text000, IsAscending);
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the caption of the table that is currently selected. Returns an error if no table is selected.

Syntax

```
Caption :=   RecordRef.Caption()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Caption Type: String The caption of the table.

Remarks

This method works just like the TableCaption Method (Record).

Example

The following example selects tables 3 through 5 and opens each table as a RecordRef variable that is named MyRecordRef. The Caption method uses the RecorRef variable to retrieve the caption for each of the tables and displays the table number and the caption in a message box. The Close Method (RecordRef) closes the table.

```
var
    varCaption: Text;
    i: Integer;
    MyRecordRef: RecordRef;
    Text000: Label 'Table No: %1 Caption: %2';
begin
    for i := 3 to 6 do begin
        MyRecordRef.Open(i);
        varCaption := MyRecordRef.Caption;
        Message(Text000, i, varCaption);
        MyRecordRef.Close;
        end;
end;
```

This example displays the following:

Table No: 3 Caption: Payment terms

Table No: 4 Caption: Currency

Table No: 5 Caption: Finance Charge Terms

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.ChangeCompany Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Redirects references to table data from one company to another.

Syntax

```
[Ok := ]  RecordRef.ChangeCompany([CompanyName: String])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

CompanyName
Type: String
The name of the company to which you want to change. If you omit this parameter, you change back to the current company.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

When executing this method, the user's access rights are respected. For example, a user cannot access data in *CompanyName* unless he already has the necessary access rights.

The **ChangeCompany** method is not affected by the Reset Method (RecordRef). You can deselect a company by making a new call to **ChangeCompany** or by using the Clear Method.

Global filters always belong to a specific company. If you use the following code to select the company named NewCompany, any filters assigned to *RecordRef* will be transferred to *RecordRef* in the new company.

```
RecordRef.ChangeCompany(NewCompany);
```

Even if you run the **ChangeCompany** method, triggers still run in the current company, not in the company that you specified in the **ChangeCompany** method.

Example

This example shows how to use the **ChangeCompany** method. The following code takes a RecordRef to table **18 Customer** in the current company and redirects it to the table in another company (in this case Company B). The last record in the Customer table of Company B is then deleted.

```
var
    RecID: RecordID;
    MyRecordRef: RecordRef;
    Text000: Label 'Record to be deleted: %1';
begin
    MyRecordRef.Open(18);
    MyRecordRef.ChangeCompany('Company B');
    MyRecordRef.FindLast;
    RecID := MyRecordRef.RecordId;
    Message(Text000, RecID);
    MyRecordRef.Delete;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.ClearMarks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.3.*

Removes all the marks from a record.

Syntax

```
RecordRef.ClearMarks()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes the current page or table.

Syntax

```
RecordRef.Close()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Remarks

You must use this method if you have several recordrefs defined as variables because these will be maintained until the variable gets out of scope.

Example

The following example opens tables 3 through 10 as a Recordref variable that is named MyRecordRef. For each table that is open, the Caption Method (RecordRef) retrieves the caption of the table and displays the table number and the caption in a messages box. After each caption is displayed, the Close method closes the table before the next table is open.

```
var
    varCaption: Text;
    MyRecordRef: RecordRef;
    i: Integer;
    Text000: Label 'Table No: %1 Caption: %2';
begin
    for i := 3 to 10 do begin
        MyRecordRef.Open(i);
        varCaption := MyRecordRef.Caption;
        Message(Text000, i, varCaption);
        MyRecordRef.Close;
    end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Copy Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.3.*

Copies a specified record's filters, views, automatically calculated FlowFields, marks, fields, and keys that are associated with the record from a table or creates a reference to a record.

Syntax

```
RecordRef.Copy(var FromRecord: Record [, ShareTable: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FromRecord
Type: Record
The record to copy.

ShareTable
Type: Boolean
Specifies whether the method copies filters, views, automatically calculated FlowFields, marks, fields, and keys of the record or creates a reference to a temporary record. If FromRecord and Record are both temporary and ShareTable is true, then the COPY method causes Record to reference the same table as FromRecord. If ShareTable is true, then both Record and FromRecord must be temporary; otherwise an error will occur. The default value is false. If you specify false, only filters, marks, and keys are copied.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Copy Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.3.*

Copies a specified record referece's filters, views, automatically calculated FlowFields, marks, fields, and keys that are associated with the record from a table or creates a reference to a record.

Syntax

```
RecordRef.Copy(FromRecordRef: RecordRef [, ShareTable: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FromRecordRef

Type: RecordRef

The record reference to copy.

ShareTable

Type: Boolean

Specifies whether the method copies filters, views, automatically calculated FlowFields, marks, fields, and keys of the record or creates a reference to a temporary record. If FromRecord and Record are both temporary and ShareTable is true, then the COPY method causes Record to reference the same table as FromRecord. If ShareTable is true, then both Record and FromRecord must be temporary; otherwise an error will occur. The default value is false. If you specify false, only filters, marks, and keys are copied.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CopyLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies all the links from a particular record.

Syntax

```
RecordRef.CopyLinks(FromRecord: Record)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FromRecord
Type: Record
Specifies the record from which you want to copy links.

Remarks

Use this method to copy all the links from a specified record and paste the links to the current record.

The link can be a link to a Web site, a file stored on the local or on a remote computer, or a link to a page in your application.

Example

The following example copies all links from a source record that is named VendorRecord to the currently open record in the Customer table. The source record is record 10000 from the Vendor table. The code opens the Customer table as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for field 1 (No.) in the Customer table. The SetRange Method (FieldRef) selects records in the range 20000 to 40000 from the Customer table and record 10000 from the Vendor table. The Find Method (RecordRef) searches the Customer table for the records in the filtered range. If the record that meets the filter criteria is found, the links from the Vendor record No. 10000 are copied to the customer records in the range 30000 to 40000. The record id of the record to which the links were copied is displayed in a message box. The process is repeated until there is no more record in the range. The Close Method (RecordRef) closes the table.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    VendorRecord: Record Vendor;
    Count: Integer;
    Text000: Label 'The links have been copied to %1';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetRange('30000' , '40000');
    VendorRecord.SetRange("No.", '10000');
    Count := 0;
    if CustomerRecref.Find('-') then
      repeat
        Count := Count + 1;
        CustomerRecref.CopyLinks(VendorRecord);
        Message(Text000, CustomerRecref.RecordId);
      until CustomerRecref.Next = 0;
    CustomerRecref.Close;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CopyLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies all the links from a particular record.

Syntax

```
RecordRef.CopyLinks(FromRecord: RecordRef)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FromRecord
Type: RecordRef
Specifies the record from which you want to copy links.

Remarks

Use this method to copy all the links from a specified record and paste the links to the current record.

The link can be a link to a Web site, a file stored on the local or on a remote computer, or a link to a page in your application.

Example

The following example copies all links from a source record that is named VendorRecord to the currently open record in the Customer table. The source record is record 10000 from the Vendor table. The code opens the Customer table as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for field 1 (No.) in the Customer table. The SetRange Method (FieldRef) selects records in the range 20000 to 40000 from the Customer table and record 10000 from the Vendor table. The Find Method (RecordRef) searches the Customer table for the records in the filtered range. If the record that meets the filter criteria is found, the links from the Vendor record No. 10000 are copied to the customer records in the range 30000 to 40000. The record id of the record to which the links were copied is displayed in a message box. The process is repeated until there is no more record in the range. The Close Method (RecordRef) closes the table.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    VendorRecord: Record Vendor;
    Count: Integer;
    Text000: Label 'The links have been copied to %1';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetRange('30000' , '40000');
    VendorRecord.SetRange("No.", '10000');
    Count := 0;
    if CustomerRecref.Find('-') then
      repeat
        Count := Count + 1;
        CustomerRecref.CopyLinks(VendorRecord);
        Message(Text000, CustomerRecref.RecordId);
      until CustomerRecref.Next = 0;
    CustomerRecref.Close;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CopyLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Copies all the links from a particular record.

Syntax

```
RecordRef.CopyLinks(FromRecordOrRecordRef: Variant)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FromRecordOrRecordRef
Type: Variant
Specifies the record from which you want to copy links.

Remarks

Use this method to copy all the links from a specified record and paste the links to the current record.

The link can be a link to a Web site, a file stored on the local or on a remote computer, or a link to a page in your application.

Example

The following example copies all links from a source record that is named VendorRecord to the currently open record in the Customer table. The source record is record 10000 from the Vendor table. The code opens the Customer table as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for field 1 (No.) in the Customer table. The SetRange Method (FieldRef) selects records in the range 20000 to 40000 from the Customer table and record 10000 from the Vendor table. The Find Method (RecordRef) searches the Customer table for the records in the filtered range. If the record that meets the filter criteria is found, the links from the Vendor record No. 10000 are copied to the customer records in the range 30000 to 40000. The record id of the record to which the links were copied is displayed in a message box. The process is repeated until there is no more record in the range. The Close Method (RecordRef) closes the table.

```
var
    MyFieldRef: FieldRef;
    CustomerRecref: RecordRef;
    VendorRecord: Record Vendor;
    Count: Integer;
    Text000: Label 'The links have been copied to %1';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetRange('30000' , '40000');
    VendorRecord.SetRange("No.", '10000');
    Count := 0;
    ifCustomerRecref.Find('-') then
      repeat
        Count := Count + 1;
        CustomerRecref.CopyLinks(VendorRecord);
        Message(Text000, CustomerRecref.RecordId);
      until CustomerRecref.Next = 0;
    CustomerRecref.Close;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Counts the number of records that are in the filters that are currently applied to the table referred to by the RecordRef.

Syntax

```
Number :=   RecordRef.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Number Type: Integer The number of records in the table.

Remarks

This method returns the number of records that meet the conditions of any filters associated with the records. If no filters are set, the method shows the total number of records in the table.

NOTE

The Count method does not lock the table before it retrieves the number of records in the table. This means that the method reads both uncommitted and committed data, which could cause the number of records that is returned to be inaccurate. To make sure that the count is accurate, use the LockTable Method (RecordRef) before you use the Count method.

In previous versions of Dynamics 365, the Count method ignored security filters and always returned the total number of records unless you called the SetPermissionFilter method to get a filtered count. In Dynamics 365 Business Central, the Count method adheres to the SecurityFiltering Property. .

This method works just like the Count Method (Record).

Example

The following example opens table number 18 (Customer) as a RecordRef that is named MyRecordRef. The LockTable Method (RecordRef) locks the table. The Count method then retrieves the number of records in the table. The number of records is stored in the Count variable. The name of the table and the number of records in the table is displayed in a message box. The varTableNo variable can be used to open any table and get the

number of records in that table by changing the value of the varTableNo variable.

```
var
    MyRecordRef: RecordRef;
    varTableNo: Integer;
    Count: Integer;
    Text000: Label 'The number of records in the %1 table is: %2.';
begin
    varTableNo := 18;
    MyRecordRef.Open(varTableNo);
    MyRecordRef.LockTable;
    Count := MyRecordRef.Count;
    Message(Text000, MyRecordRef.Name, Count);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CountApprox Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets an approximate count of the number of records in the table

Syntax

```
Number :=   RecordRef.CountApprox()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Number Type: Integer Approximate number of records in the table.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CurrentCompany Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current company of a database table referred to by a RecordRef.

Syntax

```
Company :=   RecordRef.CurrentCompany()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Company Type: String The name of the current company.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CurrentKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current key of the table referred to by the RecordRef. The current key is returned as a string.

Syntax

```
CurrentKey :=   RecordRef.CurrentKey()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

CurrentKey Type: String The name of the current key of the record.

Example

```
var
    RecRef: RecordRef;
    Text000: Label 'The current key in the "%1" table is "%2".';
begin
    RecRef.Open(18);
    Message(Text000,RecRef.Caption,RecRef.CurrentKey);
end;
```

`RecRef.Open(18)` - Opens table 18 or causes a run-time error if table 18 does not exist.

`RecRef.Caption` - Returns the caption of the table.

`RecRef.CurrentKey` - Returns the caption of the current key in the table.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.CurrentKeyIndex Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the current key of the table referred to by the RecordRef. The current key is set or returned as a number. This first key = 1, and so on. If RecordRef does not have an active record, CurrentKeyIndex will return -1. If this value is then passed to KeyIndex, an index out of bounds error will occur. Therefore it is important to implement a check of the RecordRef parameter.

Syntax

```
[CurrentKeyIndex := ]  RecordRef.CurrentKeyIndex([NewKeyIndex: Integer])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

NewKeyIndex
Type: Integer
The number of the new key.

Return Value

CurrentKeyIndex Type: Integer The number of the current key.

Example

The following example loops through four tables (36-39) opens each table as a RecordRef variable that is named MyRecordRef. The CurrentKeyIndex method retrieves the current key index of the tables. The name of the table and the current key index of each table are displayed in a message box. Each table is close before the next one is opened.

```
var
    MyRecordRef: RecordRef;
    CurrentKeyIndex: Integer;
    i: Integer;
    varFromTable: Integer;
    varToTable: Integer;
    Text000: Label 'Table: %1  Current key index: %2';
begin
    varFromTable := 36;
    varToTable := 39;
    for i := varFromTable to varToTable do begin
      MyRecordRef.Open(i);
      CurrentKeyIndex := MyRecordRef.CurrentKeyIndex;
      Message(Text000, MyRecordRef.Name, CurrentKeyIndex);
      MyRecordRef.Close;
    end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Delete Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes a record in a table.

Syntax

```
[Ok := ]  RecordRef.Delete([RunTrigger: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

RunTrigger
Type: Boolean
Specifies whether the code in the OnDelete trigger will be executed. If this parameter is true, the code will be executed. If this parameter is false, then the code in the OnDelete trigger is not executed. The default value is false. This parameter is optional.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The current key and any filters on the record do not affect this operation. The record to delete is identified by the values in its primary key.

If an end-user modifies a record between the time that another end-user or another process reads the record and modifies it, then the second user must refresh the value of the record variable before editing the record. Otherwise, the end-user receives the following run-time error:

Another user has modified the record for this <Table Name> after you retrieved it from the database.

Enter your changes again in the updated window, or start the interrupted activity again.

In earlier versions of Dynamics 365, certain situations allowed code that an end-user runs to modify a record after a newer version of the record was written and committed to the database. This would overwrite the newer changes. However, in Dynamics 365 Business Central, we have restricted the Modify Method (RecordRef), Rename Method (RecordRef), and **Delete** Method (RecordRef) so that the end-user receives the following run-time error in these certain situations:

Unable to change an earlier version of the <Table Name> record. The record should be read from the database again. This is a programming error.

You must design your application so that you use the most up-to-date version of the record for modifications to the database. You use the Get Method (RecordRef) to refresh the record with the latest version.

Example

The following example deletes a record from the Customer table. The code starts by opening the **Customer** table (18) as a RecordRef variable that is named MyRecordRef. The Field Method (RecordRef) creates a FieldRef that is named MyFieldRef for field 1, which is the primary key of the **Customer** table. The Value Method (FieldRef, TestPage Field) assigns the value 10000 to the field that the MyFieldRef variable refers to. The Find Method (RecordRef) searches the table for a record with field 1 = 10000. If the record is found, then it is deleted, the table is modified, and a message is displayed.

```
var
    varRecordToDelete: Code;
    MyRecordRef: RecordRef;
    Text000: Label 'Customer %1 is deleted.;
begin
    varRecordToDelete := '10000';
    MyRecordRef.Open(18);
    MyFieldRef := MyRecordRef.Field(1);
    MyFieldRef.Value := varRecordToDelete;
    if MyRecordRef.Find('=') then begin
      ifMyRecordRef.Delete then begin
        MyRecordRef.Modify;
        Message(Text000, MyFieldRef.Value);
      end;
    end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.DeleteAll Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes all records in a table that fall within a specified range.

Syntax

```
RecordRef.DeleteAll([RunTrigger: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

RunTrigger
Type: Boolean
Specifies whether the code in the OnDelete trigger will be executed. If this parameter is true, the code will be executed. If this parameter is false (default), the code will not be executed. This parameter is optional.

Remarks

This method works the same way as the DeleteAll Method (Record).

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecRef. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for field 1 (No.). From the No. field, the SetRange Method (FieldRef) selects records in the range from 10000 to 20000. The number of records in the range is displayed in a message box. The DeleteALL method deletes all records in that range. The number of records is displayed again. This time, 0 is displayed because all the records in the range are deleted.

```
var
    MyFieldRef: FieldRef;
    CustomerRecRef: RecordRef;
    Text000: Label 'The number of records in the range is %1.;
begin
    CustomerRecRef.Open(18);
    MyFieldRef := CustomerRecRef.Field(1);
    MyFieldRef.SetRange('10000' , '20000');
    Message(Text000 ,CustomerRecRef.Count);
    CustomerRecRef.DeleteALL;
    Message(Text000 ,CustomerRecRef.Count);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.DeleteLink Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes a specified link from a record in a table.

Syntax

```
RecordRef.DeleteLink(ID: Integer)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

ID

Type: Integer

The ID of the link you want to delete.

Remarks

When you add a link to a page or a table, an entry is created in the Record Link system table. Each entry is given an ID. This ID is specified as a parameter in the DeleteLINK method.

Example

The following example deletes a link from a customer record in the Customer table. The code starts by opening table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for the first field in the table (No.). `MyFieldRef.Value` selects record 01121212 from the No. field. This record is initialized in the CustomerNum variable. The Find Method (RecordRef) searches for record 01121212. If the record is found, the DeleteLINK method deletes the link that is specified in the varLinkid variable. A message that states that the link is deleted is displayed in a message box. You can verify that the link is deleted in the **Links** FactBox on the Customer List or Customer Card pages.

```
var
    MyFieldRef: FieldRef;
    CustomerRecRef: RecordRef;
    varLinkid: Integer;
    CustomerNUm: Integer;
    Text000: Label 'The link with id %1 is deleted.;
    Text001: Label 'The customer cannot be found.';
begin
    CustomerNum := '01121212';
    varLinkid := 21;
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value := CustomerNum;
    if CustomerRecref.Find('=') then begin
      CustomerRecref.DeleteLink(varLinkid);
      Message(Text000, varLinkid);
    end else
      Message(Text001);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.DeleteLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes all of the links that have been added to a record.

Syntax

```
RecordRef.DeleteLinks()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Example

The following example deletes all links from a customer record in the Customer table. The code starts by opening table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for the first field in the table (No.). `MyFieldRef.Value` selects record 01121212 from the No. field. This record is initialized in the CustomerNum variable. The Find Method (RecordRef) searches for record 01121212. If the record is found, the DeleteLINKS method deletes all the links in the record. A message that states that the links are deleted is displayed in a message box. You can verify that the links are deleted in the **Links** FactBox on the Customer List or Customer Card pages.

```
var
    MyFieldRef: FieldRef;
    CustomerRecRef: RecordRef;
    CustomerNUm: Integer;
    Text000: Label 'The link with id %1 is deleted.;
    Text001: Label 'The customer cannot be found.';
begin
    CustomerNum := '01121212';
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value := CustomerNum;
    if CustomerRecref.Find('=') then begin
      CustomerRecref.DeleteLinks;
      Message(Text000, CustomerNum);
    end else
      Message(Text001);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Duplicate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Duplicates the table that contains the RecordRef.

Syntax

```
RecordRef :=   RecordRef.Duplicate()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

RecordRef Type: RecordRef A RecordRef that refers to a new record with the same filters, current keys, and marks as the original RecordRef.

Remarks

The RecordRef that is returned refers to a new record with the same filters, current keys, and marks as the original RecordRef. Any changes that you make to the filters, current keys, and marks of the new record are not observed in the original. This differs from assigning one RecordRef to another RecordRef. If you assign one RecordRef to another RecordRef, then both refer to the same record and changes that you make to one RecordRef are observed in the other RecordRef.

Example

The following example opens table 18 (Customer) as a RecordRef variable named RecordRef1 and uses the DUPLICATE method to copy the filters, current keys and marks from RecordRef1 into a new RecordRef variable named RecordRef2. After the DUPLICATE method is executed, the RecordRef1 and RecordRef2 variables are identical.

```
var
    RecordRef1: RecordRef;
    RecordRef2: RecordRef;
    Text000: Label 'RecordRef1 refers to the %1 table.\\ RecordRef2 refers to the %2 table.';
begin
    RecordRef1.Open(18);
    RecordRef2 := RecordRef1.Duplicate;
    Message(Text000, RecordRef1.Caption, RecordRef2.Caption);
end;
```

See Also

RecordRef Data Type
Getting Started with AL

RecordRef.Field Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a FieldRef for the field that has the number FieldNo in the table that is currently selected. If no field has this number, the method returns an error.

Syntax

```
Field :=   RecordRef.Field(FieldNo: Integer)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FieldNo

Type: Integer

The number that the field has in the table that is currently selected. This is the field for which you want the FieldRef.

Return Value

Field Type: FieldRef A new FieldRef of the record

Remarks

This method returns an error if the record is not opened and if the field is not found.

You might obtain better performance by using the FieldIndex Method (RecordRef).

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named MyRecordRef. MyRecordRef uses the Field method to create a reference to the No. field (field 1). The value in the No. field is then set to a specified record No. In this example, the record is set to 30000. The Find Method (RecordRef) method.md records for record 30000. If record is found, the Field method retrieves the value in the Name field (field 2), stores it in the varOldName variable and displays it in a message box. The Value Method (FieldRef, TestPage Field) changes the value in the Name field to a new name. In this example, the new name is 'Contoso'. The table is then modified to reflect this change and the new value in the Name field is retrieved and displayed in a message box. You can specify any record in the table and change the value in the Name field.

```
var
    MyRecordRef: RecordRef;
    MyFieldRef: FieldRef;
    varOldName: FieldRef;
    varNewName: Text;
    MyRecord: Code;
begin
    MyRecord := '30000';
    varNewName := 'Contoso';
    MyRecordRef.Open(18);
    MyFieldRef := MyRecordRef.Field(1);
    MyFieldRef.Value := MyRecord;
    if MyRecordRef.Find('=') then begin
      varOldName := MyRecordRef.Field(2);
      Message('Old Name: %1', varOldName);
      varOldName.Value := varNewName;
      MyRecordRef.Modify;
      Message('New Name: %1', MyRecordRef.Field(2));
    end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.FieldCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of fields in the table that are currently selected or returns the number of fields that have been defined in a key. Returns an error if no table or no key is selected.

Syntax

```
Count :=   RecordRef.FieldCount()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Count Type: Integer The number of fields in the table.

Example

The following example loops through tables 3 through 5 and displays the number of fields that are defined in each table. The code starts by opening table 3 (Payment Terms) as a RecordRef variable that is named MyRecordRef. MyRecordRef variable uses the FieldCount method to retrieve the number of fields that are defined in the table and stores it in the varFieldCount variable. The name of each table and the total number of fields in the table are displayed in a message box. The table that is open is closed before the next one is open.

```
var
    MyRecordRef: RecordRef;
    varFieldCount: Integer;
    Text000: Label 'The %1 table contains %2 field\(s\).\\';
begin
    for i := 3 to 5 do begin
      MyRecordRef.Open(i);
      varFieldCount := MyRecordRef.FieldCount;
      Message(Text000, MyRecordRef.Name, varFieldCount);
      MyRecordRef.Close;
     end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL

RecordRef.FieldExist Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines if the field that has the number FieldNo exists in the table that is referred to by the RecordRef. Returns an error if no table is currently selected.

Syntax

```
Exist :=   RecordRef.FieldExist(FieldNo: Integer)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

FieldNo
Type: Integer
The FieldNo that you want to know whether exists in the table.

Return Value

Exist Type: Boolean **true** if the field exists; otherwise **false**.

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named MyRecordRef. The code loops through fields 1 through 12 and uses the FieldEXIST method to determine whether the specified field exists. If the field exists, the name of the field and a message that indicates that the field exists is displayed. Otherwise, a message that indicates that the field does not exist is displayed.

```
var
    MyRecordRef: RecordRef;
    i: Integer;
    VarFieldName: FieldRef;
    Text000: Label 'The %1 table contains %2 field\(s\).\\';
begin
    MyRecordRef.Open(18);
    for i := 1 to 12 do begin
      if MyRecordRef.FieldExist(i) then begin
         VarFieldName := MyRecordRef.Field(i);
         Message(Text000, i, VarFieldName.Name);
        end else
        Message(Text001, i);
      end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL

RecordRef.FieldIndex Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the FieldRef of the field that has the specified index in the table that is referred to by the RecordRef.

Syntax

```
Field :=   RecordRef.FieldIndex(Index: Integer)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Index
Type: Integer
The index of the field.

Return Value

Field Type: FieldRef The FieldRef of the field that has the specified index.

Remarks

The fields in the primary key are always listed first in the index. Therefore, the order of the fields in the index is not necessarily the same as the order of the fields in the table.

If the index is out of the range supplied or if no table is selected, then the method returns an error.

Example

```
var
    SalesInvHdr: RecordRef;
    FldRef: FieldRef;
    Str: Text[1024];
    Text001: Label 'Index 1: %1\\';
    Text002: Label 'Index 2: %2\\';
    Text003: Label 'Index 3: %3';
begin
    SalesInvHdr.Open(112);
    FldRef1 := SalesInvHdr.FieldIndex(1);
    FldRef2 := SalesInvHdr.FieldIndex(2);
    FldRef3 := SalesInvHdr.FieldIndex(3);
    Message(Text001 + Text002 + Text003, FldRef1.Caption, FldRef2.Caption, FldRef3.Caption);
end;
```

The message window displays the following:

- Index 1: No.
- Index 2: Sell-to Customer No.

- **Index 3: Bill-to Customer No.**

The following illustration shows the first fields in table 112, Sales Invoice Header, and shows the keys for table 112. The order of the fields in the index differs from the order of the fields in the table. The index lists the field in the primary key first.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.FilterGroup Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Changes the filter group that is being applied to the table. You can also use this method to return the number of the current filtergroup. You cannot return the number of the filtergroup and set a new filtergroup at the same time.

Syntax

```
[Group := ]  RecordRef.FilterGroup([NewGroup: Integer])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

NewGroup

Type: Integer

The ID of the new filter group.

Return Value

Group Type: Integer The ID of the filter group.

Remarks

A filtergroup can contain a filter for a RecordRef that has been set earlier with SetFilter or SetRange. The total filter applied is the combination of all the filters set in all the filtergroups.

Example

The following example determines the filtergroup that is set on the Customer table and then changes filtergroup to 1, which is the filtergroup that is applied globally to the entire application. The code starts by opening the Customer table with a RecordRef variable. The SetRecFilter Method (RecordRef) sets the values in the current key of the current record as a record filter. This filter is a standard filtergroup so it has a filtergroup number of 0. Then the FilterGroup method returns the number for the filtergroup. This filtergroup is a standard filter so the return value is 0. This value is stored in the varOrigGroup variable and displayed in a message box. The FilterGroup method changes the filtergroup to 1, which is the number for the global filtergroup. The new value is stored in the varCurrGroup variable and displayed in a message box.

```
var
    MyRecordRef: RecordRef;
    varOrigGroup: Integer;
    varCurrGroup: Integer;
    Text000: Label 'The original filtergroup is: %1';
    Text001: Label 'The current filtergroup is: %1';
begin
    MyRecordRef.Open(Database::Customer);
    MyRecordRef.SetRecFilter;
    varOrigGroup := MyRecordRef.FilterGroup;
    Message(Text000, varOrigGroup);
    varCurrGroup := MyRecordRef.FilterGroup(1);
    Message(Text001, varCurrGroup);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Find Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds a record in a table based on the values stored in the key fields.

Syntax

```
[Ok := ]  RecordRef.Find([Which: String])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Which
Type: String
Specifies how to perform the search. The table will be searched until the record is found or there are no more records. Each character in the string can be present only one time. You can use the following characters:

- = search for a record that equals the key values (default)
- > search for a record that is larger than the key values
- < search for a record that is less than the key values
- + search for the last record in the table (+ can only be used alone)
- - search for the first record in the table (- can only be used alone) You can combine the '=', '>', and '<' characters. If this parameter contains '=', '>', or '<', then you must assign value to all fields of the current and primary keys before you call Find.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Find retrieves the first record that meets the conditions set by the *Which* parameter and the filters associated with the record. The search path reflects the sort order defined by the current key. If the current key is not the primary key, there is a chance that several records might have the same values in current key fields. If this occurs, the sort order defined by the primary key is used as the search path.

Example

The following example opens table 18 (Customer) as a RecordRef variable named CustomerRecref. The value for the field 1 (No.) is set to a specified record. In this example, the field is set to record 40000. The code uses the Find method to find the record that matches. 40000 in the table. If the record is found, the number, name, address and city of the customer are displayed in message boxes. The values in the fields are retrieved by using the Field Method (RecordRef).

```
var
    CustomerRecref: RecordRef;
    MyFieldRef: FieldRef;
    varCustomerNo: Code;
    Text000: Label 'The customer was found.\\';
    Text001: Label 'Customer No. %1 is:\\%2';
    Text002: Label 'Sorry, that customer could not be found.';
begin
    varCustomerNo := '40000';
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value := varCustomerNo;
    if CustomerRecref.Find('=') then begin
      Message(Text000 + Text001,  CustomerRecref.Field(1), CustomerRecref.Field(2));
      Message('Address: %1, %2',  CustomerRecref.Field(5),  CustomerRecref.Field(7));
    end else
      Message(Text002);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.FindFirst Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the first record in a table based on the current key and filter.

Syntax

```
[Ok := ]  RecordRef.FindFirst()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You should use this method instead of Find('-') when you need only the first record.

You should use this method only when you explicitly want to find the first record in a table or set. Do not use this method in combination with repeat..until.

Example

The following example opens the Item table (27) as a RecordRef variable that is named ItemRecref. The FindFirst method searches for the first record in the table. If the record is found, the description and unit price of the item in the record are displayed in a message box. Otherwise, a message that indicates that the first item was not found is displayed.

```
var
    ItemRecref: RecordRef;
    Text000: Label 'The first item is %1 and the unit price is %2.';
    Text001: Label 'The first item was not found.';
begin
    ItemRecref.Open(27);
    if ItemRecref.FindFirst then
      Message(Text000, ItemRecref.Field(3),  ItemRecref.Field(18))
    else
      Message(Text001);
end;
```

See Also

RecordRef Data Type

RecordRef.FindLast Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the last record in a table based on the current key and filter.

Syntax

```
[Ok := ]  RecordRef.FindLast()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You should use this method instead of Find('+') when you need only the last record.

You should use this method only when you explicitly want to find the last record in a table or set. Do not use this method in combination with repeat..until.

Example

The following example opens the Item table (27) as a RecordRef variable that is named ItemRecref. The FindLast method searches for the last record in the table. If the record is found, the description and unit price of the item in the record are displayed in a message box. Otherwise, a message that indicates that the last item was not found is displayed.

```
var
    ItemRecref: RecordRef;
    Text000: Label 'The last item is %1 and the unit price is %2.';
    Text001: Label 'The last item was not found.';
begin
    ItemRecref.Open(27);
    if ItemRecref.FindLast then
      Message(Text000, ItemRecref.Field(3),  ItemRecref.Field(18))
    else
      Message(Text001);
end;
```

See Also

RecordRef Data Type

RecordRef.FindSet Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds a set of records in a table based on the current key and filter. FindSet can only retrieve records in ascending order.

Syntax

```
[Ok := ]  RecordRef.FindSet([ForUpdate: Boolean] [, UpdateKey: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

ForUpdate
Type: Boolean
Set this parameter to false if you do not want to modify any records in the set. Set this parameter to true if you want to modify records in the set. If you set this parameter to true, the LockTable method (RecordRef) is immediately performed on the table before the records are read.
UpdateKey
Type: Boolean
This parameter only applies if ForUpdate is true. If you are going to modify any field value within the current key, set this parameter to true.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You should use this method only when you explicitly want to loop through a recordset. You should only use this method in combination with repeat..until.

Furthermore, FindSet only allows you to loop through the recordset from the top down. If you want to loop from the bottom up, you should use Find('+').

The general rules for using FindSet are the following:

- FindSet(False,False) - Read-only. This uses no server cursors and the record set is read with a single server call.
- FindSet(True,False) - This is used to update non-key fields. This uses a cursor with a fetch buffer similar to Find('-').
- FindSet(True,True) - This is used to update key fields.

This method is designed to optimize finding and updating sets. If you set any or both of the parameters to **false**, you can still modify the records in the set but these updates will not be performed optimally.

This method works the same way as the FindSet Method (Record).

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named MyRecordRef. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef with the first field (No.). The SetFilter Method (FieldRef) uses the MyFieldRef variable to set a filter that selects records from 30000 to 32000. `MyRecordRef.Field(2)` creates a FieldRef for the second field (Name). The FindSet method finds the set of records based on the key and the filters that have been set. The *ForUpdate* parameters and *UpdateKeys* are both set to **False**. This makes the records in the set read-only. The record ID and name of each customer in the record set is displayed in a message box until no records are left in the record set.

```
var
    MyRecordRef: RecordRef;
    MyFieldRef: FieldRef;
    Text000: Label '%1: "%2" is found in the set of records.';
begin
    MyRecordRef.Open(18);
    MyFieldRef := MyRecordRef.Field(1);
    MyFieldRef.SetFilter('30000..32000');
    MyFieldRef := MyRecordRef.Field(2);
    if MyRecordRef.FindSet(False, False) then begin
      repeat
        Message(Text000 , MyRecordRef.RecordId, MyFieldRef.Value);
      until MyRecordRef.Next = 0;
    end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a record based on the ID of the record.

Syntax

```
[Ok := ]  RecordRef.Get(RecordID: RecordId)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

RecordID
Type: RecordId
The RecordID that contains the table number and the primary key of the table and is used to identify the record that you want to get.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method always uses the primary key for the table. It ignores any filters that are set, except security filters. Security filters are applied or ignored based on the Security Filter Mode. The current key and filters are not changed after you call this method. .

Example

The following example opens the Customer table with the RecordRef variable, RecRef. The code assigns the first field in the table, which is the No. field, to MyFieldRef variable. The variable is assigned a value of 30000 by using the Field Method (RecordRef). The RecordId Method (RecordRef) retrieves the record ID of the record that has a value of 30000 in the No. field. The Get method then uses the RecID variable then to retrieves the record.

```
var
    RecRef: RecordRef;
    MyFieldRef: FieldRef;
    RecID: RecordId;
begin
    RecRef.Open(Database::Customer);
    MyFieldRef := RecRef.Field(1);
    MyFieldRef.Value := '30000';
    if RecRef.Find('=') then begin
      RecID := RecRef.RecordId;
      RecRef.Get(RecID);
    end;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.GetBySystemId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.3.*

Gets a record based on the ID of the record. The RecordRef must already be opened.

Syntax

```
[Ok := ]  RecordRef.GetBySystemId(SystemId: Guid)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

SystemId
Type: Guid
The systemid which uniquely identifies the record that you want to get.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.GetFilters Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines which filters have been applied to the table referred to by the RecordRef.

Syntax

```
String :=   RecordRef.GetFilters()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

String Type: String Filters that have been applied to the table that is referred to by the RecordRef parameter.

Remarks

This method works just like the GetFilters Method (Record).

Property Value/Return Value

Filters that have been applied to the table that is referred to by the *RecordRef* parameter.

Example

The following example opens a table as a RecorRef variable. The variable, RecRef, is used with the GetFilters method to retrieve the filters that are applied in the Customer table. If filters are applied, they will be stored in the Filters1 variable. The Filters1 variable does not contain any filters because filters have not been set. Then the SetRecFilter Method (RecordRef) is used to set the value in the current key of the current record as a filter. The variable Filters2 will now contain No. as a filter.

```
var
    RecRef: RecordRef;
    Filters1: Text;
    Filters2: Text;
    Text000: TexConst ENU='Filters1 contains : %1  Filters2 contains: %2';
begin
    RecRef.Open(Database::Customer);
    Filters1 := RecRef.GetFilters;
    RecRef.SetRecFilter;
    Filters2 := RecRef.GetFilters;
    Message(Text000, Filters1, Filters2);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.GetPosition Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a string that contains the primary key of the current record.

Syntax

```
String :=   RecordRef.GetPosition([UseNames: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

UseNames
Type: Boolean
Indicates whether a reference to the field caption or the field number should be returned. The UseCaptions parameter is optional. If it is set to true (default value) or if it is empty, then the returned string contains references to field captions in the table with which the record is associated. If the parameter is set to false, then field numbers are used instead.

Return Value

String Type: String The name or number of the field that contains the primary key.

Remarks

This method works just like the GetPosition Method (Record).

Example

The following example opens the Customer table as a RecordRef that is named RecRef. The RecordRef variable uses the GetPosition method to retrieve the position of the primary key. The *UseCaptions* parameter is set to **true** so the caption of the field that contains the primary key is returned. If you set *UseCaptions* to **false**, the number of the field is returned.

```
var
    RecRef: RecordRef;
    varPrimaryKey: Text;
    Text000: Label 'The primary key is: %1.';
begin
    RecRef.Open(Database::Customer);
    varPrimaryKey := RecRef.GetPosition(True);
    Message(Text000, varPrimaryKey);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.GetTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the table of a Record variable and causes the RecordRef to refer to the same table.

Syntax

```
RecordRef.GetTable(Rec: Record)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Rec
Type: Record
Use this record variable to specify the table to which the RecordRefVar refers.

Remarks

Any filters that are applied to the *RecordVar* are also applied to the *RecordRefVar*.

Another way to select the table to which a RecordRef refers is to use the Open Method (RecordRef) and specify a table number in the parameters.

Example

The following example is an excerpt from codeunit 8, AccSchedManagement. It iterates through records in the G/L Account table. It sets some values on the fields of a new record in the Acc. Schedule Line table based on the current G/L Account record and inserts the new record into the Acc. Schedule Line table. It calls GetTable to cause a RecordRef variable to refer to the same table as the new Acc. Schedule Line record, and then calls the LogInsertion method from codeunit 423, Change Log Management to log the change. The LogInsertion method requires a RecordRef as a parameter.

This example assumes that the AccSchedLineNo variable has been assigned a value previously in the code.

```
var
    AccSchedLine: Record "Acc. Schedule Line";
    AccSchedLineNo: Integer;
    GLAcc: Record "G/L Account";
    RecRef: RecordRef;
    ChangeLogMgt: Codeunit "Change Log Management";
begin
    if GLAcc.Find('-') then
      repeat
        AccSchedLine.Init;
        AccSchedLine."Line No." := AccSchedLineNo;
        AccSchedLineNo := AccSchedLineNo + 10000;
        AccSchedLine.Description := GLAcc.Name;
        if GLAcc."Account Type" IN [GLAcc."Account Type"::Posting,GLAcc."Account Type"::Total,GLAcc."Account
Type"::"End-Total"] then begin
          AccSchedLine.Totaling := GLAcc."No.";
          AccSchedLine."Row No." := CopyStr(GLAcc."No.",1,MaxStrLen(AccSchedLine."Row No."));
        end;
        if GLAcc."Account Type" IN [GLAcc."Account Type"::Total,GLAcc."Account Type"::"End-Total"] then
          AccSchedLine."Totaling Type" := AccSchedLine."Totaling Type"::"Total Accounts"
        else
          AccSchedLine."Totaling Type" := AccSchedLine."Totaling Type"::"Posting Accounts";
        AccSchedLine.Insert;
        RecRef.GetTable(AccSchedLine);
        ChangeLogMgt.LogInsertion(RecRef);
      until GLAcc.Next = 0;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.GetView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a string that describes the current sort order, key, and filters on a table.

Syntax

```
String :=   RecordRef.GetView([UseNames: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

UseNames
Type: Boolean
If this parameter is true (default) or omitted, the returned string contains references to field captions in the table with which the record is associated. If this parameter is false, the returned string contains references to field numbers in the table with which the record is associated.

Return Value

String Type: String The string format is the same as the SourceTableView property on pages.

Remarks

If the SetView Method (RecordRef) has been executed, the *String* parameter will return the value set by SetView.

This method works the same way as the GetView Method (Record).

Example

The following example opens the Customer table as a RecordRef variable that is named RecRef. The RecRef variable uses the GetView method to retrieve the field that the table is sorted on and stores the value in the varView variable. The Customer table does not have any filters and keys set so no filters or keys are displayed. The *UseCaptions* parameter is set to **true** so the name of the field is displayed. If you set the *UseCaptions* to **false**, the field number will be displayed.

```
var
    RecRef: RecordRef;
    varView: Text;
    Text000: Label 'The current view of the table is: %1';
begin
    RecRef.Open(Database::Customer);
    varView := RecRef.GetView(True);
    Message(Text000, varView);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.HasFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a filter has been applied to the table that the RecordRef refers to.

Syntax

```
Ok :=   RecordRef.HasFilter()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if the table referred to by RecordRef has a filter; otherwise, **false**.

Remarks

This method works just like the HasFilter Method (Record).

Example

The following example opens the Customer table with a RecordRef variable that is named RecRef. The HasFilter method determines whether a filter has been applied in the Customer table. The method returns **false** because no filters are applied. The return value is stored in the varHasFilters variable. The SetRecFilter Method (RecordRef) is used to set a filter. The HasFilter method now returns **true**. This example requires that you create the following global variables and text constant.

```
var
    varHasFilters: Text;
    RecRef: RecordRef;
    Text000: Label 'Are there any filters? %1';
begin
    RecRef.Open(Database::Customer);
    VarHasFilters := RecRef.HasFilter;
    Message('Are there any filters? %1', VarHasFilters);
    RecRef.SetRecFilter;
    VarHasFilters := RecRef.HasFilter;
    Message(Text000, VarHasFilters);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.HasLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a record contains any links.

Syntax

```
Ok :=   RecordRef.HasLinks()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if the record contains any links, otherwise **false**.

Remarks

The link can be a link to a website, a file stored on the local computer or on a remote computer, or a link to a Dynamics 365 page.

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable for field 1 (No.) and stores the value in the MyFieldRef variable. The Value Method (FieldRef, TestPage Field) selects record number 20000. The HasLinks method determines whether the selected record has any links. The method returns **No** because there are no links in the record.

```
var
    CustomerRecref: RecordRef;
    MyFieldRef: FieldRef;
    varHasLinks: Boolean;
    Text000: Label 'Does this record have one or more links? %1.';
begin
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.Value := '20000';
    varHasLinks := CustomerRecref.HasLinks;
    Message(Text000, varHasLinks);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Init Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Initializes a record in a table.

Syntax

```
RecordRef.Init()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Remarks

This method assigns default values to each field in the record, including the SystemId field. The values that are assigned in the record correspond to those defined when the table was created. If no value was assigned when the table was created, the values are assigned based on the data type, as shown in the following table.

DATA TYPE	DEFAULT VALUE
BigInteger	0
BigText	<Empty>
BLOB	<Empty>
Boolean	No
Code	'' (empty string)
Date	0d (Undefined date)
DateFormula	'' (empty string)
DateTime	0DT (Undefined datetime)
Decimal	0.0
Duration	0
GUID	00000000-0000-0000-0000-000000000000
Integer	0

DATA TYPE	DEFAULT VALUE
Option	0
RecordID	<Empty>
TableFilter	<Empty>
Text	'' (empty string)
Time	0T (Undefined time)

NOTE

Primary key and timestamp fields are not initialized.

After the method runs, you can change the values in any or all of the fields before you call the Insert Method (RecordRef) to enter the record in the table. Be sure that the fields that make up the primary key contain values that make the total primary key unique. If the primary key is not unique (such as the record already exists), then the record is rejected.

The method works in the same way as the Init Method (Record).

Example

The following example opens a table 18 (Customer) with a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef variable that is named MyFieldRef for the field. The Init method initializes the values in the fields by using default values and then uses the Insert Method (RecordRef) to insert a new record. The new record is 1120. This is the primary key for the new record.

NOTE

In this example, the Init method is called before the primary key is assigned a value. The Init method does not initialize primary key fields. Therefore calling the Init method before or after you assign values to the primary key field does not make any difference.

```
var
    CustomerRecref: RecordRef;
    MyFieldRef: FieldRef;
    Text000: Label 'The value of the field before initialization is %1.';
    Text001: Label 'The value of the field after you insert the record is %1.';
begin
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    CustomerRecref.Init;
    Message('%1', MyFieldRef.Value);
    MyFieldRef.Value := '1120';
    CustomerRecref.Insert;
    Message('%1', MyFieldRef.Value);
end;
```

See Also

RecordRef Data Type

RecordRef.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts a record into a table without executing the code in the OnInsert trigger.

Syntax

```
[Ok := ]  RecordRef.Insert()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts a record into a table.

Syntax

```
[Ok := ]  RecordRef.Insert(RunTrigger: Boolean)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

RunTrigger
Type: Boolean
If this parameter is true, the code in the OnInsert Trigger is executed. If this parameter is false, the code in the OnInsert trigger is not executed. The default value is false.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.1.*

Inserts a record into a table.

Syntax

```
[Ok := ]  RecordRef.Insert(RunTrigger: Boolean, InsertWithSystemId: Boolean)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

RunTrigger
Type: Boolean
If this parameter is true, the code in the OnInsert Trigger is executed. If this parameter is false, the code in the OnInsert trigger is not executed. The default value is false.
InsertWithSystemId
Type: Boolean
If this parameter is true, the SystemId field of the record is given a value that you explicitly assign. If a value is not assigned, then the platform assigns one. If this parameter is false, the SystemId field is given a value that is auto-generated by the platform. The default value is false.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.IsDirty Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.0.*

Gets a boolean value that indicates whether the current in-memory instance of a record or filtered set of records has changed since being retrieved from the database.

Syntax

```
Dirty :=   RecordRef.IsDirty()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Dirty Type: Boolean **true** if the a table or filtered set of records has changed; otherwise, **false**.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.IsEmpty Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether any records exist in a filtered set of records in a table.

Syntax

```
Empty :=   RecordRef.IsEmpty()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Empty Type: Boolean **true** if the record or table is empty; otherwise, **false**.

Remarks

If you have not applied filters to the record, this method determines whether the table is empty. If you have applied filters, the method determines whether the filtered set of records is empty.

The number of filters that you have applied to the records affects the speed of the IsEmpty method. The fewer the number of filters, the faster the operation is performed.

When you are using SQL Server, this method is faster than using the Count Method (Record) and then testing the result for zero.

This method works the same as the IsEmpty Method (Record).

Example

The following example opens table 18, the customer table as a RecordRef variable that is named. CustomerRecref. The IsEmpty Method (RecordRef) determines whether the table is empty. The message box displays **false** because the Customer table is not empty. **false** represents **false**.

```
var
    CustomerRecref: RecordRef;
    Text000: Label 'Is the table empty? %1.';
begin
    CustomerRecref.Open(18);
    IsEmpty := CustomerRecref.IsEmpty;
    Message(Text000, IsEmpty);
end;
```

If you open table 78 (Printer Selection), the message will display **true** because the table is empty.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.IsTemporary Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a RecordRef refers to a temporary table.

Syntax

```
Temporary :=   RecordRef.IsTemporary()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Temporary Type: Boolean **true** if the RecordRef refers to a temporary table, otherwise **false**.

Remarks

In versions of Dynamics 365 earlier than Microsoft Dynamics NAV 2013, if a RecordID or a RecordRef referred to a temporary table, then the table number value of the RecordID or RecordRef was the run-time generated sequence ID, which is from the base value of 2000100000. You could use the table number to determine if a RecordID or a RecordRef referred to a temporary table. In Dynamics 365 Business Central, the table number value of a RecordID or a RecordRef always contains the ID of the originating physical table and not the run-time generated sequence ID. If you previously used the TableNo Method (RecordID) or the Number Method (RecordRef) to test for the sequence number and determine if the RecordID or RecordRef was temporary, then you must use the IsTemporary method in Dynamics 365 instead.

Example

This example shows that you can replace code that you used previously to determine if a RecordRef referred to a temporary table. This example requires that you create a RecordRef variable named RecordRefVar.

```
// Previous code
ifRecordRefVar.Number >= 2000100000 then begin
  // Code for temporary tables
end;

// New code
ifRecordRefVar.IsTemporary then begin
  // Code for temporary tables
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.KeyCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of keys that exist in the table that is referred to by the RecordRef. Returns an error if no table is selected.

Syntax

```
Count :=   RecordRef.KeyCount()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Count Type: Integer The number of keys that have been identified in the table.

Example

The following example opens the Customer table (18) as a RecordRef variable that is named CustomerRecref. The KeyCount method retrieves the number of keys that are defined in the Customer table. The return value of the method is stored in the KeyCount variable and displayed in a message box. The KeyCount variable contains the number 13 because 13 keys are defined in the Customer table.

```
var
    CustomerRecref: RecordRef;
    KeyCount: Integer;
    Text000: Label 'The table has %1 keys.';
begin
    CustomerRecref.Open(18);
    KeyCount := CustomerRecref.KeyCount;
    Message(Text000, KeyCount);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.KeyIndex Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the KeyRef of the key that has the index specified in the table that is currently selected. The key can be composed of fields of any supported data type. Data types that are not supported include BLOBs, FlowFilters, variables, and functions. If the sorting key is set to a field that is not part of a key, then the KeyIndex is -1.

Syntax

```
Key :=   RecordRef.KeyIndex(Index: Integer)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Index

Type: Integer

The number of the index in which you are interested.

Return Value

Key Type: KeyRef The KeyRef of the field that has the specified index.

Remarks

The first key in the index must have index 1, the second index 2, and so on. The last key must have index = KeyCount. If the specified index is out of the range or if no table is selected, the method returns an error.

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The loop starts from 1 and loops through the key indexes that are in the table. `CustomerRecref.KeyCount` returns the maximum number of keys that are defined in the table. The loop continues until the last key is reached. For each index that is specified, the KeyIndex method retrieves the KeyRef for the specified index. The key index and the KeyRef for the specified indexes are displayed in a message box.

```
var
    CustomerRecref: RecordRef;
    i: Integer;
    varKeyRef: KeyRef;
    Text000: Label 'KeyIndex: %1   KeyRef: %2';
begin
    CustomerRecref.Open(18);
      for i := 1 to CustomerRecref.KeyCount do begin
        varKeyRef := CustomerRecref.KeyIndex(i);
        Message(Text000, i, varKeyRef);
      end;
    CustomerRecref.Close;
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.LoadFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Accesses the table's corresponding data source and loads the values of the specified fields on the record.

Syntax

```
[Ok := ]  RecordRef.LoadFields(Fields: Integer,...)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Fields
Type: Integer
The FieldNo's of the fields to be loaded.

Return Value

Ok Type: Boolean **true** if all values were loaded on the record; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method will trigger a JIT load of the specified fields. The method allows for triggering the JIT load on multiple fields. If the fields are already loaded, another load won't be triggered. Using this method instead of of relying on implicit JIT loads lets you develop for more explicit error handling when a load fails.

The method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

See Also

Using Partial Records
RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.LockTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Locks a table to protect it from write transactions that conflict with each other.

Syntax

```
RecordRef.LockTable([Wait: Boolean] [, VersionCheck: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Wait
Type: Boolean
Specifies what to do if the table is already locked. If this parameter is true and if another application has already locked the table, the system will wait until the table is unlocked. If this parameter is false and if another application has already locked the table, a run-time error occurs.

VersionCheck
Type: Boolean
If this parameter is true, the version of the RecordRef will be checked. If this parameter is false, blank, or not used, the version will not be checked.

Remarks

Because all write operations automatically lock the table that is being used, LockTable would appear unnecessary. However, you could have a transaction in which an application wants to inspect data before possibly changing it, with a guarantee that the data being changed has not been modified by other applications since the read operation. The solution is to explicitly lock the table before the read operation. This makes sure that no other application makes changes between the read operation and the possible write operation.

The table lock is released (unlocked) when the transaction is committed.

This method works the same as the LockTable Method (Record).

Example 1

The following example opens table number 18 (Customer) as a RecordRef that is named MyRecordRef. The LockTable method locks the table. This ensures that no records are inserted or deleted during the counting process. The Count Method (RecordRef) then retrieves the number of records in the table. The number of records is stored in the Count variable. The name of the table and the number of records in the table is displayed in a message box. The varTableNo variable can be used to open any table and get the number of records in that table by changing the value of the varTableNo variable.

```
var
    CustomerRecref: RecordRef;
    Count: Integer;
    varTableNo: Integer;
    Text000: Label 'The number of records in the %1 table is: %2.';
begin
    varTableNo := 18;
    MyRecordRef.Open(varTableNo);
    MyRecordRef.LockTable;
    Count := MyRecordRef.Count;
    Message(Text000, MyRecordRef.Name, Count);
end;
```

Example 2

This example uses pseudo-language to show the scope of write locks. Both an explicit lock and an automatic lock are illustrated. The first line (1) explicitly locks table A. If this explicit lock was not set on table A, the Database Management System (DBMS) would automatically lock this table when a record is inserted (3). Table B is not locked explicitly, but is locked automatically by the DBMS when a record is inserted (4). Both locks are active until the system exits the AL code module (5).

```
BeginWriteTransaction
TableA.LockTable // (1)
FindRec(TableA, ...) // (2)
.
.
InsertRec(TableA,...) // (3)
.
InsertRec(TableB) // (4)
.
.
EndWriteTransaction // (5)
```

If a data update depends on a prior read operation and there is a long time between the read operation and the write operation, you may not want to lock the table as you usually would during a transaction. This enables you to prevent other users from updating the table until your transaction is committed.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Mark Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.3.*

Marks a record. You can also use this method to determine whether a record is marked.

Syntax

```
[Marked := ]  RecordRef.Mark([Mark: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Mark
Type: Boolean
Specifies if a record is marked.

Return Value

Marked Type: Boolean **true** if the record is marked; otherwise, **false**.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.MarkedOnly Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.3.*

Activates a special filter. After you use this function, your view of the table includes only records marked by this function.

Syntax

```
[MarkedOnly := ]  RecordRef.MarkedOnly([MarkedOnly: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

MarkedOnly

Type: Boolean

Activates a special filter.

Return Value

MarkedOnly Type: Boolean **true** if the special filter is being used; otherwise, **false**.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Modify Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Modifies a record in a table.

Syntax

```
[Ok := ]  RecordRef.Modify([RunTrigger: Boolean])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

RunTrigger
Type: Boolean
Specifies whether to run the AL code in the OnModify Trigger. If this parameter is true, then the code in the OnModify trigger is executed. If this parameter is false (default), then the code is not executed.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Select the record that you want to modify by using the primary key fields. The record's current key and filters do not affect the operation.

If an end-user modifies a record between the time that another end-user or another process reads the record and modifies it, then the second user must refresh the value of the record variable before editing the record. Otherwise, the end-user receives the following run-time error:

Another user has modified the record for this <Table Name> after you retrieved it from the database.

Enter your changes again in the updated window, or start the interrupted activity again.

In earlier versions of Dynamics 365, certain situations allowed code that an end-user runs to modify a record after a newer version of the record was written and committed to the database. This would overwrite the newer changes. However, in Dynamics 365 Business Central, we have restricted the **Modify** Method (RecordRef), Rename Method (RecordRef), and Delete Method (RecordRef) so that the end-user receives the following run-time error in these certain situations:

Unable to change an earlier version of the <Table Name> record. The record should be read from the database again. This is a programming error.

You must design your application so that you use the most up-to-date version of the record for modifications to the database. You use the Get Method (RecordRef) to refresh the record with the latest version.

This method works the same as the Modify Method (Record).

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Identifies the name of the table

Syntax

```
Name :=   RecordRef.Name()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Name Type: String The name of the table.

Remarks

This method works the same as the TableName Method (Record).

Example

The following example opens a table as a RecordRef variable that is named MyRecordRef. You can specify any table number in the Open Method (RecordRef). In this example, the table 18 (Customer) is open. The Name method retrieves the name of table 18 and stores it in the varTableName variable. The table number and name are displayed in a message box.

```
var
    MyRecordRef: RecordRef;
    varTableName: Text;
    Text000: Label 'Table %1 is the %2 table.';
begin
    TableNo := 18;
    MyRecordRef.Open(TableNo);
    varTableName := MyRecordRef.Name;
    Message(Text000, TableNo, varTableName);
end;
```

See Also

RecordRef Data Type
Getting Started with AL

RecordRef.Next Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Steps through a specified number of records and retrieves a record.

Syntax

```
[Steps := ]  RecordRef.Next([Steps: Integer])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Steps
Type: Integer
Defines the direction of the search and how many records to step include. If this parameter is greater than zero, the method will search the number of records specified in Steps forward in the table. If this parameter is less than zero, the method will search the number of records specified in Steps backward in the table. If this parameter is 0, no records are stepped over. If you do not specify this parameter, the method finds the next record.

Return Value

Steps Type: Integer Defines the direction of the search and how many records to include.

Remarks

This method locates a record positioned a given number of steps forward or backward from the record specified by *RecordRef*. Movement through the table is governed by the filters and the current key associated with the records. The fields in the record which will be compared with the current key fields must contain appropriate values before the method is called.

This method works the same as the Next Method (Record).

Example

The following example opens the Customer table as a RecordRef object, creates a reference to the first (No.) field, and stores the reference in the MyFieldRef variable. The SetRange method sets a filter that selects all records from 10000 to 40000 in the No. field. The Find Method (RecordRef) searches and selects the first record in the filter and counts the number of records that are found. The number of records is stored in the Count variable. The process is repeated by looping through all the records in the filter until no more records are found. The Next method steps through the records and finds the next record because no value is specified for the *Steps* parameter. The number of records that are found in the range is stored in the Count variable and displayed in a message box.

```
var
    CustomerRecref: RecordRef;
    MyFieldRef: FieldRef;
    Count: Integer;
    Text000: Label '%1 records were retrieved.';
begin
    CustomerRecref.Open(Database::Customer);
    MyFieldRef := CustomerRecref.Field(1);
    MyFieldRef.SetRange('10000' , '40000');
    Count := 0;
    if CustomerRecref.Find('-') then
      repeat
        Count := Count + 1;
      until CustomerRecref.Next = 0;
    Message(Text000 , Count);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Number Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the table ID (number) of the table that contains the record that was referred to by the RecordRef.

Syntax

```
No :=   RecordRef.Number()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

No Type: Integer The table ID of the table that contains the record that was referred to by the RecordRef.

Remarks

In versions of Dynamics 365 earlier than Microsoft Dynamics NAV 2013, if a RecordRef referred to a temporary table, then the table number value of the RecordRef was the run-time generated sequence ID, which is from the base value of 2000100000. You could use the table number to determine whether a RecordRef referred to a temporary table. In Dynamics 365 Business Central, the table number value of a RecordRef always contains the ID of the originating physical table and not the run-time generated sequence ID. If you previously used the Number Method (RecordRef) to test for the sequence number and determine whether the RecordRef was temporary, then you must use the IsTemporary Method (RecordRef) in Dynamics 365 instead.

Example

The following example opens the Customer table (18) as a RecordRef object. The Open Method (RecordRef) accepts `Database::Customer` as an integer. The Number method retrieves the table number and displays the name and number of the table in a message box.

```
var
    MyRecordRef: RecordRef;
    varDatabaseName: Integer;
    varTableNumber: Integer;
    Text000: Label '%1 is table %2.';
begin
    varDatabaseName := Database::Customer;
    MyRecordRef.Open(varDatabaseName);
    varTableNumber := MyRecordRef.Number;
    Message(Text000, MyRecordRef.Caption, varTableNumber);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Open Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Causes a RecordRef variable to refer to a table, which is identified by its number in a particular company.

Syntax

```
RecordRef.Open(No: Integer [, Temp: Boolean] [, CompanyName: String])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

No
Type: Integer
The number of the table.

Temp
Type: Boolean

CompanyName
Type: String
The name of the company to which you want to change. If you omit this parameter, the current company is used.

Remarks

When you use the RecordRef.Open method a new object is created. This object contains references to the open table, filters, and the record itself and all the fields it contains. You can open a table by using the table number or the name of the table that represents the table number. For example, you open the Customer table by using following syntax: `RecordRef.Open(18)` or `RecordRef.Open(Database::Customer)`

If you use the *CompanyName* parameter, then this method works the same as the ChangeCompany Method (Record).

Example 1

The following example uses the Open method to create a RecordRef variable that is named MyRecordRef for the Customer table. The parameters are omitted in this example because there is only one company in this example and the table will not be open as temporary table. The caption and number of records in the table are displayed in a Message box. At the end of the display, the Close Method (RecordRef) closes the table.

```
var
    MyRecordRef: RecordRef;
    Text000: Label 'The %1 table contains %2 records.';
begin
    MyRecordRef.Open(Database::Customer);
    Message(Text000, MyRecordRef.Caption, MyRecordRef.Count);
    MyRecordRef.Close;
end;
```

Example 2

This example shows how to use the Open method. In this example, "MyRecordRef" opens table 27 and then "Find('-')" finds the first record in the table. "TempMyRecordRef" opens a temporary table which is empty and therefore the "Find('-')" returns false.

```
var
    MyRecordRef: RecordRef;
    TempMyRecordRef: RecordRef;
begin
    MyRecordRef.Open(27,false);
    TempMyRecordRef.Open(27,true);

    if MyRecordRef.Find('-') then // This is true and will find the first record in the table.
      Message('MyRecordRef finds')
    else
      Message('MyRecordRef does not find');

    if TempMyRecordRef.Find('-') then // This is false because there are no records in a temporary table.
      Message('TempMyRecordRef finds')
    else
      Message('TempMyRecordRef does not find');
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.ReadConsistency Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether read consistency is enabled.

Syntax

```
Ok :=   RecordRef.ReadConsistency()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if read consistency is enabled; otherwise, **false**.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.ReadPermission Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines if you can read from a table.

Syntax

```
Ok :=   RecordRef.ReadPermission()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if you can read from some or all of the table; otherwise, **false**.

Remarks

This method can test for both full read permission and a partial read permission that has been granted with a security filter.

This method uses the filter that is currently applied to the *RecordRef* to determine whether you have read permission. If no filter is applied, the method tests for full read permission. If a filter has been set, the method only tests for read permission in the range of the filter.

To determine whether the user has a partial read permission because a security filter has been applied, view the **Permissions** page.

If you do not have permission to read from a table and you attempt to read, a run-time error occurs. This method lets you determine in advance if you have read permission. When the permissions are checked, the combination of the permissions in the license file and the user's permissions in the Permission table is considered.

This method works the same as the ReadPermission Method (Record).

Example

The following example opens table 18 (Customer) and creates a RecordRef variable that is named MyRecordRef for the table. The ReadPermission method determines whether the table has read permission and stores the return value in the varHasReadPerm variable. The Customer table has read permission, so the message displays **Yes**. You can initialize the varTableNo variable with any table number.

```
var
    MyRecordRef: RecordRef;
    varHasReadPerm: Boolean;
    varTableNo: Integer;
    Text000: Label 'Does the %1 table have read permission? %2';
begin
    varTableNo := 18;
    MyRecordRef.Open(varTableNo);
    varHasReadPerm := MyRecordRef.ReadPermission;
    Message(Text000, MyRecordRef.Name, varHasReadPerm);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.RecordId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the RecordID of the record that is currently selected in the table. If no table is selected, an error is generated.

Syntax

```
RecordID :=   RecordRef.RecordId()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

RecordID Type: RecordId The ID of the table.

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named MyRecordRef. The FindLast Method (RecordRef) finds the last record in the table. The record id of the last record is retrieved, stored in the RecID variable displayed in message box.

```
var
    MyRecordRef: RecordRef;
    RecID: RecordId;
    Text000: Label 'The record id for the last record is: %1';
begin
    MyRecordRef.Open(18);
    MyRecordRef.FindLast;
    RecID := MyRecordRef.RecordId;
    Message(Text000, RecID);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.RecordLevelLocking Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether record level locking is enabled.

Syntax

```
Ok :=   RecordRef.RecordLevelLocking()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean **true** if record level locking is enabled, otherwise **false**.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Rename Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Changes the value of a primary key in a table.

Syntax

```
[Ok := ]  RecordRef.Rename(Value1: Any [, Value2: Any,...])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Value1
Type: Any
The new values for the primary key.
Value2
Type: Any
The new values for the primary key.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You cannot rename some tables. Examples of the types of tables that you cannot rename are:

- Tables in which the user is not allowed to rename the Document No. for legal or business reasons.
- Tables in which an Option data type field, such as Document Type, is part of the primary key.

Some examples of tables that you cannot rename are:

- Table 36, Sales Header
- Table 38, Purchase Header
- Table 5405, Production Order
- Table 5766, Warehouse Activity Header

If an end-user modifies a record between the time that another end-user or another process reads the record and modifies it, then the second user must refresh the value of the record variable before editing the record. Otherwise, the end-user receives the following run-time error:

Another user has modified the record for this <Table Name> after you retrieved it from the database.

Enter your changes again in the updated window, or start the interrupted activity again.

In earlier versions of Dynamics 365, certain situations allowed code that an end-user runs to modify a record after a newer version of the record was written and committed to the database. This would overwrite the newer changes. However, in Dynamics 365 Business Central, we have restricted the Modify Method (RecordRef), **Rename** Method (RecordRef), and Delete Method (RecordRef) so that the end-user receives the following run-time error in these certain situations:

Unable to change an earlier version of the <Table Name> record. The record should be read from the database again. This is a programming error.

You must design your application so that you use the most up-to-date version of the record for modifications to the database. You use the Get Method (RecordRef) to refresh the record with the latest version.

Example

This example shows how to change the value of the primary key of a Record variable, and how to change the value of the primary key of a RecordRef variable.

```
var
    CustomerRecRef: RecordRef;
    NewNo1: Code;
    NewNo2: Code;
    result: Boolean;
    CustomerRec: Record Customer;
begin
    CustomerRec.Get('0112121');
    NewNo1 := '9999999';
    NewNo2 := '8888888';
    Message('Customer name: %1; Customer number: %2',CustomerRec.Name, CustomerRec."No.");
    result := CustomerRec.Rename(NewNo1);
    if result then
      Message('After rename - Customer name: %1; Customer number: %2',CustomerRec.Name, CustomerRec."No.")
    else
      Message('No rename.');
    CustRecRef.GetTable(CustomerRec);
    result := CustRecRef.Rename(NewNo2);
    if result then begin
      CustomerRec.Get(NewNo2);
      Message('After rename 2 - Customer name: %1; Customer number: %2',CustomerRec.Name, CustomerRec."No.")
    end else
      Message('No rename.');
end;
```

If a record with No. 0112121 is found, and if the renames are successful, then the following messages are displayed:

Customer name: Spotsmeyer's Furnishings; Customer number: 0112121

After rename - Customer name: Spotsmeyer's Furnishings; Customer number: 9999999

After rename 2 - Customer name: Spotsmeyer's Furnishings; Customer number: 8888888

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.Reset Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all filters, including any special filters set by the MarkedOnly method (Record), changes fields select for loading back to all, and changes the current key to the primary key. Also removes any marks on the record and clears any AL variables defined on its table definition.

Syntax

```
RecordRef.Reset()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Remarks

If no table is selected, this method returns an error.

This method works the same as the Reset Method (Record).

Example

The following example opens the Customer (18) table and creates a RecordRef variable that is named RecRef. The GetFilters Method (RecordRef) gets filters that have been applied to records in the table. The filters that are returned, if any, are stored in the Filters1 variable and displayed in message box. In this example, no filters are set so the message is blank. The SetRecFilter Method (RecordRef) sets a filter on the current key of the current record that is represented by the RecRef variable. The GetFilters Method (RecordRef) gets the filters that have been set and stores the value in the Filters2 variable. The message displays No. because the No. field is set as a filter. The Reset method removes the filter that was set. The value of the filter that is returned by the GetFilters Method (RecordRef) after the Reset Method (RecordRef) is executed is stored in the Filters3 variable. Filter3 is blank because the filter that was set by `RecRef.SetRecFilter;` is removed by the Reset method.

```
var
    RecRef: RecordRef;
    Filters1: Text;
    Filters2: Text;
    Filters3: Text;
    Text000: Label 'Filter before filter is set is: %1.';
    Text001: Label 'Filter after filter is set is: %1.';
    Text002: Label 'Filter before filter is reset is: %1.';
begin
    RecRef.Open(Database::Customer);
    Filters1 := RecRef.GetFilters;
    Message(Text000, Filters1);
    RecRef.SetRecFilter;
    Filters2 := RecRef.GetFilters;
    Message(Text001, Filters2);
    RecRef.Reset;
    Filters3 := RecRef.GetFilters;
    Message(Text002, Filters3);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SecurityFiltering Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets how security filters are applied to the RecordRef.

Syntax

```
[SecurityFiltering := ]  RecordRef.SecurityFiltering([NewSecurityFiltering: SecurityFilter])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

NewSecurityFiltering

Type: SecurityFilter

The new security filter for the RecordRef.

Return Value

SecurityFiltering Type: SecurityFilter The security filter applied to the RecordRef.

See Also

RecordRef Data Type

Getting Started with AL

Developing Extensions

RecordRef.SetLoadFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Sets the fields to be initially loaded when the record is retrieved from its data source. This will overwrite fields previously selected for initial load.

Syntax

```
[Ok := ]  RecordRef.SetLoadFields([Fields: Integer,...])
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Fields
Type: Integer
The FieldNo's of the fields to be loaded.

Return Value

Ok Type: Boolean **true** if all fields are selected for subsequent loads; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Calling SetLoadFields on a recordref without passing any fields will reset the fields selected to load to the default, where all readable normal fields are selected for load.

It is not necessary to include the following fields, because they are always selected for loading: Primary key, SystemId, and data audit fields (SystemCreatedAt, SystemCreatedBy, SystemModifiedAt, SystemModifiedBy).

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

Example

This code example uses the SetLoadFields method to speedup the calculation of the mean for values in a table field. The other fields aren't needed for the calculation, so they're not loaded.

```
procedure ComputeAritmeticMean(MyRecordRef: RecordRef; FieldNo: Integer): Decimal
var
    SumTotal: Decimal;
    Counter: Integer;
begin
    MyRecordRef.SetLoadFields(FieldNo);
    if MyRecordRef.FindSet() then begin
        repeat
            SumTotal := MyRecordRef.Field(FieldNo).Value;
            Counter += 1;
        until MyRecordRef.Next() = 0;
        exit(SumTotal / Counter);
    end;
end;
```

See Also

Using Partial Records
RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SetPermissionFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies the user's security filter to the referenced record. The security filter is combined with any other filters that are placed on the record with SetFilter or SetRange. The combined filter will not include any records outside the range of the security filter and this will prevent a runtime permission error from occuring when the record is read. If the permission filter is not set, an error can occur if you attempt to read a record that is outside the range of the user's security filter.

Syntax

```
RecordRef.SetPermissionFilter()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SetPosition Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the fields in a primary key on a record to the values specified in the String parameter. The remaining fields are not changed.

Syntax

```
RecordRef.SetPosition(String: String)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

String

Type: String

The string that is used to set the primary key. This string contains the primary key value to set.

Remarks

This method works the same as the SetPosition Method (Record).

Example

The following example changes the value in the primary key, the No. field, in table 23 (Vendor). Other fields are not changed. The code starts by opening table 23 (Vendor) as a RecordRef variable that is named MyRecordRef. The Field Method (RecordRef) selects the first field (No.) and stores the value in the MyFieldRef variable. The SetFilter Method (FieldRef) sets a filter that selects records from 10000 to 20000. The FindLast Method (RecordRef) finds and retrieves the last record in the record set. The SetPosition method changes the value in the No. field from 20000 to 20001. The record No. and the name of the record are displayed before and displayed again after the primary key value is changed. The string that contains the new primary key is initialized in the InputString variable.

```
var
    MyRecordRef: RecordRef;
    InputString: Text;
    MyFieldRef: FieldRef;
    Text000: Label 'The record No. before the primary key was changed is %1.\\ The vendor name before the
primary key was changed is %2.';
    Text001: Label 'The record No. after the primary key was changed is %1. \\ The vendor name after the
primary key was changed is %2';
begin
    InputString := 'No.=Const(20001)';
    MyRecordRef.Open(23);
    MyFieldRef := MyRecordRef.Field(1);
    MyFieldRef.SetFilter('10000..20000');
    MyRecordRef.FindLast;
    Message(Text000, MyRecordRef.RecordId, MyRecordRef.Field(2));
    MyRecordRef.SetPosition(InputString);
    Message(Text001, MyRecordRef.RecordId, MyRecordRef.Field(2));
end;
```

The following is displayed before the SetPosition method is called:

The record No. before the primary key was changed is Vendor: 20000.

The vendor name before the primary key was changed is AR Day property Management.

The following is displayed after the SetPosition method is called:

The record No. after the primary key was changed is Vendor: 20001.

The vendor name after the primary key was changed is AR Day property Management.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SetRecFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a filter on a record that is referred to by a RecordRef.

Syntax

```
RecordRef.SetRecFilter()
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Remarks

This method works the same as the SetRecFilter Method (Record).

Example

The following example opens the Customer table as a RecordRef variable that is named MyRecordRef. The SetRecFilter method sets the values in the current key of the current record as a record filter. The GetFilters Method (RecordRef) retrieves the filters that have been set and displays them in a message box. No. is displayed because the filter is set on the No. field, which is the current key.

```
var
    MyRecordRef: RecordRef;
    varFilters: Text;
    Text000: Label 'The filter is set on the %1 field.';
begin
    MyRecordRef.Open(Database::Customer);
    MyRecordRef.SetRecFilter;
    varFilters := MyRecordRef.GetFilters;
    Message(Text000, varFilters);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SetTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the table to which a Record variable refers as the same table as a RecordRef variable.

Syntax

```
RecordRef.SetTable(Rec: Record)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Rec
Type: Record
Specifies the Record that you want to refer to the table.

Remarks

Any filters that are applied to the RecordRef are also applied to the Record. If you change the filter that is applied to the RecordRef, you must call SetTable again to apply the new filter to the Record.

Example

This example shows that if you have a RecordID data type, you can get a RecordRef for the table that the RecordID refers to. Then you can use the RecordRef to set the table to which a Record variable refers.

```
var
    RecRef: RecordRef;
    InvtEventBuf: Record "Inventory Event Buffer";
    RecID: RecordId;
    ProdOrderComp: Record "Prod. Order Component";
begin
    InvtEventBuf.Find('-');
    RecID := InvtEventBuf."Source Line ID";
    RecRef := RecID.GetRecord;
    RecRef.SetTable(ProdOrderComp);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SetView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current sort order, key, and filters on a table.

Syntax

```
RecordRef.SetView(String: String)
```

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

String
Type: String
The string format is the same as the SourceTableView property on pages.

Remarks

The value of the *String* parameter can be returned by the GetView Method (RecordRef).

If the SetView method is executed with an empty string, all the filters are removed and the primary key is used.

If no table is selected, the SetView method fails.

This method works the same as the SetView Method (Record).

Example

The following example opens the Customer (18) table as a RecordRef variable that is named CustomerRecRef. The SetView method sets the sort key to the Name field, sort order to Ascending and sets a filter that selects records between 1000 and 2000. The GetView Method (RecordRef) retrieves the sort order, key and filters that have been set and stores the value in the ViewString variable. The value in the ViewString variable is displayed in a message box.

```
var
    CustomerRecRef: RecordRef;
    ViewString: Text;
    Text000: Label 'The following is the current sort order, key, and filters that are set: %1';
begin
    CustomerRecRef.Open(18);
    CustomerRecRef.SetView('Sorting(Name) Order(Ascending) Where(No.=Const(10000..20000))');
    ViewString := CustomerRecRef.GetView;
    Message(Text000, ViewString);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SystemCreatedAtNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Gets the field number that is used by the SystemCreatedAt field. The SystemCreatedAt field is a system field that the platform adds to all table objects.

Syntax

```
SystemCreatedAtFieldNo :=   RecordRef.SystemCreatedAtNo()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

SystemCreatedAtFieldNo Type: Integer The field number of the SystemCreatedAt field.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SystemCreatedByNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Gets the field number that is used by the SystemCreatedBy field. The SystemCreatedBy field is a system field that the platform adds to all table objects.

Syntax

```
SystemCreatedByFieldNo :=   RecordRef.SystemCreatedByNo()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

SystemCreatedByFieldNo Type: Integer The field number of the SystemCreatedBy field.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SystemIdNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets the field number that is used by the SystemId field. The SystemId field is a system field that the platform adds to all table objects.

Syntax

```
SystemIdFieldNo :=   RecordRef.SystemIdNo()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

SystemIdFieldNo Type: Integer The field number of the SystemId field.

Example

This example shows how to use the SystemIdNo method to retrieve the field number that is used by the SystemId field of a table.

```
var
    CustomerRec: Record Customer;
    SystemIdFieldNo: Integer;
    Text000: Label 'The field number is: %1.';

begin
    CustomerRec.Open(Database::Customer);
    SystemIdFieldNo := CustomerRec.SystemIdNo();
    Message(Text000, Format(SystemIdFieldNo));
end;
```

See Also

SystemId Field
RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SystemModifiedAtNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Gets the field number that is used by the SystemModifiedAt field. The SystemModifiedAt field is a system field that the platform adds to all table objects.

Syntax

```
SystemModifiedAtFieldNo :=   RecordRef.SystemModifiedAtNo()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

SystemModifiedAtFieldNo Type: Integer The field number of the SystemModifiedAt field.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.SystemModifiedByNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Gets the field number that is used by the SystemModifiedBy field. The SystemModifiedBy field is a system field that the platform adds to all table objects.

Syntax

```
SystemModifiedByFieldNo :=   RecordRef.SystemModifiedByNo()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

SystemModifiedByFieldNo Type: Integer The field number of the SystemModifiedBy field.

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

RecordRef.WritePermission Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines if you can write to a table.

Syntax

```
Ok :=   RecordRef.WritePermission()
```

NOTE

This method can be invoked using property access syntax.

Parameters

RecordRef Type: RecordRef An instance of the RecordRef data type.

Return Value

Ok Type: Boolean Specifies if you have permission to write to the table

Remarks

This method can test for both full write permission and a partial write permission that has been granted with a security filter. A write permission consists of Insert, Delete, and Modify permissions.

This method uses the filter that is currently applied to the *RecordRef* to determine whether you have write permission. If no filter is applied, the method tests for full write permission. If a filter has been set, the method only tests for write permission in the range of the filter.

To determine whether the user has partial write permission, because a security filter has been applied, view the **Permissions** page.

If you do not have permission to write to a table and you attempt to write, a run-time error occurs. This method lets you determine in advance if you have write permission. When the permissions are checked, the combination of permissions in the license file and the user's permissions in the Permission table is considered.

This method works the same as the WritePermission Method (Record).

Example

The following example opens table 18 (Customer) and creates a RecordRef variable that is named MyRecordRef for the table. The WritePermission method determines whether the table has write permission and stores the return value in the varHasWritePerm variable. The Customer table has write permission, so the message displays **Yes**. You can initialize the varTableNo variable with any table number.

```
var
    MyRecordRef: RecordRef;
    varTableNo: Integer;
    varHasWritePerm: Boolean;
    Text000: Label 'Does the %1 table have write permission? %2';
begin
    varTableNo := 18;
    MyRecordRef.Open(varTableNo);
    varHasWritePerm := MyRecordRef.WritePermission;
    Message(Text000, MyRecordRef.Name, varHasWritePerm);
end;
```

See Also

RecordRef Data Type
Getting Started with AL
Developing Extensions

Report Data Type

2/17/2021 • 7 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is used to display, print, or process information from a database.

The following methods are available on the Report data type.

METHOD NAME	DESCRIPTION
DefaultLayout(Integer)	Gets the default built-in layout type that is used on a specified report.
Execute(Integer, String [, RecordRef])	Runs a report in preview or processing-only mode without showing the request page in the client. The method gets the request page parameter values as an input parameter string from a RunRequestPage method call. The OnOpen and OnClose triggers on the request page will run even though the request page is not shown.
GetSubstituteReportId(Integer)	Gets the ID of the report that will be run by the platform after considering any substitutions made by extensions.
Print(Integer, String [, String] [, RecordRef])	Prints a specified report without running the request page. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from a RunRequestPage method call.
RdlcLayout(Integer, InStream)	Gets the RDLC layout that is used on a report and returns it as a data stream. The method has an instance call and a static call. The following code shows the syntax of the RDLC function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.
Run(Integer [, Boolean] [, Boolean] [, var Record])	Loads and executes the report that you specify.
RunModal(Integer [, Boolean] [, Boolean] [, var Record])	Loads and executes the report that you specify.
RunRequestPage(Integer [, String])	Runs the request page for a report without running the report. Returns an XML string that contains the request page parameters that are entered on the request page.
SaveAs(Integer, String, ReportFormat, var OutStream [, RecordRef])	Runs a specific report without a request page and saves the report as a PDF, Excel, Word, HTML, or XML file. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from the return value of a RunRequestPage method call.
SaveAsExcel(Integer, String [, var Record])	Saves a report on the computer that is running the server as a Microsoft Excel (.xls) workbook.

METHOD NAME	DESCRIPTION
SaveAsHtml(Integer, String [, var Record])	Saves a report as an HTML file. The file is saved on the computer where the server instance is running, and then downloaded to the client when ready. > This method is only supported when a report uses a Word report layout when it is run.
SaveAsPdf(Integer, String [, var Record])	Saves a report as a .pdf file.
SaveAsWord(Integer, String [, var Record])	Saves a report on the computer that is running the server as a Microsoft Word (.doc) document.
SaveAsXml(Integer, String [, var Record])	Saves the resulting data set of a query as an .xml file. The following code shows the syntax of the SaveAsXML function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.
WordLayout(Integer, InStream)	Gets the Word report layout that is used on a report and returns it as a data stream. The method has an instance call and a static call. The following code shows the syntax of the WORDLAYOUT function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.
WordXmlPart(Integer [, Boolean])	Returns the report data structure as structured XML that is compatible with Microsoft Word custom XML parts. The method has an instance call and a static call. The following code shows the syntax of the WORDXMLPART function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

The following methods are available on instances of the Report data type.

METHOD NAME	DESCRIPTION
Break()	Exits from a loop or a trigger in a data item trigger of a report or XmlPort.
CreateTotals(var Decimal [, var Decimal,...])	Maintains totals for a variable in AL.
CreateTotals(Array of [Decimal])	Maintains totals for a variable in AL.
DefaultLayout()	Gets the default built-in layout type that is used on a specified report.
Execute(String [, RecordRef])	Runs a report in preview or processing-only mode without showing the request page in the client. The method gets the request page parameter values as an input parameter string from a RunRequestPage method call. The OnOpen and OnClose triggers on the request page will run even though the request page is not shown.
IsReadOnly()	Gets if the current report's data access intent is readonly.
Language([Integer])	Gets or sets the current language setting for the report.
NewPage()	Forces a page break when printing a report.

METHOD NAME	DESCRIPTION
NewPagePerRecord([Boolean])	Gets or sets the current setting of the NewPagePerRecord property.
ObjectId([Boolean])	Gets or sets the name or number of the report.
PageNo([Integer])	Gets or sets the current page number of a report.
PaperSource(Integer [, Integer])	Gets or sets the paper source used for the current page or a specified page.
Preview()	Indicates whether a report is being printed in preview mode.
Print(String [, String] [, RecordRef])	Prints a specified report without running the request page. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from a RunRequestPage method call.
PrintOnlyIfDetail([Boolean])	Gets or sets the current settings of the PrintOnlyIfDetail property.
Quit()	Aborts the processing of a report or XmlPort.
RDLCLayout(var InStream)	Gets the RDLC layout that is used on a report and returns it as a data stream.The method has an instance call and a static call. The following code shows the syntax of the RDLC method. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.
Run()	Loads and executes the report that you specify.
RunModal()	Loads and executes the report that you specify.
RunRequestPage([String])	Runs the request page for a report without running the report. Returns an XML string that contains the request page parameters that are entered on the request page.
SaveAs(String, ReportFormat, var OutStream [, RecordRef])	Runs a specific report without a request page and saves the report as a PDF, Excel, Word, or XML file. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from the return value of a RunRequestPage method call.
SaveAsExcel(String)	Saves a report on the computer that is running the server as a Microsoft Excel (.xls) workbook.
SaveAsHtml(String)	Saves a report as an HTML file. The file is saved on the computer where the server instance is running, and then downloaded to the client when ready. > This method is only supported when a report uses a Word report layout when it is run.
SaveAsPdf(String)	Saves a report as a .pdf file.

METHOD NAME	DESCRIPTION
SaveAsWord(String)	Saves a report on the computer that is running the server as a Microsoft Word (.doc) document.
SaveAsXml(String)	Saves the resulting data set of a query as an .xml file.The following code shows the syntax of the SaveAsXML method. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.
SetTableView(var Record)	Applies the table view on the current record as the table view for the page, report, or XmlPort.
ShowOutput()	Returns the current setting of whether a section should be printed, and changes this setting.
ShowOutput(Boolean)	Returns the current setting of whether a section should be printed, and changes this setting.
Skip()	Skips the current iteration of the current report or XmlPort.
TotalsCausedBy()	Determines which field caused a group total to be calculated. This determines which field changed contents and thereby concluded a group.
UseRequestPage([Boolean])	Gets or sets whether a request page is presented to the user.
WordLayout(var InStream)	Gets the Word report layout that is used on a report and returns it as a data stream. The method has an instance call and a static call. The following code shows the syntax of the WORDLAYOUT method. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.
WordXmlPart([Boolean])	Gets the report data structure as structured XML that is compatible with Microsoft Word custom XML parts.

See Also

Report Object
Getting Started with AL
Developing Extensions

Report.DefaultLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the default built-in layout type that is used on a specified report.

Syntax

```
DefaultLayout :=   Report.DefaultLayout(Number: Integer)
```

Parameters

Number

Type: Integer

The ID of the report that you want to save. If the report that you specify does not exist, then a run-time error occurs.

Return Value

DefaultLayout Type: DefaultLayout The default built-in layout type that is used on a specified report.

Remarks

The default layout for a report is specified by the report's DefaultLayout Property.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Execute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs a report in preview or processing-only mode without showing the request page in the client. The method gets the request page parameter values as an input parameter string from a RunRequestPage method call. The OnOpen and OnClose triggers on the request page will run even though the request page is not shown.

Syntax

```
Report.Execute(Number: Integer, Parameters: String [, RecordRef: RecordRef])
```

Parameters

Number

Type: Integer

The ID of the report that you want to run. If the report that you specify does not exist, then a run-time error occurs.

Parameters

Type: String

A string of request page parameters as XML to use to run the report. The parameter string is typically retrieved from the return value a RunRequestPage method call.

RecordRef

Type: RecordRef

The RecordRef that refers to a record in a table.

Remarks

You typically use this method together with the RunRequestPage Method method. The RunRequestPage Method) runs a report request page without actually running the report, but instead, returns the parameters that are set on the request page as a string. You can then call the Execute method to get the parameter string and run the report.

For RunRequestPage Method method topic.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.GetSubstituteReportId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the ID of the report that will be run by the platform after considering any substitutions made by extensions.

Syntax

```
NewReportId :=   Report.GetSubstituteReportId(ReportId: Integer)
```

Parameters

ReportId

Type: Integer

The ID of the report for which you want to retrieve the ID of the possible report substitute.

Return Value

NewReportId Type: Integer The ID of the report that will be run by the platform after considering any substitutions made by extensions.

See Also

Substituting Reports
Report Data Type
Getting Started with AL
Developing Extensions

Report.Print Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Prints a specified report without running the request page. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from a RunRequestPage method call.

Syntax

```
Report.Print(Number: Integer, Parameters: String [, PrinterName: String] [, RecordRef: RecordRef])
```

Parameters

Number
Type: Integer
The ID of the report that you want to print. If the report that you specify does not exist, then a run-time error occurs.

Parameters
Type: String
A string of request page parameters as XML to use to run the report. The parameter string is typically retrieved from the return value a RunRequestPage method call.

PrinterName
Type: String
The name of the printer to use to print the report. The printer must be set up on the client computer. If you do not set this variable, the printer that is set as the default printer is used.

RecordRef
Type: RecordRef
The RecordRef that refers to the table in which you want to find a record.

Remarks

You typically use this method together with the RunRequestPage Method. The RunRequestPage method runs a report request page without actually running the report, but instead, returns the parameters that are set on the request page as a string. You can then call the Print method to get the parameter string and print the report.

For a simple example that illustrates how to use the Print method, see example in the RunRequestPage Method topic.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.RdlcLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the RDLC layout that is used on a report and returns it as a data stream. The method has an instance call and a static call. The following code shows the syntax of the RDLC function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

Syntax

```
[Ok := ]  Report.RdlcLayout(Number: Integer, InStream: InStream)
```

Parameters

Number
Type: Integer
The ID of the report object for which you want to get the RDLC layout.

InStream
Type: InStream
The variable in which to return the RDLC layout.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Using the return value is optional. When you use the return value, if the RDLC layout cannot be retrieved at run-time, then the system returns **false** and no error recorded. When you omit the return value, if the RDLC layout cannot be retrieved at run-time, then an error occurs, which states that the layout could not be retrieved.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the report that you specify.

Syntax

```
Report.Run(Number: Integer [, RequestWindow: Boolean] [, SystemPrinter: Boolean] [, var Record: Record])
```

Parameters

Number

Type: Integer

The ID of the report that you want to run.

RequestWindow

Type: Boolean

Specifies whether the request window for the report will be displayed. The request window is part of the report object.

SystemPrinter

Type: Boolean

Specifies whether to use the default Windows printer or use table 78, Printer Selection, to find the correct printer for this report.

Record

Type: Record

Specifies which record to use in the report. Any filters that are attached to the record that you specify are used.

Remarks

Use this method, or the Report.RunModal Method, if you do not know the specific report that you want to run when you are designing your application. If you do know the specific report that you want to run, then you can use the Run Method or the RunModal Method.

If the report you specify does not exist, then a compile error occurs.

NOTE

Internet browsers can only handle one file per request. Therefore, with the Web client, if this method is called in a repetitive statement (or loop) that generates multiple files, only the last file will be sent to the browser. Alternatively, when designing for the Web client, bundle the files in an archive file (.zip), for example, by using the methods found in codeunit **419 File Management**. For more details about this design pattern, see Multi-File Download. Although this article is written for Dynamics NAV, it is still relevant for Business Central. The methods in codeunit 419 are not external, therefore cannot be used in extensions. Instead, when developing extensions in AL, use the external methods of codeunit **425 Data Compression**. The approach is similar.

Example 1

This example shows how to run a report. This example displays the request window and sends the report to the printer that is selected in the Printer Selection table.

```
Report.Run(1001);
```

Example 2

This example shows how to run a report. This example skips the request window, starts the report immediately, and sends the report to the printer that is selected in the Printer Selection table.

```
Report.Run(1001, False);
```

Example 3

This example shows how to run a report. This example skips the request window and starts the report immediately. It sends the report to the system printer instead of the printer that is selected in the Printer Selection table.

```
Report.Run(1001, False, True);
```

Example 4

This example shows how to run a report for which you specify a record. This example displays the request window and sends the report to the system printer.

```
var
    MyRec: Record Customer;
begin
    MyRec.FindLast;
    MyRec.SetRecFilter;
    Report.Run(101, True, True, MyRec);
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the report that you specify.

Syntax

```
Report.RunModal(Number: Integer [, RequestWindow: Boolean] [, SystemPrinter: Boolean] [, var Record: Record])
```

Parameters

Number

Type: Integer

The ID of the report that you want to run.

RequestWindow

Type: Boolean

Specifies whether the request window for the report will be displayed. The request window is part of the report object.

SystemPrinter

Type: Boolean

Specifies whether to use the default Windows printer or use table 78, Printer Selection, to find the correct printer for this report.

Record

Type: Record

Specifies which record to use in the report. Any filters that are attached to the record that you specify are used.

Remarks

Use this method, or the Report.Run Method, if you do not know the specific report that you want to run when you are designing your application. If you do know the specific report that you want to run, then you can use the RunModal Method or the Run Method.

The request page is run modally when you use this method. However, when the user chooses **Preview** on the request page, the **Print Preview** page does not run modally.

NOTE

Internet browsers can only handle one file per request. Therefore, with the Web client, if this method is called in a repetitive statement (or loop) that generates multiple files, only the last file will be sent to the browser. Alternatively, when designing for the Web client, bundle the files in an archive file (.zip), for example, by using the methods found in codeunit **419 File Management**. For more details about this design pattern, see Multi-File Download. Although this article is written for Dynamics NAV, it is still relevant for Business Central. The methods in codeunit 419 are not external, therefore cannot be used in extensions. Instead, when developing extensions in AL, use the external methods of codeunit **425 Data Compression**. The approach is similar.

Example 1

This example shows how to run a report. This example displays the request window and sends the report to the printer selected through the Printer Selection table.

```
Report.RunModal(1001);
```

Example 2

This example shows how to run a report. This example skips the request window, starts the report immediately, and sends the report to the printer that is selected in the Printer Selection table.

```
Report.RunModal(1001, False);
```

Example 3

This example shows how to run a report. This example skips the request window and starts the report immediately. It sends the report to the system printer instead of the printer that is selected in the Printer Selection table.

```
Report.RunModal(1001, False, True);
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.RunRequestPage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs the request page for a report without running the report. Returns an XML string that contains the request page parameters that are entered on the request page.

Syntax

```
Parameters :=   Report.RunRequestPage(Number: Integer [, Parameters: String])
```

Parameters

Number

Type: Integer

The ID of the report for which you want to run the request page. If the report that you specify does not exist, then a run-time error occurs.

Parameters

Type: String

A string of request page parameters as XML to use to run the report.

Return Value

Parameters Type: String XML string that contains the request page parameters that are entered on the request page

Remarks

This method opens the request page for the specified report, where the user can provide parameters for the report. When the user closes the request page by choosing the **OK** button, a string that contains the parameter values that were set by the user is returned. When the user chooses the **Cancel** button, an empty string will be returned. The returned parameter string can be picked up by calling one of the following methods:

- Execute Method
- Print Method
- SaveAs Method

> **NOTE**
>
> You can use these methods to schedule reports in the job queue.

Because the request page runs in the context of where it was invoked from, users cannot bookmark a link to this page from the user interface.

Example

This example illustrates how to use the RunRequestPage method to run the request page for report ID 206 Sales Invoice. The request page parameters are saved to a table, and then uses the parameters with the Execute, SaveAs, and Print methods to preview the report, save it as a PDF file, and print it.

This example requires that you create a table for holding parameters that are entered on the report request page and a codeunit that runs the report methods.

Create a table called **Request Parameters** that has the following fields.

```
var
    ReportId: Integer;
    UserId: Code[100];
    Parameters: BLOB;
```

Create a codeunit and add the following code to the *OnRun* trigger of the codeunit.

```
var
    ReportParameters: Record "Report Parameters";
    XmlParameters: Text;
    OStream: OutStream;
    IStream: InStream;
    CurrentUser: Code[100];
    Content: File;
    TempFileName: Text;

begin
    // Use the Report.RunRequestPage method to run the request page to get report parameters
    XmlParameters := Report.RunRequestPage(206);
    CurrentUser := UserId;

    // Save the request page parameters to the database table
    with ReportParameters do begin
        // Cleanup
        if Get(206,CurrentUser) then
        Delete;

        SetAutoCalcFields(Parameters);
        ReportId := 206;
        UserId := CurrentUser;
        Parameters.CreateOutStream(OStream,TextEncoding::UTF8);
        Message(XmlParameters);
        OStream.WriteText(XmlParameters);

        Insert;
    end;

    Clear(ReportParameters);
    XmlParameters := '';

    // Read the request page parameters from the database table
    with ReportParameters do begin
        SetAutoCalcFields(Parameters);
        Get(206,CurrentUser);
        Parameters.CreateInStream(IStream,TextEncoding::UTF8);
        IStream.ReadText(XmlParameters);
    end;

    // Use the Report.SaveAs method to save the report as a PDF file
    Content.Create('TestFile.pdf');
    Content.CreateOutStream(OStream);
    Report.SaveAs(206,XmlParameters,ReportFormat::Pdf,OStream);
    Content.Close;

    // Use the Report.Execute method to preview the report
    Report.Execute(206,XmlParameters);

    // Use the Report.Print method to print the report
    Report.Print(206,XmlParameters);
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAs Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs a specific report without a request page and saves the report as a PDF, Excel, Word, HTML, or XML file. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from the return value of a RunRequestPage method call.

Syntax

```
[Ok := ]  Report.SaveAs(Number: Integer, Parameters: String, Format: ReportFormat, var OutStream: OutStream
[, RecordRef: RecordRef])
```

Parameters

Number

Type: Integer

The ID of the report that you want to save. If the report that you specify does not exist, then a run-time error occurs.

Parameters

Type: String

A string of request page parameters as XML to use to run the report. The parameter string is retrieved from the return value a RunRequestPage method call.

Format

Type: ReportFormat

The type of file to save the report as. The following options are supported: Pdf, Excel, Word, and XML.

OutStream

Type: OutStream

The stream to which to write a report.

RecordRef

Type: RecordRef

The RecordRef that refers to the table in which you want to find a record.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You typically use this method together with the RunRequestPage Method method. The RunRequestPage method runs a report request page without actually running the report, but instead, returns the parameters that are set on the request page as a string. You can then call the SaveAs method to get the parameter string and save the report to a file of the specified format.

For a simple example that illustrates how to use the SaveAs method, see example in the RunRequestPage Method method topic.

> **NOTE**
>
> By default, when a report uses an RDLC report layout at runtime, fonts are embedded in the generated PDF. You can specify whether fonts are embedded in the PDF for RDLC reports by changing the **Report PDF Font Embedding** setting in the Dynamics 365 Business Central service instance configuration or changing the **PDFFontEmbedding** property in report objects.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsExcel Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report on the computer that is running the server as a Microsoft Excel (.xls) workbook.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsExcel(Number: Integer, FileName: String [, var Record: Record])
```

Parameters

Number
Type: Integer
The ID of the report that you want to run. If the report you specify does not exist, then a run-time error occurs.

FileName
Type: String
The path and the name of the file that you want to save the report as. The path must exist, the file must not be used, and the server process must have permission to write to the file. Otherwise, you will get errors.

Record
Type: Record
Specifies which record to use in the report. Any filters that have been applied to the record that you specify will be used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the SaveAsExcel method on the global Report object and on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you do know which report you want to run, then create a Report variable, set the Subtype of the variable to a specific report, and use this variable when you call the SaveAsExcel method.

When you call the SaveAsExcel method, the report is generated and saved to "*FileName*." The request page is not shown.

If the destination folder that you specify in *FileName* does not exist, then you get the following error:

The specified path is invalid.

If the file that you specify in *FileName* is being used, then you get the following error:

An I/O exception occurred during the operation.

If the Dynamics 365 Business Central service process does not have permission to write to the file that you specify in *FileName*, then you get the following error:

Either the caller does not have the required permission or the specified path is read-only.

Example

This example shows how to use the SaveAsExcel method to save the Excel workbook to the Dynamics 365 Business Central service and then download the file to a computer that is running the Dynamics 365 application.

```
var
    TempFile: File;
    Name: Text[250];
    NewStream: InStream;
    ToFile: Text[250];
    ReturnValue: Boolean;
begin
    // Specify that TempFile is opened as a binary file.
    TempFile.TextMode(False);
    // Specify that you can write to TempFile.
    TempFile.WriteMode(True);
    Name := 'C:\Temp\TempReport.xls';
    // Create and open TempFile.
    TempFile.Create(Name);
    // Close TempFile so that the SaveAsExcel method can write to it.
    TempFile.Close;

    Report.SaveAsExcel(406,Name);

    TempFile.Open(Name);
    TempFile.CreateInStream(NewStream);
    ToFile := 'Report.xls';

    // Transfer the content from the temporary file on the
    // server to a file on the client.
    ReturnValue := DownloadFromStream(
      NewStream,
      'Save file to client',
      '',
      'Excel File *.xls| *.xls',
      ToFile);

    // Close the temporary file.
    TempFile.Close();
end;
```

You can create an action on a page and set the action to run this code. When you run the action, the **Export File** dialog box opens. Choose **Save** to save the file to the client.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsHtml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as an HTML file. The file is saved on the computer where the server instance is running, and then downloaded to the client when ready. > This method is only supported when a report uses a Word report layout when it is run.

NOTE

This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsHtml(Number: Integer, FileName: String [, var Record: Record])
```

Parameters

Number

Type: Integer

The ID of the report object that you want to run.

FileName

Type: String

The folder path and name of the file that you want to save the report as. The path must already exist and the service (login) account that is used by the server instance must have permission to write to the target folder. Otherwise, you will get errors.

Record

Type: Record

Specifies which record to use in the report. Any filters that have been applied to the record that you specify will be used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The SaveAsHTML method uses the logic in the codeunit **9651 Document Report Mgt.** code unit to handle the format transformation.

The SaveAsHTML method can be used on the Report data type and on Report variables. When you are programming the SaveAsHTML method, if you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you know which report you want to run, then create a Report variable, set the **Subtype** of the variable to a specific report, and then use this

variable when you call the SaveAsHTML method.

When you call the SaveAsXML method, the report is generated and saved to "FileName." The request page is not shown.

Reports that use an RDLC layout when run cannot be saved in the HTML format. A runtime error will occur if SaveAsHTML is used on an RDLC report.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsPdf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as a .pdf file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsPdf(Number: Integer, FileName: String [, var Record: Record])
```

Parameters

Number
Type: Integer
The ID of the report that you want to run.

FileName
Type: String
The path and name of the file that you want to save the report as.

Record
Type: Record
Specifies which record to use in the report. Any filters that have been applied to the record that you specify will be used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the **SaveAsPDF** method on the global Report object or on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you do know which report you want to run, then create a Report variable, set the Subtype of the variable to a specific report, and use this variable when you call the **SaveAsPDF** method.

When you call **SaveAsPDF**, the report is generated and saved to "*FileName*." A **Saving to PDF** window shows the status of the process. Note that the request page will not be shown.

The *FileName* parameter specifies a location on the computer that is running Dynamics 365 Business Central service. If you call this method from a client, such as from an action on a page, then use the DOWNLOAD Method (File) to download the .pdf file from the computer that is running Dynamics 365 Business Central service to the computer that is running the client.

> **NOTE**
>
> By default, when a report uses an RDLC report layout at runtime, fonts are embedded in the generated PDF. You can specify whether fonts are embedded in the PDF for RDLC reports by changing the **Report PDF Font Embedding** setting in the Dynamics 365 Business Central service instance configuration or changing the **PDFFontEmbedding** property in report objects.

Example

This example shows how to use the **SaveAsPDF** method to save a specific report as a PDF file on the computer that is running Dynamics 365 Business Central service.

```
var
    Filename: Text;
    ReturnValue: Boolean;
    Report206: Report " Sales - Invoice";
begin
    Filename := 'C:\MyReports\report206.pdf';
    ReturnValue := Report206.SaveAsPDF(Filename);
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsWord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report on the computer that is running the server as a Microsoft Word (.doc) document.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsWord(Number: Integer, FileName: String [, var Record: Record])
```

Parameters

Number
Type: Integer
The ID of the report that you want to run. If the report you specify does not exist, then a run-time error occurs.

FileName
Type: String
The path and the name of the file that you want to save the report as. The path must exist, the file must not be being used, and the server process must have permission to write to the file. Otherwise, you will get errors.

Record
Type: Record
Specifies which record to use in the report. Any filters that have been applied to the record that you specify will be used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the SaveAsWORD method on the global Report object or on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you do know which report you want to run, then create a Report variable, set the Subtype of the variable to a specific report, and use this variable when you call the SaveAsWORD method.

When you call the SaveAsWORD method, the report is generated and saved to "*FileName*." The request page is not shown.

If the destination folder that you specify in *FileName* does not exist, then you get the following error:

The specified path is invalid.

If the file that you specify in *FileName* is being used, then you get the following error:

An I/O exception occurred during the operation.

If the Dynamics 365 Business Central service process does not have permission to write to the file that you specify in *FileName*, then you get the following error:

Either the caller does not have the required permission or the specified path is read-only.

Example

This example shows how to use the SaveAsWORD method to save the Word document on the Dynamics 365 Business Central service, and then download the file to a different computer that is running the Dynamics 365 application.

```
var
    TempFile: File;
    Name: Text[250];
    NewStream: InStream;
    ToFile: Text[250];
    ReturnValue: Boolean;
begin
    // Specify that TempFile is opened as a binary file.
    TempFile.TextMode(False);
    // Specify that you can write to TempFile.
    TempFile.WriteMode(True);
    Name := 'C:\Temp\TempReport.doc';
    // Create and open TempFile.
    TempFile.Create(Name);
    // Close TempFile so that the SaveAsWORD method can write to it.
    TempFile.Close;

    Report.SaveAsWORD(406,Name);

    TempFile.Open(Name);
    TempFile.CreateInStream(NewStream);
    ToFile := 'Report.doc';

    // Transfer the content from the temporary file on the
    // server to a file on the client.
    ReturnValue := DownloadFromStream(
      NewStream,
      'Save file to client',
      '',
      'Word File *.doc| *.doc',
      ToFile);

    // Close the temporary file.
    TempFile.Close();
end;
```

You can create an action on a page and set the action to run this code. When you run the action, the **Export File** dialog box opens. Choose **Save** to save the file to the client.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as an .xml file. The following code shows the syntax of the SaveAsXML function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsXml(Number: Integer, FileName: String [, var Record: Record])
```

Parameters

Number
Type: Integer
The ID of the query object that you want to save as an .xml file. If the query that you specify does not exist, then a run-time error occurs.

FileName
Type: String
The path and name of the file that you want to save the query to.

Record
Type: Record
Specifies which record to use in the report. Any filters that have been applied to the record that you specify will be used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the SaveAsXML method on the global Report object and on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you know which report you want to run, then create a Report variable, set the **Subtype** of the variable to a specific report, and then use this variable when you call the SaveAsXML method.

When you call the SaveAsXML method, the report is generated and saved to "*FileName*." The request page is not shown.

If the destination folder that you specify in *FileName* does not exist, then you get the following error:

The specified path is invalid.

If the file that you specify in *FileName* is being used, then you get the following error:

An I/O exception occurred during the operation.

If the Dynamics 365 Business Central service does not have permission to write to the file that you specify in *FileName*, then you get the following error:

Either the caller does not have the required permission or the specified path is read-only.

To resolve this issue, verify that the service account that is running the Dynamics 365 Business Central service instance has write permissions on the path.

Example

This example shows how to use the SaveAsXML method to save a report as an .xml file on the Dynamics 365 Business Central service, and then download the file to a computer that is running the Dynamics 365.

```
var
    TempFile: File;
    Name: Text[250];
    NewStream: InStream;
    ToFile: Text[250];
    ReturnValue: Boolean;
begin
    // Specify that TempFile is opened as a binary file.
    TempFile.TextMode(False);
    // Specify that you can write to TempFile.
    TempFile.WriteMode(True);
    Name := 'C:\Temp\TempReport.xml';
    // Create and open TempFile.
    TempFile.Create(Name);
    // Close TempFile so that the SaveAsXML method can write to it.
    TempFile.Close;

    Report.SaveAsXML(406,Name);

    TempFile.Open(Name);
    TempFile.CreateInStream(NewStream);
    ToFile := 'Report.xml';

    // Transfer the content from the temporary file on
    // server to a file on the client.
    ReturnValue := DownloadFromStream(
      NewStream,
      'Save file to client',
      '',
      'Excel File *.xml| *.xml',
      ToFile);

    // Close the temporary file.
    TempFile.Close();
end;
```

You can create an action on a page and set the action to run this code. When you run the action, the **Export File** dialog box opens. Choose **Save** to save the file to the client.

See Also

Report Data Type

Report.WordLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the Word report layout that is used on a report and returns it as a data stream. The method has an instance call and a static call. The following code shows the syntax of the WORDLAYOUT function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

Syntax

```
[Ok := ]  Report.WordLayout(Number: Integer, InStream: InStream)
```

Parameters

Number

Type: Integer

The ID of the report object for which you want to get the Word report layout.

InStream

Type: InStream

The variable in which to return the Word report layout.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Using the return value is optional. When you use the return value, if the Word report layout cannot be retrieved at run-time, then the system returns **false** and no error recorded. When you omit the return value, if the Word report layout cannot be retrieved at run-time, then an error occurs, which states that the Word report could not be retrieved.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.WordXmlPart Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the report data structure as structured XML that is compatible with Microsoft Word custom XML parts. The method has an instance call and a static call. The following code shows the syntax of the WORDXMLPART function. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

Syntax

```
String :=   Report.WordXmlPart(Number: Integer [, ExtendedFormat: Boolean])
```

Parameters

Number

Type: Integer

The ID of the report that you want to run. If the report you specify does not exist, then a run-time error occurs.

ExtendedFormat

Type: Boolean

If you set this variable to true, then XML elements will include the following attributes:

- ElementType="Parameter|Column|DataItem". Specifies the element type as defined for the report in Report Designer. Parameter is typically used for elements, such as captions.
- ElementId="ID". Specifies the ID that is assigned to the element by its ID Property.
- DataType="Type". Specifies the data type of the element. If you omit this parameter or set it to false, then the element attributes are not included in the XML. This is the recommended setting when you will use the Word XML part in Word for modifying the report layout because the XML is simpler. The following example illustrates an XML element that has the ExtendedFormat set to true: <CompanyName ElementType="Column" ElementId="3" DataType="OemText"> The following example illustrates the same XML with the ExtendedFormat set to false:<CompanyName>

Return Value

String Type: String The report data structure as structured XML that is compatible with Microsoft Word custom XML parts.

Remarks

This method returns the data structure of the report as structured XML that complies with custom XML parts in Microsoft Word 2013. The following table provides a simplified overview of the XML that is returned by the method.

XML	DESCRIPTION
`<?xml version="1.0" encoding="utf-16"?>`	Header

XML	DESCRIPTION
`<NavWordReportXmlPart xmlns="urn:microsoft-../report/<reportname>/<id>/"`	XML namespace specification. `<reportname>` is the name assigned to the report object. `<id>` is the ID that is assigned to the report.

`..<Labels>`
 	
`....<ColumnNameCaption>ColumnNameCaption</ColumnNameCaption>`
 	
`....<LabelName>LabelCaption</LabelName>`
 	
`..</Labels>`
	Contains all the labels for the report. Labels are listed in alphabetical. The element includes labels that are related to columns that have the IncludeCaption Property set to **Yes** and labels that are defined in Report Label Designer.
	
	- Label elements that are related to columns have the format `<ColumnNameCaption>ColumnNameCaption</ColumnNameCaption>`, where `ColumnName` is determined by the column's Name Property.
	- Label elements from Report Label Designer have the format `<LabelName>LabelCaption</LableName`, where `LabelName` is determined by the label's Name Property and `LabelCaption` is determined by the label's Caption Property.

`..<DataItem1>`
 	
`....<DataItem1Column1>DataItem1Column1</DataItem1Column1>`
	Top-level data item and columns. Columns are listed in alphabetical order.
	
	The element names and values are determined by the Name property of the data item or column.

`....<DataItem2>`
 	
`......<DataItem2Column1>DataItem2Column1</DataItem2Column1>`
 	
`....</DataItem2>`
 	
`....<DataItem3>`
 	
`......<DataItem3Column1>DataItem3Column1</DataItem3Column1>`
 	
`....</DataItem3>`
	Data items and columns that are nested in the top-level data item. Columns are listed in alphabetical order under the respective data item.

`..</DataItem1>`
 	
`</NavWordReportXmlPart>`
	Closing elements.

Word custom XML parts enable you to integrate business data into Word documents. For example, the WORDXMLPART method is used internally by Dynamics 365 when you are creating report layouts in Word. You can use this method to create a custom XML part, and then, together with the SaveAsXML Method (Report) method and additional data merging tools, you can implement your own functionality for mapping and laying out report data in Word documents. To create a custom XML part, you can save the return value to an .xml file that is encoded in UTF-16 (16-bit Unicode Transformation Format). The resultant file can be added to Word documents as a custom XML part to map the report data set as XML data.

Example

The following example uses the WORDXMLPART method to save the data structure of Report 112 Sales Statistics in an XML file in a predefined folder *C:\Report Documents*. The resultant file can be used in Word as a custom

XML part.

```
var
    ReportAsString Text;
    SalesStatsReport: File;
begin
    ReportAsString := Report.WORDXMLPART(112);
    SalesStatsReport.TextMode(True);
    SalesStatsReport.WriteMode(True);
    SalesStatsReport.Create('C:\Report Documents\SalesStatsReport.xml', TextEncoding::UTF16);
    SalesStatsReport.WRITE(ReportAsString);
    SalesStatsReport.Close;
end;
```

The code generates the report structure as XML, and then writes the XML to the file *C:\Report Documents\SalesStatsReport.xml*.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Break Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exits from a loop or a trigger in a data item trigger of a report or XmlPort.

Syntax

```
Report.Break()
```

Parameters

Report Type: Report An instance of the Report data type.

Remarks

BREAK causes the current trigger to end. When used inside a loop, such as a WHILE-DO or REPEAT-UNTIL construction, BREAK interrupts the loop and causes the current trigger to end.

Compare this with the QUIT Method (Report, XMLport).

> **TIP**
> You can also use the AL BREAK Statement to exit an iteration or loop. The difference is that the BREAK statement does not terminate the trigger. It just exits the loop.

Example

```
var
    MyVar: Integer;
    Text000: Label "The variable is now %1.";
begin
    MyVar := 0;
    repeat
      MyVar := MyVar + 1;
      if MyVar = 5 then
        CurrReport.BREAK;
      Message(Text000,MyVar);
    until Myvar = 10;
    Message('After REPEAT-UNTIL loop'); //This statement is never called.
end;
```

When you run the previous code, the loop will end when MyVar is 5 and the execution of the current trigger ends. Statements after the loop are not executed.

See Also

Report Data Type
Getting Started with AL

Report.CreateTotals Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Maintains totals for a variable in AL.

Syntax

```
Report.CreateTotals(var Var1: Decimal [, var Var2: Decimal,...])
```

Parameters

Report Type: Report An instance of the Report data type.

Var1
Type: Decimal
Variable for which the system will maintain the total.

Var2
Type: Decimal
Variable for which the system will maintain the total.

Remarks

IMPORTANT

This method will be deprecated in a future update and we recommend that you do not use it.

CreateTOTALS maintains group and grand totals. The totals can be printed by placing controls that have the variable or variables that are the arguments of CreateTOTALS as their source expressions in the appropriate sections. The group totals are printed in GroupFooter sections, and the grand totals are printed in Footer sections.

This method is not supported on client report definition (RDLC) report layouts. In most cases, when you create a layout suggestion for a Classic report layout that uses the CreateTOTALS method, a SUM expression is created instead and no action is required.

Example

This example shows how to use the CreateTOTALS method to maintain totals for the two variables Amount and Quantity.

```
CurrReport.CreateTOTALS(Amount, Quantity);
```

See Also

Report Data Type

Report.CreateTotals Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Maintains totals for a variable in AL.

Syntax

```
Report.CreateTotals(Vars: Array of [Decimal])
```

Parameters

Report Type: Report An instance of the Report data type.

Vars
Type: Decimal
Array of variables for which the system will maintain individual totals.

Remarks

IMPORTANT

This method will be deprecated in a future update and we recommend that you do not use it.

CreateTOTALS maintains group and grand totals. The totals can be printed by placing controls that have the variable or variables that are the arguments of CreateTOTALS as their source expressions in the appropriate sections. The group totals are printed in GroupFooter sections, and the grand totals are printed in Footer sections.

This method is not supported on client report definition (RDLC) report layouts. In most cases, when you create a layout suggestion for a Classic report layout that uses the CreateTOTALS method, a SUM expression is created instead and no action is required.

Example

This example shows how to use the CreateTOTALS method to maintain totals for the two variables Amount and Quantity.

```
CurrReport.CreateTOTALS(Amount, Quantity);
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.DefaultLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the default built-in layout type that is used on a specified report.

Syntax

```
DefaultLayout :=   Report.DefaultLayout()
```

Parameters

Report Type: Report An instance of the Report data type.

Return Value

DefaultLayout Type: DefaultLayout The default built-in layout type that is used on a specified report.

Remarks

The default layout for a report is specified by the report's DefaultLayout Property.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Execute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs a report in preview or processing-only mode without showing the request page in the client. The method gets the request page parameter values as an input parameter string from a RunRequestPage method call. The OnOpen and OnClose triggers on the request page will run even though the request page is not shown.

Syntax

```
Report.Execute(Parameters: String [, RecordRef: RecordRef])
```

Parameters

Report Type: Report An instance of the Report data type.

Parameters

Type: String

A string of request page parameters as XML to use to run the report. The parameter string is typically retrieved from the return value a RunRequestPage method call.

RecordRef

Type: RecordRef

The RecordRef that refers to a record in a table.

Remarks

After the Execute method is executed, the system does not automatically clear the *Report* variable. You must handle clearing the variable.

You typically use this method together with the RunRequestPage Method. The RunRequestPage method runs a report request page without actually running the report, but instead, returns the parameters that are set on the request page as a string. You can then call the Execute method to get the parameter string and run the report.

For a simple example that illustrates how to use the Execute method, see example in the RunRequestPage Method topic.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.IsReadOnly Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 7.0.*

Gets if the current report's data access intent is readonly.

Syntax

```
DataAccessIntent :=   Report.IsReadOnly()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

Return Value

DataAccessIntent Type: Boolean A value specifying the readonly data access intent.

See Also

Report Data Type Getting Started with AL Developing Extensions

Report.Language Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the current language setting for the report.

Syntax

```
[CurrentLanguage := ]  Report.Language([Language: Integer])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

Language
Type: Integer
The new language setting for the report.

Return Value

CurrentLanguage Type: Integer The current language setting for the report.

Example

If you have documents that you want to print in the language of the recipient rather than in your own working language, you can add a single line of code in the document to handle this. This functionality is already enabled for most reports in the standard Business Central database. The document is printed in the language that is specified in the **Language Code** field on the **Customer Card** page.

In reports that need the multiple document languages functionality, you must insert the following AL code as the first line in the `OnAfterGet Record()` trigger:

```
Report.LANGUAGE := Language.GetLanguageID("Language Code")
```

For each of these reports, you must create a new variable, `Language`, with the data type `Record` pointing to the `Language` table. When you have compiled the object, it will no longer print in the user's working application language if another language has been specified on the **Customer Card** page.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.NewPage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Forces a page break when printing a report.

Syntax

```
Report.NewPage()
```

Parameters

Report Type: Report An instance of the Report data type.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.NewPagePerRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Gets or sets the current setting of the NewPagePerRecord property.

Syntax

```
[IsNewPagePerRecord := ]  Report.NewPagePerRecord([SetNewPagePerRecord: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

SetNewPagePerRecord

Type: Boolean

The new setting of the NewPagePerRecord property.

Return Value

IsNewPagePerRecord Type: Boolean The setting of the NewPagePerRecord property.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.ObjectId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the name or number of the report.

Syntax

```
String :=   Report.ObjectId([UseNames: Boolean])
```

Parameters

Report Type: Report An instance of the Report data type.

UseNames
Type: Boolean
true (default value) if the returned string contains the name of the report, **false** if the returned string contains the number of the report.

Return Value

String Type: String The name or number of the report.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.PageNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Gets or sets the current page number of a report.

Syntax

```
[CurrPageNo := ]  Report.PageNo([NewPageNo: Integer])
```

Parameters

Report Type: Report An instance of the Report data type.

NewPageNo
Type: Integer
The new page number of a report.

Return Value

CurrPageNo Type: Integer The current page number of a report.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.PaperSource Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Gets or sets the paper source used for the current page or a specified page.

Syntax

```
Report.PaperSource(PaperBinNo: Integer [, PhysicalPage: Integer])
```

Parameters

Report Type: Report An instance of the Report data type.

PaperBinNo
Type: Integer
The tray number.

PhysicalPage
Type: Integer
The page number.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Preview Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether a report is being printed in preview mode.

Syntax

```
IsPreview :=   Report.Preview()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

Return Value

IsPreview Type: Boolean **true** if the report is being printed in preview mode, otherwise **false**.

Remarks

You must use the return value. A compile error is generated if this value is not used.

If you run a client report definition (RDLC) report layout in preview mode and then call the CurrReport.PREView method, then the Print icon, Print Layout icon, Page Setup icon, and Save As icon are not displayed.

If you run a client report definition (RDLC) report layout in preview mode and do not call the CurrReport.PREView method, then you can print from the **Print Preview** window.

Example

This example shows how to use the PREView method. You can use this method in an application that stores a count of how many times a document has been printed in the database. This number must be updated from inside the report that is used to print the document. To avoid updating the number when the report is run in preview mode, add a construct to the OnPreDataItem trigger that resembles the one that is shown in this example when you add the code that updates the count.

```
if CurrReport.PREView then
... // Preview-specific processing.
else
... // Processing that is not preview specific:
... // Update the print count.
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Print Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Prints a specified report without running the request page. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from a RunRequestPage method call.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Report.Print(Parameters: String [, PrinterName: String] [, RecordRef: RecordRef])
```

Parameters

Report Type: Report An instance of the Report data type.

Parameters
Type: String
A string of request page parameters as XML to use to run the report. The parameter string is typically retrieved from the return value a RunRequestPage method call.

PrinterName
Type: String
The name of the printer to use to print the report. The printer must be set up on the client computer. If you do not set this variable, the printer that is set as the default printer is used.

RecordRef
Type: RecordRef
The RecordRef that refers to the table in which you want to find a record.

Remarks

You typically use this method together with the RunRequestPage Method method. The RunRequestPage method runs a report request page without actually running the report, but instead, returns the parameters that are set on the request page as a string. You can then call the Print method to get the parameter string and print the report.

For a simple example that illustrates how to use the Print method, see example in the RunRequestPage Method method topic.

See Also

Report Data Type
Getting Started with AL

Report.PrintOnlyIfDetail Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the current settings of the PrintOnlyIfDetail property.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[IsPrintOnlyIfDetail := ]  Report.PrintOnlyIfDetail([SetPrintOnlyIfDetail: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

SetPrintOnlyIfDetail
Type: Boolean
The new setting of PrintOnlyIfDetail property.

Return Value

IsPrintOnlyIfDetail Type: Boolean The current settings of the PrintOnlyIfDetail property.

Example 1

The following example is from the OnAfterGetRecord trigger of a report. If the PrintOnlyIfDetail property is true and if a GLEntryPage record exists, given the current filters, then the PageGroupNo is incremented.

```
var
    GLEntryPage: Record "G/L Entry";
    PageGroupNo: Integer;
begin
    if CurrReport.PrintONLYifDETAIL and GLEntryPage.Find('-') then
      PageGroupNo := PageGroupNo + 1;
end;
```

Example 2

The following example sets the value of the PrintOnlyIfDetail Property to true. It requires that you create a Report variable named Report111. The Subtype of the variable is report 111, Customer - Top 10 List.

```
IsPrintOnlyIfDetail := Report111.PrintONLYifDETAIL(true);
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Quit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Aborts the processing of a report or XmlPort.

Syntax

```
Report.Quit()
```

Parameters

Report Type: Report An instance of the Report data type.

Remarks

When you use the QUIT method, the report or XMLport is exited without committing any changes that were made to the database during the execution. The OnPostReport Trigger or OnPostXMLport Trigger trigger will not be called.

Example

The following example shows how to use the QUIT method to abort an execution without committing any changes that were made during the processing.

```
// Do some database processing.
CurrReport.QUIT;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.RDLCLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the RDLC layout that is used on a report and returns it as a data stream.The method has an instance call and a static call. The following code shows the syntax of the RDLC method. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

Syntax

```
[Ok := ]  Report.RDLCLayout(var InStream: InStream)
```

Parameters

Report Type: Report An instance of the Report data type.

InStream
Type: InStream
The variable in which to return the RDLC layout.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Using the return value is optional. When you use the return value, if the RDLC layout cannot be retrieved at run-time, then the system returns **false** and no error recorded. When you omit the return value, if the RDLC layout cannot be retrieved at run-time, then an error occurs, which states that the layout could not be retrieved.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the report that you specify.

Syntax

```
Report.Run()
```

Parameters

Report Type: Report An instance of the Report data type.

Remarks

After you define the *Report* variable, you can run this method or the RunModal Method (Report) on the variable. With the Run method, the variable is automatically cleared after the method is executed. With the RunModal method, the variable is not automatically cleared.

Use Run method or the RunModal method if you know at design time the exact report that you want to run. Otherwise, use the Report.Run Method or Report.RunModal Method.

If the report you specify does not exist, then a compile error occurs.

> **NOTE**
> Internet browsers can only handle one file per request. Therefore, with the Web client, if this method is called in a repetitive statement (or loop) that generates multiple files, only the last file will be sent to the browser. Alternatively, when designing for the Web client, bundle the files in an archive file (.zip), for example, by using the methods found in codeunit **419 File Management**. For more details about this design pattern, see Multi-File Download. Although this article is written for Dynamics NAV, it is still relevant for Business Central. The methods in codeunit 419 are not external, therefore cannot be used in extensions. Instead, when developing extensions in AL, use the external methods of codeunit **425 Data Compression**. The approach is similar.

Example

```
var
    CustomerRec: Record Customer;
    SomeReport: Report "Salesperson - Sales Statistics";
begin
    CustomerRec.SetCurrentKey("No.");
    CustomerRec.SetFilter("Salesperson Code", 'JR|PS');
    SomeReport.SetTableView(CustomerRec);
    SomeReport.Run;
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.RunModal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the report that you specify.

Syntax

```
Report.RunModal()
```

Parameters

Report Type: Report An instance of the Report data type.

Remarks

After you define the *Report* variable, you can run this method or the Run Method (Report) on the variable. As opposed to the Run method, with the RunModal method, the variable is not cleared after it executes this function. You must handle clearing the variable.

Use the RunModal method or the Run method if you know at design time the exact report you want to run. Otherwise, use the Report.RunModal Method or Report.Run Method.

If the report you specify does not exist, then a compile error occurs.

The request page is run modally when you use this method.

> **NOTE**
> Internet browsers can only handle one file per request. Therefore, with the Web client, if this method is called in a repetitive statement (or loop) that generates multiple files, only the last file will be sent to the browser. Alternatively, when designing for the Web client, bundle the files in an archive file (.zip), for example, by using the methods found in codeunit **419 File Management**. For more details about this design pattern, see Multi-File Download. Although this article is written for Dynamics NAV, it is still relevant for Business Central. The methods in codeunit 419 are not external, therefore cannot be used in extensions. Instead, when developing extensions in AL, use the external methods of codeunit **425 Data Compression**. The approach is similar.

Example

```
var
    CustomerRec: Record Customer;
    SomeReport: Report "Chart of Accounts";
begin
    …
    Clear(CustomerRec);
    CustomerRec.SetRecFilter;
    SomeReport.XXX; // Any user-defined method.
    SomeReport.SetTableView(CustomerRec);
    SomeReport.RunModal();
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.RunRequestPage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs the request page for a report without running the report. Returns an XML string that contains the request page parameters that are entered on the request page.

Syntax

```
ReportParameters :=   Report.RunRequestPage([PageParameters: String])
```

Parameters

Report Type: Report An instance of the Report data type.

PageParameters
Type: String
A string of request page parameters as XML to use to run the report.

Return Value

ReportParameters Type: String XML string that contains the request page parameters that are entered on the request page

Remarks

This method opens the request page for the specified report, where the user can provide parameters for the report. When the user closes the request page by choosing the **OK** button, a string that contains the parameter values that were set by the user is returned. When the user chooses the **Cancel** button, an empty string will be returned. The returned parameter string can be picked up by calling one of the following methods:

- Execute Method
- Print Method
- SaveAs Method

> **NOTE**
> You can use these methods to schedule reports in the job queue.

Because the request page runs in the context of where it was invoked from, users cannot bookmark a link to this page from the user interface.

Example

This example illustrates how to use the RunRequestPage method to run the request page for report ID 206 Sales Invoice. The request page parameters are saved to a table, and then uses the parameters with the Execute,

SaveAs, and Print methods to preview the report, save it as a PDF file, and print it.

This example requires that you create a table for holding parameters that are entered on the report request page and a codeunit that runs the report methods.

Create a table called **Request Parameters** that has the following fields.

```
var
    ReportId: Integer;
    UserId: Code[100];
    Parameters: BLOB;
```

Create a codeunit and add the following code to the *OnRun* trigger of the codeunit.

```
var
    ReportParameters: Record "Report Parameters";
    XmlParameters: Text;
    OStream: OutStream;
    IStream: InStream;
    CurrentUser: Code[100];
    Content: File;
    TempFileName: Text;
begin
    // Use the Report.RunRequestPage method to run the request page to get report parameters
    XmlParameters := Report.RunRequestPage(206);
    CurrentUser := UserId;

    // Save the request page parameters to the database table
    with ReportParameters do begin
        // Cleanup
        if Get(206,CurrentUser) then
        Delete;

        SetAutoCalcFields(Parameters);
        ReportId := 206;
        UserId := CurrentUser;
        Parameters.CreateOutStream(OStream,TextEncoding::UTF8);
        Message(XmlParameters);
        OStream.WriteText(XmlParameters);

        Insert;
    end;

    Clear(ReportParameters);
    XmlParameters := '';

    // Read the request page parameters from the database table
    with ReportParameters do begin
        SetAutoCalcFields(Parameters);
        Get(206,CurrentUser);
        Parameters.CreateInStream(IStream,TextEncoding::UTF8);
        IStream.ReadText(XmlParameters);
    end;

    // Use the Report.SaveAs method to save the report as a PDF file
    Content.Create('TestFile.pdf');  // only supported in Business Central on-premises
    Content.CreateOutStream(OStream);  // only supported in Business Central on-premises
    Report.SaveAs(206,XmlParameters,ReportFormat::Pdf,OStream);
    Content.Close;  // only supported in Business Central on-premises

    // Use the Report.Execute method to preview the report
    Report.Execute(206,XmlParameters);

    // Use the Report.Print method to print the report
    Report.Print(206,XmlParameters);
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAs Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Runs a specific report without a request page and saves the report as a PDF, Excel, Word, or XML file. Instead of using the request page to obtain parameters at runtime, the method gets the parameter values as an input parameter string, typically from the return value of a RunRequestPage method call.

Syntax

```
[Ok := ]  Report.SaveAs(Parameters: String, Format: ReportFormat, var OutStream: OutStream [, RecordRef:
RecordRef])
```

Parameters

Report Type: Report An instance of the Report data type.

Parameters
Type: String
A string of request page parameters as XML to use to run the report. The parameter string is retrieved from the return value a RunRequestPage method call.

Format
Type: ReportFormat
The type of file to save the report as. The following options are supported: Pdf, Excel, Word, and XML.

OutStream
Type: OutStream
The stream to which to write a report.

RecordRef
Type: RecordRef
The RecordRef that refers to the table in which you want to find a record.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You typically use this method together with the RunRequestPage Method method. The RunRequestPage method runs a report request page without actually running the report, but instead, returns the parameters that are set on the request page as a string. You can then call the SaveAs method to get the parameter string and save the report to a file of the specified format.

For a simple example that illustrates how to use the SaveAs method, see example in the RunRequestPage Method method topic.

NOTE

By default, when a report uses an RDLC report layout at runtime, fonts are embedded in the generated PDF. You can specify whether fonts are embedded in the PDF for RDLC reports by changing the **Report PDF Font Embedding** setting in the Dynamics 365 Business Central service instance configuration or changing the **PDFFontEmbedding** property in report objects.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsExcel Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report on the computer that is running the server as a Microsoft Excel (.xls) workbook.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsExcel(FileName: String)
```

Parameters

Report Type: Report An instance of the Report data type.

FileName
Type: String
The path and the name of the file that you want to save the report as. The path must exist, the file must not be being used, and the server process must have permission to write to the file. Otherwise, you will get errors.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the SaveAsExcel method on the global Report object and on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you do know which report you want to run, then create a Report variable, set the Subtype of the variable to a specific report, and use this variable when you call the SaveAsExcel method.

When you call the SaveAsExcel method, the report is generated and saved to "*FileName*." The request page is not shown.

If the destination folder that you specify in *FileName* does not exist, then you get the following error:

The specified path is invalid.

If the file that you specify in *FileName* is being used, then you get the following error:

An I/O exception occurred during the operation.

If the Dynamics 365 Business Central service process does not have permission to write to the file that you specify in *FileName*, then you get the following error:

Either the caller does not have the required permission or the specified path is read-only.

Example

This example shows how to use the SaveAsExcel method to save the Excel workbook to the Dynamics 365 Business Central service and then download the file to a computer that is running the Dynamics 365 application.

```
var
    TempFile: File;
    Name: Text[250];
    NewStream: InStream;
    ToFile: Text[250];
    ReturnValue: Boolean;
begin
    // Specify that TempFile is opened as a binary file.
    TempFile.TextMode(False);
    // Specify that you can write to TempFile.
    TempFile.WriteMode(True);
    Name := 'C:\Temp\TempReport.xls';
    // Create and open TempFile.
    TempFile.Create(Name);
    // Close TempFile so that the SaveAsExcel method can write to it.
    TempFile.Close;

    Report.SaveAsExcel(406,Name);

    TempFile.Open(Name);
    TempFile.CreateInStream(NewStream);
    ToFile := 'Report.xls';

    // Transfer the content from the temporary file on the
    // server to a file on the client.
    ReturnValue := DownloadFromStream(
      NewStream,
      'Save file to client',
      '',
      'Excel File *.xls| *.xls',
      ToFile);

    // Close the temporary file.
    TempFile.Close();
end;
```

You can create an action on a page and set the action to run this code. When you run the action, the **Export File** dialog box opens. Choose **Save** to save the file to the client.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsHtml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as an HTML file. The file is saved on the computer where the server instance is running, and then downloaded to the client when ready. > This method is only supported when a report uses a Word report layout when it is run.

NOTE

This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsHtml(FileName: String)
```

Parameters

Report Type: Report An instance of the Report data type.

FileName

Type: String

The folder path and name of the file that you want to save the report as. The path must already exist and the service (login) account that is used by the server instance must have permission to write to the target folder. Otherwise, you will get errors.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The SaveAsHTML method uses the logic in the codeunit **9651 Document Report Mgt.** code unit to handle the format transformation.

The SaveAsHTML method can be used on the Report data type and on Report variables. When you are programming the SaveAsHTML method, if you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you know which report you want to run, then create a Report variable, set the **Subtype** of the variable to a specific report, and then use this variable when you call the SaveAsHTML method.

When you call the SaveAsXML method, the report is generated and saved to "FileName." The request page is not shown.

Reports that use an RDLC layout when run cannot be saved in the HTML format. A runtime error will occur if SaveAsHTML is used on an RDLC report.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsPdf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as a .pdf file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsPdf(FileName: String)
```

Parameters

Report Type: Report An instance of the Report data type.

FileName
Type: String
The path and name of the file that you want to save the report as.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the **SaveAsPDF** method on the global Report object or on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you do know which report you want to run, then create a Report variable, set the Subtype of the variable to a specific report, and use this variable when you call the **SaveAsPDF** method.

When you call **SaveAsPDF**, the report is generated and saved to "*FileName*." A **Saving to PDF** window shows the status of the process. Note that the request page will not be shown.

The *FileName* parameter specifies a location on the computer that is running Dynamics 365 Business Central service. If you call this method from a client, such as from an action on a page, then use the DOWNLOAD Method (File) to download the .pdf file from the computer that is running Dynamics 365 Business Central service to the computer that is running the client.

NOTE

By default, when a report uses an RDLC report layout at runtime, fonts are embedded in the generated PDF. You can specify whether fonts are embedded in the PDF for RDLC reports by changing the **Report PDF Font Embedding** setting in the Dynamics 365 Business Central service instance configuration or changing the **PDFFontEmbedding** property in report objects.

Example

This example shows how to use the **SaveAsPDF** method to save a specific report as a PDF file on the computer that is running Dynamics 365 Business Central service.

```
var
    Filename: Text;
    ReturnValue: Boolean;
    Report206: Report "Sales - Invoice';
begin
    Filename := 'C:\MyReports\report206.pdf';
    ReturnValue := Report206.SaveAsPDF(Filename);
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsWord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report on the computer that is running the server as a Microsoft Word (.doc) document.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsWord(FileName: String)
```

Parameters

Report Type: Report An instance of the Report data type.

FileName
Type: String
The path and the name of the file that you want to save the report as. The path must exist, the file must not be being used, and the server process must have permission to write to the file. Otherwise, you will get errors.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the SaveAsWORD method on the global Report object or on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you do know which report you want to run, then create a Report variable, set the Subtype of the variable to a specific report, and use this variable when you call the SaveAsWORD method.

When you call the SaveAsWORD method, the report is generated and saved to "*FileName*." The request page is not shown.

If the destination folder that you specify in *FileName* does not exist, then you get the following error:

The specified path is invalid.

If the file that you specify in *FileName* is being used, then you get the following error:

An I/O exception occurred during the operation.

If the Dynamics 365 Business Central service process does not have permission to write to the file that you specify in *FileName*, then you get the following error:

Either the caller does not have the required permission or the specified path is read-only.

Example

This example shows how to use the SaveAsWORD method to save the Word document on the Dynamics 365 Business Central service, and then download the file to a different computer that is running the Dynamics 365 application.

```
var
    TempFile: File;
    Name: Text[250];
    NewStream: InStream;
    ToFile: Text[250];
    ReturnValue: Boolean;
begin
    // Specify that TempFile is opened as a binary file.
    TempFile.TextMode(False);
    // Specify that you can write to TempFile.
    TempFile.WriteMode(True);
    Name := 'C:\Temp\TempReport.doc';
    // Create and open TempFile.
    TempFile.Create(Name);
    // Close TempFile so that the SaveAsWORD method can write to it.
    TempFile.Close;

    Report.SaveAsWORD(406,Name);

    TempFile.Open(Name);
    TempFile.CreateInStream(NewStream);
    ToFile := 'Report.doc';

    // Transfer the content from the temporary file on the
    // server to a file on the client.
    ReturnValue := DownloadFromStream(
      NewStream,
      'Save file to client',
      '',
      'Word File *.doc| *.doc',
      ToFile);

    // Close the temporary file.
    TempFile.Close();
end;
```

You can create an action on a page and set the action to run this code. When you run the action, the **Export File** dialog box opens. Choose **Save** to save the file to the client.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the resulting data set of a query as an .xml file.The following code shows the syntax of the SaveAsXML method. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  Report.SaveAsXml(FileName: String)
```

Parameters

Report Type: Report An instance of the Report data type.

FileName
Type: String
The path and name of the file that you want to save the query to.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use the SaveAsXML method on the global Report object and on Report variables. If, at design time, you do not know the specific report that you want to run, then use the global Report object and specify the report number in the *Number* parameter. If you know which report you want to run, then create a Report variable, set the **Subtype** of the variable to a specific report, and then use this variable when you call the SaveAsXML method.

When you call the SaveAsXML method, the report is generated and saved to "*FileName*." The request page is not shown.

If the destination folder that you specify in *FileName* does not exist, then you get the following error:

The specified path is invalid.

If the file that you specify in *FileName* is being used, then you get the following error:

An I/O exception occurred during the operation.

If the Dynamics 365 Business Central service does not have permission to write to the file that you specify in

FileName, then you get the following error:

Either the caller does not have the required permission or the specified path is read-only.

To resolve this issue, verify that the service account that is running the Dynamics 365 Business Central service instance has write permissions on the path.

Example

This example shows how to use the SaveAsXML method to save a report as an .xml file on the Dynamics 365 Business Central service, and then download the file to a computer that is running the Dynamics 365.

```
var
    TempFile: File;
    Name: Text[250];
    NewStream: InStream;
    ToFile: Text[250];
    ReturnValue: Boolean;
begin
    // Specify that TempFile is opened as a binary file.
    TempFile.TextMode(False);
    // Specify that you can write to TempFile.
    TempFile.WriteMode(True);
    Name := 'C:\Temp\TempReport.xml';
    // Create and open TempFile.
    TempFile.Create(Name);
    // Close TempFile so that the SaveAsXML method can write to it.
    TempFile.Close;

    Report.SaveAsXML(406,Name);

    TempFile.Open(Name);
    TempFile.CreateInStream(NewStream);
    ToFile := 'Report.xml';

    // Transfer the content from the temporary file on
    // server to a file on the client.
    ReturnValue := DownloadFromStream(
      NewStream,
      'Save file to client',
      '',
      'Excel File *.xml| *.xml',
      ToFile);

    // Close the temporary file.
    TempFile.Close();
end;
```

You can create an action on a page and set the action to run this code. When you run the action, the **Export File** dialog box opens. Choose **Save** to save the file to the client.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.SetTableView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies the table view on the current record as the table view for the page, report, or XmlPort.

Syntax

```
Report.SetTableView(var Record: Record)
```

Parameters

Report Type: Report An instance of the Report data type.

Record
Type: Record
The record that has a table view that you want to apply to the page or data item.

Remarks

The table view is the view of the table that you present to the user. You determine what records that the user can see by setting filters, determining the sorting order, and selecting the keys.

This method only narrows the view of the table that was set through the SourceTableView Property of the page or through the DataItemTableView Property of the data item.

> **IMPORTANT**
> SETTableView is not supported for setting views on subpages from code on table headers. For example, you cannot set a table view on the SalesOrder subpage from the SalesHeader.

Example

This example is based on the Sales Header table and shows how SETTableView is used for a page object.

```
var
    SalesHeader: Record "Sales Header";
    SomePage: Page "Sales List";
begin
    SalesHeader.SETCURRENTKey("Document Type");
    SalesHeader.SETRANGE("Document Type",SalesHeader."Document Type"::Order);
    SomePage.SETTableView(SalesHeader); // Only view sales orders.
    SomePage.Run;
end;
```

The page that is reference by the SomePage variable can be any page object that has Sales Header as the value of the SourceTable Property.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.ShowOutput Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Returns the current setting of whether a section should be printed, and changes this setting.

Syntax

```
Show :=   Report.ShowOutput()
```

Parameters

Report Type: Report An instance of the Report data type.

Return Value

Show Type: Boolean **true** if the section is printed, otherwise **false**.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.ShowOutput Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.3 until version 2.3 where it was deprecated.*

Returns the current setting of whether a section should be printed, and changes this setting.

Syntax

```
[Show := ]  Report.ShowOutput(Value: Boolean)
```

NOTE

This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

Value

Type: Boolean

true if the section is printed, otherwise **false**.

Return Value

Show Type: Boolean **true** if the section is printed, otherwise **false**.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.Skip Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Skips the current iteration of the current report or XmlPort.

Syntax

```
Report.Skip()
```

Parameters

Report Type: Report An instance of the Report data type.

Remarks

The SKIP Method (Report, XMLport) allows you to conditionally skip processing of the current report or XMLport. If the processing occurs inside a loop, the processing continues with the next record after the SKIP Method (Report, XMLport) is called.

A typical situation in which you will use SKIP is when you want to retrieve records from a related table by using values in the current record for forming a filter. If the values in the current record already indicate that no records from the related table will be retrieved, there is no need to perform this processing and you can use SKIP to avoid the processing.

Example

The following example shows how to use the SKIP Method (Report, XMLport) to skip processing the current record if the balance field of the record is zero. Processing of records will continue until a record that has a balance of 0 is encountered.

```
var
    Balance: Decimal;
begin
    if Balance = 0 then
      CurrReport.SKIP
    else
      ... // Do some processing.
end;
```

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.TotalsCausedBy Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Determines which field caused a group total to be calculated. This determines which field changed contents and thereby concluded a group.

Syntax

```
FieldNo :=   Report.TotalsCausedBy()
```

Parameters

Report Type: Report An instance of the Report data type.

Return Value

FieldNo Type: Integer The number of the field that caused the group to end and a group total to be calculated.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.UseRequestPage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets whether a request page is presented to the user.

Syntax

```
[IsUseRequestPage := ]  Report.UseRequestPage([SetUseRequestPage: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Report Type: Report An instance of the Report data type.

SetUseRequestPage

Type: Boolean

true if a request page will be presented to the user, otherwise **false**.

Return Value

IsUseRequestPage Type: Boolean **true** if a request page is presented to the user, otherwise **false**.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.WordLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the Word report layout that is used on a report and returns it as a data stream. The method has an instance call and a static call. The following code shows the syntax of the WORDLAYOUT method. The first line of code is the syntax for an instance method call. The second line of code is the syntax for a static method call.

Syntax

```
[Ok := ]  Report.WordLayout(var InStream: InStream)
```

Parameters

Report Type: Report An instance of the Report data type.

InStream
Type: InStream
The variable in which to return the Word report layout.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Using the return value is optional. When you use the return value, if the Word report layout cannot be retrieved at run-time, then the system returns **false** and no error recorded. When you omit the return value, if the Word report layout cannot be retrieved at run-time, then an error occurs, which states that the Word report could not be retrieved.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

Report.WordXmlPart Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the report data structure as structured XML that is compatible with Microsoft Word custom XML parts.

Syntax

```
String :=   Report.WordXmlPart([ExtendedFormat: Boolean])
```

Parameters

Report Type: Report An instance of the Report data type.

ExtendedFormat
Type: Boolean
If you set this variable to true, then XML elements will include the following attributes:

- ElementType="Parameter|Column|DataItem". Specifies the element type as defined for the report in Report Designer. Parameter is typically used for elements, such as captions.
- ElementId="ID". Specifies the ID that is assigned to the element by its ID Property.
- DataType="Type". Specifies the data type of the element. If you omit this parameter or set it to false, then the element attributes are not included in the XML.

Return Value

String Type: String A string representation of the report data structure as structured XML that is compatible with Microsoft Word custom XML parts.

Remarks

This method returns the data structure of the report as structured XML that complies with custom XML parts in Microsoft Word 2013. The following table provides a simplified overview of the XML that is returned by the method.

XML	DESCRIPTION
`<?xml version="1.0" encoding="utf-16"?>`	Header
`<NavWordReportXmlPart xmlns="urn:microsoft-../report/<reportname>/<id>/"`	XML namespace specification. `<reportname>` is the name assigned to the report object. `<id>` is the ID that is assigned to the report.

XML	DESCRIPTION

`..<Labels>`
 	
`....`
`<ColumnNameCaption>ColumnNameCaption</ColumnNameCaption>`
 	
`....<LabelName>LabelCaption</LabelName>`
 	
`..</Labels>`
	Contains all the labels for the report. Labels are listed in alphabetical. The element includes labels that are related to columns that have the IncludeCaption Property set to **Yes** and labels that are defined in Report Label Designer.
	
	- Label elements that are related to columns have the format `<ColumnNameCaption>ColumnNameCaption</ColumnNameCaption>`, where `ColumnName` is determined by the column's Name Property.
	- Label elements from Report Label Designer have the format `<LabelName>LabelCaption</LableName`, where `LabelName` is determined by the label's Name Property and `LabelCaption` is determined by the label's Caption Property.

`..<DataItem1>`
 	
`....`
`<DataItem1Column1>DataItem1Column1</DataItem1Column1>`
	Top-level data item and columns. Columns are listed in alphabetical order.
	
	The element names and values are determined by the Name property of the data item or column.

`....<DataItem2>`
 	
`......`
`<DataItem2Column1>DataItem2Column1</DataItem2Column1>`
 	
`....</DataItem2>`
 	
`....<DataItem3>`
 	
`......`
`<DataItem3Column1>DataItem3Column1</DataItem3Column1>`
 	
`....</DataItem3>`
	Data items and columns that are nested in the top-level data item. Columns are listed in alphabetical order under the respective data item.

`..</DataItem1>`
 	
`</NavWordReportXmlPart>`
	Closing elements.

Word custom XML parts enable you to integrate business data into Word documents. For example, the WORDXMLPART method is used internally by Dynamics 365 when you are creating report layouts in Word. You can use this method to create a custom XML part, and then, together with the SaveAsXML Method (Reports) method and additional data merging tools, you can implement your own functionality for mapping and laying out report data in Word documents. To create a custom XML part, you can save the return value to an .xml file that is encoded in UTF-16 (16-bit Unicode Transformation Format). The resultant file can be added to Word documents as a custom XML part to map the report data set as XML data.

Example

The following example uses the WORDXMLPART method to save the data structure of Report 112 Sales Statistics in an XML file in a predefined folder *C:\Report Documents*. The resultant file can be used in Word as a custom XML part.

```
var
    SalesStatsReport: File;
    ReportAsString: Text;
begin
    ReportAsString := Report.WORDXMLPART(112);
    SalesStatsReport.TextMode(True);
    SalesStatsReport.WriteMode(True);
    SalesStatsReport.Create('C:\Report Documents\SalesStatsReport.xml', TextEncoding::UTF16);
    SalesStatsReport.WRITE(ReportAsString);
    SalesStatsReport.Close;
end;
```

The code generates the report structure as XML, and then writes the XML to the file *C:\Report Documents\SalesStatsReport.xml*.

See Also

Report Data Type
Getting Started with AL
Developing Extensions

RequestPage Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a page that is run before the report starts to execute. Request pages enable end-users to specify options and filters for a report.

The following methods are available on instances of the RequestPage data type.

METHOD NAME	DESCRIPTION
Activate([Boolean])	Activates the current page on the client if possible. The data on the page will not be refreshed.
Caption([String])	Shows the caption in the title bar. For example, the default value in English (United States) is the same as the name of the page.
Close()	Closes the current page.
Editable([Boolean])	Gets or sets the default editability of the page.
LookupMode([Boolean])	Gets or sets the default lookup mode for the page.
ObjectId([Boolean])	Returns a string in the "Page xxx" format, where xxx is the caption or ID of the application object.
SaveRecord()	Saves the current record as if performed by the client. If the record does not exist, it is inserted, otherwise it is modified.
SetSelectionFilter(var Record)	
Update([Boolean])	Saves the current record and then updates the controls on the page. If you set the SaveRecord parameter to false, this method will not save the record before the page is updated.

See Also

Getting Started with AL
Developing Extensions

RequestPage.Activate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Activates the current page on the client if possible. The data on the page will not be refreshed.

Syntax

```
[Ok := ]  RequestPage.Activate([Refresh: Boolean])
```

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

Refresh

Type: Boolean

If set to **true**, the data on the page will be refreshed.

Return Value

Ok Type: Boolean

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Shows the caption in the title bar. For example, the default value in English (United States) is the same as the name of the page.

Syntax

```
[Caption := ]  RequestPage.Caption([NewCaption: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

NewCaption
Type: String

Return Value

Caption Type: String

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes the current page.

Syntax

```
RequestPage.Close()
```

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.Editable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the default editability of the page.

Syntax

```
[Editable := ]  RequestPage.Editable([NewEditable: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

NewEditable

Type: Boolean

Return Value

Editable Type: Boolean

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.LookupMode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the default lookup mode for the page.

Syntax

```
[LookupMode := ]  RequestPage.LookupMode([NewLookupMode: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

NewLookupMode
Type: Boolean

Return Value

LookupMode Type: Boolean

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.ObjectId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a string in the "Page xxx" format, where xxx is the caption or ID of the application object.

Syntax

```
String :=   RequestPage.ObjectId([UseNames: Boolean])
```

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

UseNames
Type: Boolean

Return Value

String Type: String

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.SaveRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the current record as if performed by the client. If the record does not exist, it is inserted, otherwise it is modified.

Syntax

```
RequestPage.SaveRecord()
```

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.SetSelectionFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Syntax

```
RequestPage.SetSelectionFilter(var Record: Record)
```

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

Record
Type: Record

Remarks

If all records are selected, marks will not be used.

If only the current record is selected on the page, then SetSelectionFilter does the following:

- Sets the current filter group to 0 on the destination record
- Adds filters on the primary key fields that point to the current record of the page

If more than one record is selected on the page, then SetSelectionFilter does the following:

- Copies the current key from the page source table to the destination record
- Copies the current sort order from the table to the destination record
- Copies the current filters that are set in all filter groups
- Copies the current filter group
- Marks the selected records and sets the "marked only" filter

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

RequestPage.Update Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves the current record and then updates the controls on the page. If you set the SaveRecord parameter to false, this method will not save the record before the page is updated.

Syntax

```
RequestPage.Update([SaveRecord: Boolean])
```

Parameters

RequestPage Type: RequestPage An instance of the RequestPage data type.

SaveRecord

Type: Boolean

Indicates if the current record is saved. To save the current record, set the value to true. If the value is set to false, the page is updated without saving the record.

See Also

RequestPage Data Type
Getting Started with AL
Developing Extensions

Session Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a Microsoft Dynamics Business Central session.

The following methods are available on the Session data type.

METHOD NAME	DESCRIPTION
ApplicationArea([String])	Gets or sets the application areas for the current session.
ApplicationIdentifier()	Gets the application ID associated with the current thread.
BindSubscription(Codeunit)	Binds the event subscriber methods in the codeunit to the current codeunit instance for handling the events that they subscribe to. This essentially activates the subscriber functions for the codeunit instance.
CurrentClientType()	Gets the client type that is running in current session.
CurrentExecutionMode()	Specifies the mode in which the session is running.
DefaultClientType()	Gets the default client that is configured for the server instance that is used by the current session.
EnableVerboseTelemetry(Boolean, Duration)	Temporarily enable verbose telemetry on the current session.
GetCurrentModuleExecutionContext()	Gets the current session's execution context for the currently executing module.
GetExecutionContext()	Gets the current session's execution context.
GetModuleExecutionContext([Guid])	Gets the current session's execution context scoped to a specific module.
IsSessionActive(Integer)	Tests if the specified SessionID is active on the server instance where it was started.
LogMessage(String, String, Verbosity, DataClassification, TelemetryScope, String, String [, String] [, String])	Logs a trace message to a telemetry account.
LogMessage(String, String, Verbosity, DataClassification, TelemetryScope, Dictionary of [Text, Text])	Logs a trace message to a telemetry account.
SendTraceTag(String, String, Verbosity, String [, DataClassification])	Send a trace tag to the telemetry service.
SetDocumentServiceToken(String)	Sets the document service token in the current session.

METHOD NAME	DESCRIPTION
StartSession(var Integer, Integer [, String] [, var Record])	Starts a session without a UI and runs the specified codeunit.
StopSession(Integer [, String])	Stops a session.
UnbindSubscription(Codeunit)	Unbinds the event subscriber methods from in the codeunit instance. This essentially deactivates the subscriber methods for the codeunit instance.

See Also

Getting Started with AL
Developing Extensions

Session.ApplicationArea Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the application areas for the current session.

Syntax

```
[ApplicationArea := ]  Session.ApplicationArea([ApplicationArea: String])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

ApplicationArea
Type: String
The new application areas for the current session.

Return Value

ApplicationArea Type: String The application areas for the current session.

Remarks

This method lets you hide certain user interface elements (including page fields and actions, and report request page options) based on the application area to which they belong. Controls that define these items can be tagged with one or more application areas by setting the ApplicationArea property. When the ApplicationArea method is called in a client session, only those controls that are tagged with the application areas set by the method will be appear in the user interface.

> **NOTE**
> Currently, this functionality is intended for the application areas that are defined in **table 9178 Application Area Setup**. You can define your own application areas but be aware that the implementation might change in future release.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.ApplicationIdentifier Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the application ID associated with the current thread.

Syntax

```
AppId :=   Session.ApplicationIdentifier()
```

NOTE

This method can be invoked without specifying the data type name.

Return Value

AppId Type: String The application ID associated with the current thread.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.BindSubscription Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Binds the event subscriber methods in the codeunit to the current codeunit instance for handling the events that they subscribe to. This essentially activates the subscriber functions for the codeunit instance.

Syntax

```
[Ok := ]  Session.BindSubscription(Codeunit: Codeunit)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Codeunit
Type: Codeunit
The codeunit that contains the event subscribers.

Return Value

Ok Type: Boolean **true** if the event subscriber methods bind successfully to the codeunit instance and no errors occurred, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can only call this method on codeunits that have the EventSubscriberInstance Property set to **Manual**.

The codeunit instance that event subscribers are bound to will be this exact instance. Events will be raised on this instance. You can't bind the same instance more than once, but you can bind multiple instances of the same codeunit. This condition will result in an event subscriber call on each bound instance when a given event is raised.

Example

The following sample code illustrates a typical use of the BindSubscription method.

```
Method MyFunction(….)
LocalVar
  SubScriberCodeunit5000;
begin
  // Set global information on the subscriber codeunit if required
  // You can rely on the instance being the same as the one receiving the event subscriber call

  SubScriberCodeunit5000.MySetGlobalInfo(<info you can later test in the subscriber event method>)
  BindSubscription(SubscriberCodeunit5000);
  DoSomething(…);  // After binding, all subscriptions on SubscriberCodeunit5000 are "active".

end; // Notice, that when SubScriberCodeunit5000 goes out of scope, all bindings are removed.
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.CurrentClientType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the client type that is running in current session.

Syntax

```
ClientType :=   Session.CurrentClientType()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

ClientType Type: ClientType The client type that is running in current session.

Remarks

You can use CurrentClientType as a parameter in the GetURL Method to get the URL of the current client.

Example 1

In the following example, CurrentClientType is used to get the client type for the session and return a message if the session uses the Tablet client.

```
if CurrentClientType = ClientType::Tablet then
  Message('The session is running the Tablet client');
```

Example 2

In the following example, CurrentClientType is used as a parameter of the GetURL Method to return the URL of the client that invokes the code.

```
url := GetURL(CurrentClientType);
Message('The URL is %1.', url);
```

See Also

Session Data Type

Session.CurrentExecutionMode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies the mode in which the session is running.

Syntax

```
ExecutionMode :=   Session.CurrentExecutionMode()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

ExecutionMode Type: ExecutionMode The mode in which the session is running.

Example

This example requires that you create a variable named Mode that has a DataType of ExecutionMode.

```
Mode := CurrentExecutionMode;
Message('Current execution mode is %1.', Mode);
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.DefaultClientType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the default client that is configured for the server instance that is used by the current session.

Syntax

```
ClientType :=   Session.DefaultClientType()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

ClientType Type: ClientType The default client that is configured for the server instance that is used by the current session.

Remarks

You can use DefaultClientType in a GetURL Method call to get the URL of the default client.

Example 1

In the following example, DefaultClientType is used to return the default client type that is configured for the Dynamics 365 Business Central service instance that is used by the current session.

```
if DefaultClientType = ClientType::Web then
  Message('The default client is Web client');
```

Example 2

In the following example, DefaultClientType is used as a parameter in the GetURL method to return the URL of the default client that is configured for the Dynamics 365 Business Central service instance.

```
url := GetURL(DefaultClientType);
Message('The URL is %1.', url);
```

See Also

Session.EnableVerboseTelemetry Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.2.*

Temporarily enable verbose telemetry on the current session.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Session.EnableVerboseTelemetry(EnableFullALFunctionTracing: Boolean, Duration: Duration)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

EnableFullALFunctionTracing
Type: Boolean
Specifies whether to enable method tracing.
Duration
Type: Duration
Specifies the amount of time, in milliseconds, that verbose telemetry is enabled on the session. When the time is exceeded, system specified telemetry level is used again. The maximum value is 3600000, one hour.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.GetCurrentModuleExecutionContext Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current session's execution context for the currently executing module.

Syntax

```
ExecutionContext :=   Session.GetCurrentModuleExecutionContext()
```

NOTE

This method can be invoked without specifying the data type name.

Return Value

ExecutionContext Type: ExecutionContext The current session's execution context for the currently executing module.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.GetExecutionContext Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current session's execution context.

Syntax

```
ExecutionContext :=   Session.GetExecutionContext()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

ExecutionContext Type: ExecutionContext The current session's execution context.

Remarks

For an example of when and how to run the `GetExecutionContext` method, see Protecting sensitive code from running during upgrade.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.GetModuleExecutionContext Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current session's execution context scoped to a specific module.

Syntax

```
ExecutionContext :=   Session.GetModuleExecutionContext([AppId: Guid])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

AppId
Type: Guid
The application ID.

Return Value

ExecutionContext Type: ExecutionContext The current session's execution context scoped to a specific module.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.IsSessionActive Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests if the specified SessionID is active on the server instance where it was started.

Syntax

```
Ok :=   Session.IsSessionActive(SessionID: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

SessionID
Type: Integer
The ID of the session that you want to test if it is still active.

Return Value

Ok Type: Boolean **true** if the specified SessionID is active on the server instance where it was started, otherwise **false**.

Remarks

Use this method to test if a session has completed or is still active, for example if you want to check that a session started with StartSession is still running.

> **NOTE**
> The method looks for sessions on the local machine.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.LogMessage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.4.*

Logs a trace message to a telemetry account.

Syntax

```
Session.LogMessage(EventId: String, Message: String, Verbosity: Verbosity, DataClassification:
DataClassification, TelemetryScope: TelemetryScope, Dimension1: String, Value1: String [, Dimension2:
String] [, Value2: String])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

EventId

Type: String

The event ID of trace message.

Message

Type: String

The message logged into telemetry.

Verbosity

Type: Verbosity

The verbosity of the log.

DataClassification

Type: DataClassification

Classification of data in message.

TelemetryScope

Type: TelemetryScope

Specifies the scope of this trace message:

- ExtensionPublisher: Will emit this trace message to the Extension Publisher's telemetry account.
- All: Will emit this trace message additionally to the Partner's telemetry account.

  Dimension1

  Type: String

  Additional dimension that will be emitted to the telemetry account and that can be used to specify filters in the query.

  Value1

  Type: String

  The value of Dimension1.

  Dimension2

  Type: String

  Additional dimension that will be emitted to the telemetry account and that can be used to specify filters in the query.

Value2

Type: String

The value of Dimension2.

Remarks

For more information about using this method, see Creating Custom Telemetry Events for Application Insights.

Example

```
trigger OnRun();
begin
    LogMessage('MyExt-0001', 'This is a critical error message', Verbosity::Critical,
DataClassification::CustomerContent, TelemetryScope::ExtensionPublisher, 'result', 'failed', 'reason',
'critical error in code');
end;
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.LogMessage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 5.4.*

Logs a trace message to a telemetry account.

Syntax

```
Session.LogMessage(EventId: String, Message: String, Verbosity: Verbosity, DataClassification:
DataClassification, TelemetryScope: TelemetryScope, CustomDimensions: Dictionary of [Text, Text])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

EventId
Type: String
The event ID of trace message.
Message
Type: String
The message logged into telemetry.
Verbosity
Type: Verbosity
The verbosity of the log.
DataClassification
Type: DataClassification
Classification of data in message.
TelemetryScope
Type: TelemetryScope
Specifies the scope of this trace message:

- ExtensionPublisher: Will emit this trace message to the Extension Publisher's telemetry account.
- All: Will emit this trace message additionally to the Partner's telemetry account.

CustomDimensions
Type: Dictionary of [Text, Text]
Set of additional dimensions, specified as a dictionary, that will be emitted to the telemetry account and that can be used to specify filters in the query.

Remarks

For more information about using this method, see Creating Custom Telemetry Events for Application Insights.

Example

```
trigger OnRun();
var
    CustDimension: Dictionary of [Text, Text];
begin
    CustDimension.Add('result', 'failed');
    CustDimension.Add('reason', 'critical error in code');
    LogMessage('MyExt-0001', 'This is a critical error message', Verbosity::Normal,
DataClassification::OrganizationIdentifiableInformation, TelemetryScope::ExtensionPublisher, CustDimension);
end;
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.SendTraceTag Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 6.0 where it was deprecated for the following reason: "Use LogMessage instead."*

Send a trace tag to the telemetry service.

NOTE

This method is supported only in Business Central on-premises.

Syntax

```
Session.SendTraceTag(Tag: String, Category: String, Verbosity: Verbosity, Message: String [,
DataClassification: DataClassification])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Tag

Type: String

The tag.

Category

Type: String

The category.

Verbosity

Type: Verbosity

The verbosity.

Message

Type: String

The message.

DataClassification

Type: DataClassification

Classification of data in message.

Remarks

You use the SendTraceTag method for instrumenting the application for telemetry. When the SendTraceTag method called, a telemetry trace event is emitted. The event can then be recorded in the Windows event log or collected by other event trace collection tools, like PerfView, Logman, and Performance Monitor.

A telemetry event is given one of the following event IDs, depending on the `DataClassification` and `Verbosity`:

DATACLASSIFICATION	VERBOSITY	ID
All except `CustomerContent` and `EndUserIdentifiableInformation`	Critical	700
	Error	701
	Informational	702
	Verbose	704
	Warning	705
`CustomerContent` or `EndUserIdentifiableInformation`	Critical	707
	Error	708
	Informational	709
	Verbose	711
	Warning	712

NOTE

The SendTraceTag method is marked as obsolete in Business Central 2020 release wave 2 (v17). You can still use it, but we recommend that you send traces to Application Insights using the LOGMessage method instead. For more information, see Creating Custom Telemetry Traces for Application Insights Monitoring.

Example

The following code defines simple telemetry events for the five different severity levels.

```
SendTraceTag('Cronus-0001', 'Action', Verbosity::Critical, 'This is a critical message.',
DataClassification::CustomerContent);
SendTraceTag('Cronus-0002', 'Action', Verbosity::Error, 'This is an error message.',
DataClassification::EndUserIdentifiableInformation);
SendTraceTag('Cronus-0003', 'Action', Verbosity::Warning, 'This is a warning message.',
DataClassification::AccountData);
SendTraceTag('Cronus-0004', 'Action', Verbosity::Normal, 'This is an informational message.',
DataClassification::OrganizationIdentifiableInformation);
SendTraceTag('Cronus-0005', 'Action', Verbosity::Verbose, 'This is a verbose message.',
DataClassification::SystemMetadata);
```

The events emitted by this code will have the events IDs (listed in the order that the are called): 707, 708, 705, 702, and 704.

See Also

Session Data Type
Getting Started with AL

Session.SetDocumentServiceToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Sets the document service token in the current session.

Syntax

```
Session.SetDocumentServiceToken(Token: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Token

Type: String

The access token.

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.StartSession Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Starts a session without a UI and runs the specified codeunit.

Syntax

```
[Ok := ]  Session.StartSession(var SessionId: Integer, CodeunitId: Integer [, Company: String] [, var
Record: Record])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

SessionId

Type: Integer

The ID of the new session that is started. The ID is assigned to the SessionID variable after the session is started successfully. This parameter is passed by reference to the method.

CodeunitId

Type: Integer

The ID of the codeunit to run when the session is started.

Company

Type: String

The company in which to start the session. By default, the session is started in the same company as the calling session.

Record

Type: Record

A record that is passed to the OnRun trigger of the codeunit that runs when the session is started. This parameter is optional.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The session is started on the same Dynamics 365 Business Central service instance from which the method is called. The session that is started is a background session and therefore has no UI. The session executes using the same user credentials as the calling AL code.

The following table describes how dialog boxes are handled in a background session, which has no UI.

METHOD THAT CREATES THE DIALOG BOX	BEHAVIOR
Confirm Method (Dialog)	- Dialog box is suppressed. - The following error occurs on the Dynamics 365 Business Central service instance: **Dynamics 365 Business Central service attempted to issue a client callback to show a confirmation dialog box.**
Error Method (Dialog)	- Dialog box is suppressed. - AL code execution ends. - The error is logged to the event log of the Dynamics 365 Business Central service instance. - The error is added to the **Comments** field of the **Session Event** table.
Message Method (Dialog)	- Dialog box is suppressed. - The message is recorded in the event log of the computer that is running Dynamics 365 Business Central service instance. The log entry has type Information and includes the context of the message.
Open Method (Dialog)	- Dialog box is suppressed. - Dialog box text is not displayed or logged.

Each background session has the same impact on resources as a regular user session. In addition, it takes time and resources to start each background session. Therefore, we recommend that you consider when and how you use background sessions. For example, do not use background sessions for small tasks that occur often because the cost of starting the session for each tasks is high.

Example

In this example, the Cache Stress Test codeunit is a custom codeunit.

```
var
    CacheStressTestRec: Record Customer;
    SessionID: Integer;
    OK: Boolean;
begin
    OK := StartSession(SessionId, CodeUnit::"Cache Stress Test", CompanyName, CacheStressTestRec);
    if OK then
      StopSession(SessionId, 'Logoff cache stress test session')
    else
      Error('The session was not started successfully.');
end;
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.StopSession Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stops a session.

Syntax

```
[Ok := ]  Session.StopSession(SessionId: Integer [, Comment: String])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

SessionId

Type: Integer

The ID of the session that you want to stop.The session can be any of the following:

- Windows client session
- Web client session
- NAS services session
- SOAP web services client session
- OData web services client session
- Background session

Comment

Type: String

An optional comment about the session event. The comment is stored in Table 2000000111, the Session Event table.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The session that you want to stop and the session that calls StopSession must be running on the same instance of Dynamics 365 Business Central service. The session is stopped before the next AL statement executes. Open transactions are rolled back.

NOTE

If the current executing statement is the **Sleep** method, then the session is stopped immediately.

When a session is executing AL that does not interact with the server connection or the access lock used by the connection, StopSession cannot terminate the connection. StopSession can terminate connections that are inactive, idle, or using the database but not blocked.

You cannot stop the current, active session in which you are executing the **StopSession** call.

Example

This example assumes that you have a table named CacheStressTest that you use for testing.

```
var
    CacheStressTestRec: Record Customer;
    SessionID: Integer;
begin
    StartSession(SessionId, CodeUnit::"Cache Stress Test", CompanyName, CacheStressTestRec);
    StopSession(SessionId, 'Logoff cache stress test session');
end;
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

Session.UnbindSubscription Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Unbinds the event subscriber methods from in the codeunit instance. This essentially deactivates the subscriber methods for the codeunit instance.

Syntax

```
[Ok := ]  Session.UnbindSubscription(Codeunit: Codeunit)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Codeunit
Type: Codeunit
The codeunit that contains the event subscribers.

Return Value

Ok Type: Boolean **true** if the event subscriber methods unbind successfully to the codeunit instance and no errors occurred, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can only call this method on codeunits that have the EventSubscriberInstance Property set to **Manual**.

Calling this method on a codeunit that hasn't been bound (by the BindSubscription Method) will result in an error. If the call to this method is successful, all bindings are removed.

The codeunit instance that is unbound will be the same instance that previously was bound.

Example

The following sample code illustrates a typical use of the BindSubscription method.

```
Method MyFunction(….)
LocalVar
  SubScriberCodeunit5000;
begin
  // Set global information on the subscriber codeunit if required
  // You can rely on the instance being the same as the one receiving the event subscriber call

  SubScriberCodeunit5000.MySetGlobalInfo(<info you can later test in the subscriber event method>)
  BindSubscription(SubscriberCodeunit5000);
  DoSomething(…);  // After binding, all subscriptions on SubscriberCodeunit5000 are "active".
  UNBindSubscription(SubscriberCodeunit888);  // Now deactivating again
  DoStuff(…);  // This time no events are raised inside SubscriberCodeunit888;

end;
```

See Also

Session Data Type
Getting Started with AL
Developing Extensions

SessionSettings Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a complex data type for passing user personalization settings for a client session as an object. The object contains properties that correspond to the fields in the system table **2000000073 User Personalization**, including: App ID, Company, Language ID, Locale ID, Profile ID, Scope, and Time Zone. You can use the AL methods of the SessionSettings data type to get, set, and send the user personalization settings for the current client session.

The following methods are available on instances of the SessionSettings data type.

METHOD NAME	DESCRIPTION
Company([String])	Gets or sets the company property in a SessionSettings object.
Init()	Populates the instance of a SessionsSettings with the current client user's personalization properties (such as Profile ID and Company) that are stored in the database.
LanguageId([Integer])	Gets or sets the language ID property in a SessionSettings object.
LocaleId([Integer])	Gets or sets the locale ID property in a SessionSettings object.
ProfileAppId([Guid])	Gets or sets the ID of an extension, which provides a profile, in a SessionSettings object.
ProfileId([String])	Gets or sets the profile ID property in a SessionSettings object.
ProfileSystemScope([Boolean])	Gets or sets the profile scope property in a SessionSettings object.
RequestSessionUpdate(Boolean)	Passes a SessionSettings object to the client to request a new session that uses the user personalization properties that are set in the object. The current client session is abandoned and a new session is started.
TimeZone([String])	Gets or sets the time zone property in a SessionSettings object.

See Also

Getting Started with AL
Developing Extensions

SessionSettings.Company Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the company property in a SessionSettings object.

Syntax

```
[Company := ]  SessionSettings.Company([NewCompanyName: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewCompanyName
Type: String
Specifies the name of the company in the SessionSettings object. The company must already exist in the database, otherwise you will get an error at runtime.

Return Value

Company Type: String The name of the company that is set in the SessionSettings object.

Remarks

The company property in the SessionSettings object corresponds to the **Company** field in the in the system table **2000000073 User Personalization**.

Example

This example creates a SessionSettings object that is populated with the current client user's personalization data, and then calls the Company method to change the company to 'MyCompany'. Finally, the RequestSessionUpdate method sends a request to the client to abandon the current session and start a new session that uses the new company. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init;
  MySessionSettings.Company('MyCompany');
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.Init Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Populates the instance of a SessionsSettings with the current client user's personalization properties (such as Profile ID and Company) that are stored in the database.

Syntax

```
SessionSettings.Init()
```

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

Remarks

The method gets the data from the following fields in system table **2000000073 User Personalization**: App ID, Company, Language ID, Locale ID, Profile ID, Scope, and Time Zone. In the SessionSettings object, the data is stored in properties that correspond to the fields of the system table.

After you call the Init method, you can change the values in the object by calling the following methods:

- Company
- LanguageId
- LocalId method
- ProfileAppId
- ProfileId
- ProfileSystemScope
- TimeZone

The Init method is useful before calling the RequestSessionUpdate method to ensure that all properties are initialized before sending the request to the server instance to start a new client session.

Example

This example uses the Init method to create a SessionSettings object that includes the current client user's personalization settings from the database, and uses the Company method to set the company to 'MyCompany'. Then, the RequestSessionUpdate method sends a request to the client to abandon the current client session and start a new session that uses the personalization settings in the SessionSettings object.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init
  MySessionSettings.Company('MyCompany');
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.LanguageId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the language ID property in a SessionSettings object.

Syntax

```
[LanguageId := ]  SessionSettings.LanguageId([NewLanguageId: Integer])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewLanguageId
Type: Integer
Specifies the language ID to set in the SessionSettings object. The value must be a valid Windows language ID, which is typically a 4-digit value such as 1033 for English or 1030 for Danish. The default value is 1033.

Return Value

LanguageId Type: Integer The language ID that is set in the SessionSettings object.

Remarks

The langauge ID in the SessionSettings object corresponds to the **Language ID** field in the system table **2000000073 User Personalization**.

Example

This example creates a SessionSettings object that is populated with the current client user's personalization data, and then uses the LanguageId method to change the language ID to '1030'. Finally, the RequestSessionUpdate method sends a request to the client to abandon the current client session and start a new session that uses the new language. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init;
  MySessionSettings.LanguageId(1030);
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.LocaleId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the locale ID property in a SessionSettings object.

Syntax

```
[LocaleId := ]  SessionSettings.LocaleId([NewLocaleId: Integer])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewLocaleId
Type: Integer
Specifies the locale ID to set in the SessionSettings object. The value must be a valid Windows locale ID.

Return Value

LocaleId Type: Integer The locale ID that is set in the SessionSettings object.

Remarks

The locale ID in the SessionSettings object corresponds to the **Locale ID** field in the system table **2000000073 User Personalization** for the client session user.

Example

This example creates a SessionSettings object that is populated with the current client user's personalization data, and then uses the LocaleId method to set the locale ID to '1033'. Finally, the RequestSessionUpdate method sends a request to the client to abandon the current client session and start a new session that uses the new locale ID. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init;
  MySessionSettings.LocaleId(1033);
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.ProfileAppId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the ID of an extension, which provides a profile, in a SessionSettings object.

Syntax

```
[ProfileAppId := ]  SessionSettings.ProfileAppId([NewProfileAppId: Guid])
```

NOTE

This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewProfileAppId

Type: Guid

Sets the GUID of the extension that provides the profile. The value must be a valid GUID for an extension in the system table **2000000072 Profile**.

Return Value

ProfileAppId Type: Guid The ID of the extension that is set in the SessionSettings object.

Remarks

A profile can be included as part of an extension, instead of being defined as part of the base application. In order to properly identify a profile from an extension in the SessionSettings object, you must specify the extension ID, by using the ProfileAppId method, and the profile ID, by using ProfileId method.

The ProfileAppId property in a SessionSettings object corresponds to the **App ID** field in the in the system table **2000000073 User Personalization**

Example

This example creates a SessionSettings object that is populated with the current client user's personalization data, and then uses the ProfileAppId method and ProfileId method to set the object to use the profile that has the ID 'MyExtensionProfile', which is provided in the extension that has the ID '12345678-1234-1234-1234-1234567890AB'. Finally, the RequestSessionUpdate method sends a request to the client to abandon the current client session and start a new session that uses the new profile. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init;
  MySessionSettings.ProfileId('MyExtensionProfile');
  MySessionSettings.ProfileAppId('12345678-1234-1234-1234-1234567890AB');
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.ProfileId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the profile ID property in a SessionSettings object.

Syntax

```
[ProfileId := ]  SessionSettings.ProfileId([NewProfileId: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewProfileId
Type: String
Specifies the ID of the profile to set in the SessionSettings object. The value must be a valid profile ID in the system table **2000000072 Profile**.

Return Value

ProfileId Type: String The profile ID that is set in the SessionSettings object.

Remarks

The profile ID determines the Role Center that is used in the client session. The ProfileId property in a SessionSettings object corresponds to the **Profile ID** field in the in the system table **2000000073 User Personalization**.

Example

This example creates a SessionSettings object that is populated with the current client user's personalization data, and uses the changes the Profile method to set the profile to 'Business Manager'. Finally, the RequestSessionUpdate method sends a request to the client to abandon the current session and start a new session that uses the new profile ID. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init;
  MySessionSettings.ProfileId('Business Manager');
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.ProfileSystemScope Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the profile scope property in a SessionSettings object.

Syntax

```
[ProfileSystemScope := ]  SessionSettings.ProfileSystemScope([NewProfileScope: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewProfileScope
Type: Boolean
Specifies whether the profile applies to the system or to a tenant only. **true** sets the profile to apply to the system; **false** sets the profile to apply to a tenant only.

Return Value

ProfileSystemScope Type: Boolean **true** if the profile applies to the system; **false** if the profile applies to a tenant.

Remarks

The ProfileSystemScope property in a SessionSettings object corresponds to the **Scope** field in the in the system table **2000000073 User Personalization**.

Example

This example creates a SessionSettings object, and then uses the ProfileSystemScope method to set the profile scope to apply to the system. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.ProfileSystemScope(true);
end;
```

See Also

SessionSettings Data Type

SessionSettings.RequestSessionUpdate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Passes a SessionSettings object to the client to request a new session that uses the user personalization properties that are set in the object. The current client session is abandoned and a new session is started.

Syntax

```
SessionSettings.RequestSessionUpdate(saveSettings: Boolean)
```

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

saveSettings
Type: Boolean

Specifies whether to save the personalization properties in the SessionSettings object to the table **2000000073 User Personalization** in the database for the current client user. **true** saves the settings; **false** does not. If you set this parameter to **true**, before sending the request to the client, the server instance will store the property values of the SessionSettings object to the corresponding fields in the table **2000000073 User Personalization**. If you set this to **false**, when the new client session is closed, the next time the user signs in, the session will return to the previous personalization settings. This enables you to use the SessionSettings object to temporarily change the personalization settings for the current session.

Remarks

Within the scope of the RequestSessionUpdate method, it is not recommended to run code after the RequestSessionUpdate method call that requires client communication because the current client session will be abandoned.

Before the RequestSessionUpdate method is called, be sure that all user personalization properties in the SessionSettings object have values; otherwise the system default values will be used for empty properties. To help, you can use the Init method to populate the object with the values in the database.

Example

This example uses the Init method to create a SessionSettings object that includes the current client user's personalization settings from the database, and then uses the Company method to set the company to 'MyCompany'. Finally, the RequestSessionUpdate method sends a request that saves the SessionSettings object property values to the database and starts a new client session that uses the new company.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init
  MySessionSettings.Company('MyCompany');
  MySessionSettings.RequestSessionUpdate(true);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions

SessionSettings.TimeZone Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the time zone property in a SessionSettings object.

Syntax

```
[TimeZone := ]  SessionSettings.TimeZone([NewTimeZone: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

SessionSettings Type: SessionSettings An instance of the SessionSettings data type.

NewTimeZone
Type: String
Specifies the time zone property in the SessionsSettings object. The value must be a valid Windows time zone name, such as **UTC** or **Pacific Standard Time**.

Return Value

TimeZone Type: String The time zone set in the SessionSettings object.

Remarks

The default time zone is **UTC**. This method can also be used with web service sessions.

Example

This example creates a SessionSettings object that is populated with the current client user's personalization data, and then uses the TimeZone method to set the time zone to 'UTC' (Coordinated Universal Time). Finally, the RequestSessionUpdate method sends a request to the client to abandon the current session and start a new session that uses the new profile ID. This example requires a SessionSettings data type variable.

```
var
  MySessionSettings : SessionSettings;
begin
  MySessionSettings.Init;
  MySessionSettings.TimeZone('UTC');
  MySessionSettings.RequestSessionUpdate(false);
end;
```

See Also

SessionSettings Data Type
Getting Started with AL
Developing Extensions
Managing Time Zones with Web Services

System Data Type

2/17/2021 • 6 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a complex data type.

The following methods are available on the System data type.

METHOD NAME	DESCRIPTION
Abs(Decimal)	Calculates the absolute value of a number (Decimal, Integer or BigInteger). ABS always returns a positive numeric value or zero.
ApplicationPath()	Returns the path of the directory where the executable file for the product is installed.
ArrayLen(Array of [Any] [, Integer])	Returns the total number of elements in an array or the number of elements in a specific dimension.
CalcDate(String [, Date])	Calculates a new date that is based on a date expression and a reference date.
CalcDate(DateFormula [, Date])	Calculates a new date that is based on a date expression and a reference date.
CanLoadType(DotNet)	Tests if the specified .NET Framework type can be loaded.
CaptionClassTranslate(String)	Returns a translated version of the caption string. The string is translated to the current local language.
Clear(var Array of [Any])	Clears the value of a single variable. Also, it clears all the filters that were set if the variable is a record and resets the key to the primary key and the company on a record variable.
Clear(var Any)	Clears the value of a single variable. Also, it clears all the filters that were set if the variable is a record and resets the key to the primary key and the company on a record variable.
ClearAll()	Clears all internal variables (except REC variables), keys, and filters in the object and in any associated objects, such as reports, pages, codeunits, and so on that contain AL code.
ClearLastError()	Removes the last error message from memory.
ClosingDate(Date)	Gets the closing date for a Date Data Type.
CodeCoverageInclude(var Record)	Includes the code that has been logged.

METHOD NAME	DESCRIPTION
CodeCoverageLoad()	Loads the code that has been logged.
CodeCoverageLog([Boolean] [, Boolean])	Starts and stops the logging of code. You can also use this method to retrieve the current logging status.
CodeCoverageRefresh()	Refreshes the code that has been logged.
CompressArray(Array of [String])	Moves all non-empty strings (text) in an array to the beginning of the array. The resulting StringArray has the same number of elements as the input array, but empty entries appear at the end of the array.
CopyArray(Array of [Any], Array of [Any], Integer [, Integer])	Copies one or more elements in an array to a new array.
CopyStream(OutStream, InStream [, Integer])	Copies the information that is contained in an InStream to an OutStream.
CreateDateTime(Date, Time)	Creates a DateTime object from a date and a time.
CreateEncryptionKey()	Creates an encryption key for the current tenant.
CreateGuid()	Creates a new unique GUID. The value can then be assigned to a GUID data type or a text data type. Use the text data type if you want to compare the GUID to another text string.
CurrentDateTime()	Gets the current DateTime.
Date2DMY(Date, Integer)	Gets the day, month, or year of a Date Data Type.
Date2DWY(Date, Integer)	Gets the day of the week, week number, or year of a Date Data Type.
DaTi2Variant(Date, Time)	Creates a variant that contains an encapsulation of a COM VT_DATE.
Decrypt(String)	Takes a string as input and returns the decrypted value of the string.
DeleteEncryptionKey()	Deletes an encryption key for the current tenant.
DMY2Date(Integer [, Integer] [, Integer])	Gets a Date object based on a day, month, and year.
DT2Date(DateTime)	Gets the date part of a DateTime object.
DT2Time(DateTime)	Gets the time part of a DateTime object.
DWY2Date(Integer [, Integer] [, Integer])	Gets a Date that is based on a week day, a week, and a year.
Encrypt(String)	Takes a string as input and returns the encrypted value of the string.

METHOD NAME	DESCRIPTION
EncryptionEnabled()	Checks if the tenant is configured to allow encryption.
EncryptionKeyExists()	Checks whether an encryption key for the current tenant is present on the server tenant.
Evaluate(var Any, String [, Integer])	Evaluates a string representation of a value into its typical representation. The result is assigned to a variable.
ExportEncryptionKey(String)	Returns a password protected temporary filepath containing the encryption key. When encrypting or decrypting data in Dynamics 365 Business Central, an encryption key is used. A single key is used per tenant and every tenant will have a different key. Keys can be exported to a file which may be necessary in the case of upgrading or migrating a system from one set of hardware to another. The EXPORTEncryptIONKey method allows an administrator to specify a destination file for the key and specify a password protection for the file.
ExportObjects(String, var Record [, Integer])	Exports application objects to a file.
Format(Any [, Integer] [, Integer])	Formats a value into a string.
Format(Any, Integer, String)	Formats a value into a string.
GetDocumentUrl(Guid)	Gets the URL for the specified temporary media object ID.
GetDotNetType(Any)	Gets the System.Type that corresponds to the given value.
GetLastErrorCallStack()	Gets the call stack from where the last error occurred.
GetLastErrorCode()	Gets the classification of the last error that occurred.
GetLastErrorObject()	Gets the last System.Exception object that occurred.
GetLastErrorText()	Gets the last error that occurred in the debugger.
GetUrl(ClientType [, String] [, ObjectType] [, Integer] [, Record] [, Boolean])	Generates a URL for the specified client target that is based on the configuration of the server instance. If the code runs in a multitenant deployment architecture, the generated URL will automatically apply to the tenant ID of the current user.
GetUrl(ClientType, String, ObjectType, Integer, RecordRef [, Boolean])	Generates a URL for the specified client target that is based on the configuration of the server instance. If the code runs in a multitenant deployment architecture, the generated URL will automatically apply to the tenant ID of the current user.
GlobalLanguage([Integer])	Gets and sets the current global language setting.
GuiAllowed()	Checks whether the AL code can show any information on the screen.
Hyperlink(String)	Passes a URL as an argument to an Internet browser, such as Windows Internet Explorer.

METHOD NAME	DESCRIPTION
ImportEncryptionKey(String, String)	Points to a password protected file that contains the key on the current server. When encrypting or decrypting data in Dynamics 365 Business Central, an encryption key is used. A single key is used per tenant, and every tenant will have a different key. Keys can be created or imported if one exists already, as may be the case if upgrading or migrating a system from one set of hardware to another. The IMPORTEncryptIONKey method allows an administrator to specify a file (password protected) which contains a key and imports it to the current Dynamics 365 Business Central service.
ImportObjects(String [, Integer])	Imports application objects from a file.
ImportStreamWithUrlAccess(InStream, String [, Integer])	Imports an object into a media container to be used in a temporary URL with a default expiration time.
IsNull(DotNet)	Gets a value indicating whether a DotNet object has been created or not.
IsNullGuid(Guid)	Indicates whether a value has been assigned to a GUID. A null GUID that consists only of zeros is valid but must never be used for references.
IsServiceTier()	Gets a value indicating whether the runtime is a service tier.
NormalDate(Date)	Gets the regular date (instead of the closing date) for the argument Date.
Power(Decimal, Decimal)	Raises a number to a power. For example, you can use this method to square the number 2 to get the result of 4.
Random(Integer)	Returns a pseudo-random number.
Randomize([Integer])	Generates a set of random numbers from which the RANDOM method (Integer) will select a random number.
Round(Decimal [, Decimal] [, String])	Rounds the value of a numeric variable.
RoundDateTime(DateTime [, BigInteger] [, String])	Rounds a DateTime.
Sleep(Integer)	Returns control to the operating system for a specified time.
TemporaryPath()	Gets the path of the directory where the temporary file is stored.
Time()	Gets the current time from the operating system.
Today()	Gets the current date set in the operating system.
Variant2Date(Variant)	Gets a date from a variant.
Variant2Time(Variant)	Gets a time from a variant.

METHOD NAME	DESCRIPTION
WindowsLanguage()	Gets the current Windows language setting.
WorkDate([Date])	Gets and sets the work date for the current session.

See Also

Getting Started with AL
Developing Extensions

System.Abs Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates the absolute value of a number (Decimal, Integer or BigInteger). ABS always returns a positive numeric value or zero.

Syntax

```
NewNumber :=   System.Abs(Number: Decimal)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Number

Type: Decimal

The input value.

Return Value

NewNumber Type: Decimal

Remarks

The system automatically converts all of the numeric data types for you.

Example

This example shows how to remove the sign from a negative numeric value.

```
var
    x: Decimal;
    y: Decimal;
    Text000: Label "x = %1, y = %2";
begin
    x := -10.235; // x is assigned a negative value
    y := ABS(x); // y is assigned the value of x without sign
    Message(Text000, x, y);
end;
```

The message window displays the following:

x = -10.235, y = 10.235

See Also

System.ApplicationPath Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the path of the directory where the executable file for the product is installed.

NOTE

This method is supported only in Business Central on-premises.

Syntax

```
String :=   System.ApplicationPath()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

String Type: String

Remarks

This method returns a string that contains the path of the directory where the executable file for Dynamics 365 Business Central is installed. This string ends with a backslash ('\') and does not contain the name of the executable file.

The string cannot contain more than 255 characters.

If this method is called from an application that is running on a Dynamics 365 Business Central Application Server, it returns the path of the directory where the Dynamics 365 Business Central Application Server is installed.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ArrayLen Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the total number of elements in an array or the number of elements in a specific dimension.

Syntax

```
Length :=   System.ArrayLen(Array: Array of [Any] [, Dimension: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Array

Type: Any

The array that you want to investigate.

Dimension

Type: Integer

If you omit this optional argument, the method returns the total number of elements in the array. To get the number of elements in a specific dimension, use Dimension with a valid value. The valid values are determined by the number of dimensions of the input array. For example, the valid values for a three-dimensional array would be 1, 2, and 3.

Return Value

Length Type: Integer

Remarks

If you use ArrayLen with an input parameter that is not an array, a run-time error occurs.

Example

This example shows how to use the ArrayLen method.

```
var
    Array1: array[2] of Integer;
    Array2: array[3,4] of Integer;
begin
    Message('Array1, Total number of elements: %1', ArrayLen(Array1));
    Message('Array2, Dimension 1 size: %1', ArrayLen(Array2,1));
    Message('Array2, Dimension 2 size: %1', ArrayLen(Array2,2));
    Message('Array2, Total number of elements: %1', ArrayLen(Array2));
end;
```

The following messages are displayed.

Array1, Total number of elements: 2

Array2, Dimension 1 size: 3

Array2, Dimension 2 size: 4

Array2, Total number of elements: 12

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CalcDate Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates a new date that is based on a date expression and a reference date.

Syntax

```
NewDate :=   System.CalcDate(DateExpression: String [, Date: Date])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

DateExpression

Type: String

The date expression can be any length. The string is interpreted from left to right with one subexpression at a time. The following rules describe the valid syntax of date expressions:

- DateExpression = [<Subexpression>][<Subexpression>][<Subexpression>]
- <Subexpression> = [<Sign>] <Term>- <Sign> = + | -
- <Term> = <Number><Unit> | <Unit><Number> | <Prefix><Unit>
- <Number> = Positive integer
- <Unit> = D | WD | W | M | Q | Y (D=day, WD=weekday, W=week, M=month, Q=quarter, Y=year)
- <Prefix> = C (C=current) These production rules show that date expressions consist of zero, one, two, or three subexpressions. Each subexpression consists of an optional sign and a term. The following are some typical examples of terms:
- 30D (30 days; corresponds to <Number><Unit>)
- WD2 (weekday number 2; corresponds to <Unit><Number>)
- CW (current week; corresponds to <Prefix><Unit>) The internal calendar starts on Monday and ends on Sunday. This means that Monday is weekday 1 and Sunday is weekday 7. A run-time error occurs if the syntax of DateExpression is incorrect.

Date

Type: Date

Use this optional parameter to define a reference date. The default is the current system date. If you omit this optional value, the current system date is used.

Return Value

NewDate Type: Date

Remarks

DateExpression can be a field or variable of type Text or Code, and it can be a field or variable of type DateFormula. The benefit of using a DateFormula field or variable is that the date formula becomes language independent.

The user can enter formulas in the currently selected language. The formula is stored in a generic format in a field or variable. When the formula must be displayed, the actual string that is displayed is converted to the currently selected language.

For example, if a user who has language set to ENG (English) enters the date formula "1W+1D" for one week and one day, then a user who has the language set to FRA (French) sees "1S+1J," and a user who has the language set to ESP (Spanish) sees "1S+1D".

If a date formula is entered with < > delimiters surrounding it, then the date formula is stored in a generic, nonlanguage-dependent format. This makes it possible to develop date formulas that are not dependent on the currently selected language.

For more information about how to calculate the duration between two DateTimes, see Duration Data Type.

Example 1

This code example shows how to use the production rules that were previously described.

```
<CQ+1M-10D>
```

This should be interpreted as the following: current quarter + 1 month - 10 days.

The DateExpression is composed of the following:

<Prefix><Unit><Sign><Number><Unit><Sign><Number><Unit>

> **NOTE**
> The angle brackets (< >) specify that the expression is not translated, regardless of the application language.

Example 2

This example shows how to use the CalcDate method.

```
var
    Expr1: Text[30];
    Expr2: Text[30];
    Expr3: Text[30];
    RefDate: Date;
    Date1: Date;
    Date2: Date;
    Date3: Date;
    RefDateTxt: Label 'The reference date is: %1 \\';
    Expr1Txt: Label 'The expression: %2 returns %3\\';
    Expr2Txt: Label 'The expression: %4 returns %5\\';
    Expr3Txt: Label 'The expression: %6 returns %7';
begin
    Expr1 := '<CQ+1M-10D>'; // Current quarter + 1 month - 10 days
    Expr2 := '<-WD2>'; // The last weekday no.2, (last Tuesday)
    Expr3 := '<CM+30D>'; // Current month + 30 days
    RefDate := 19960521D;
    Date1 := CalcDate(Expr1, RefDate);
    Date2 := CalcDate(Expr2, RefDate);
    Date3 := CalcDate(Expr3, RefDate);
    Message(RefDateTxt + Expr1Txt + Expr2Txt + Expr3Txt,
      RefDate, Expr1, Date1, Expr2, Date2, Expr3, Date3);
end;
```

The message window displays the following text:

The reference date is: 05/21/96

The expression: CQ+1M-10D returns 07/20/96

The expression: -WD2 returns 05/14/96

The expression: CM+30D returns 06/30/96

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CalcDate Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates a new date that is based on a date expression and a reference date.

Syntax

```
NewDate :=   System.CalcDate(DateExpression: DateFormula [, Date: Date])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

DateExpression

Type: DateFormula

The date expression can be any length. The string is interpreted from left to right with one subexpression at a time. The following rules describe the valid syntax of date expressions:

- DateExpression = [<Subexpression>][<Subexpression>][<Subexpression>]
- <Subexpression> = [<Sign>] <Term>
- <Sign> = + | -
- <Term> = <Number><Unit> | <Unit><Number> | <Prefix><Unit>
- <Number> = Positive integer
- <Unit> = D | WD | W | M | Q | Y (D=day, WD=weekday, W=week, M=month, Q=quarter, Y=year)
- <Prefix> = C (C=current) These production rules show that date expressions consist of zero, one, two, or three subexpressions. Each subexpression consists of an optional sign and a term. The following are some typical examples of terms:
- 30D (30 days; corresponds to <Number><Unit>)
- WD2 (weekday number 2; corresponds to <Unit><Number>)
- CW (current week; corresponds to <Prefix><Unit>) The internal calendar starts on Monday and ends on Sunday. This means that Monday is weekday 1 and Sunday is weekday 7. A run-time error occurs if the syntax of DateExpression is incorrect.

Date

Type: Date

Use this optional parameter to define a reference date. The default is the current system date. If you omit this optional value, the current system date is used.

Return Value

NewDate Type: Date

Remarks

DateExpression can be a field or variable of type Text or Code, and it can be a field or variable of type DateFormula. The benefit of using a DateFormula field or variable is that the date formula becomes language independent.

The user can enter formulas in the currently selected language. The formula is stored in a generic format in a field or variable. When the formula must be displayed, the actual string that is displayed is converted to the currently selected language.

For example, if a user who has language set to ENG (English) enters the date formula "1W+1D" for one week and one day, then a user who has the language set to FRA (French) sees "1S+1J," and a user who has the language set to ESP (Spanish) sees "1S+1D".

If a date formula is entered with < > delimiters surrounding it, then the date formula is stored in a generic, nonlanguage-dependent format. This makes it possible to develop date formulas that are not dependent on the currently selected language.

For more information about how to calculate the duration between two DateTimes, see Duration Data Type.

Example 1

This code example shows how to use the production rules that were previously described.

```
<CQ+1M-10D>
```

This should be interpreted as the following: current quarter + 1 month - 10 days.

The DateExpression is composed of the following:

<Prefix><Unit><Sign><Number><Unit><Sign><Number><Unit>

> **NOTE**
> The angle brackets (< >) specify that the expression is not translated, regardless of the application language.

Example 2

This example shows how to use the CalcDate method.

```
var
    Expr1: Text[30];
    Expr2: Text[30];
    Expr3: Text[30];
    RefDate: Date;
    Date1: Date;
    Date2: Date;
    Date3: Date;
    Text000: Label 'The reference date is: %1 \\';
    Text001: Label 'The expression: %2 returns %3\\';
    Text002: Label 'The expression: %4 returns %5\\';
    Text003: Label 'The expression: %6 returns %7';
begin
    Expr1 := '<CQ+1M-10D>'; // Current quarter + 1 month - 10 days
    Expr2 := '<-WD2>'; // The last weekday no.2, (last Tuesday)
    Expr3 := '<CM+30D>'; // Current month + 30 days
    RefDate := 19960521D;
    Date1 := CalcDate(Expr1, RefDate);
    Date2 := CalcDate(Expr2, RefDate);
    Date3 := CalcDate(Expr3, RefDate);
    Message(Text000 + Text001 + Text002 + Text003,
      RefDate, Expr1, Date1, Expr2, Date2, Expr3, Date3);
end;
```

The message window displays the following text:

The reference date is: 05/21/96

The expression: CQ+1M-10D returns 07/20/96

The expression: -WD2 returns 05/14/96

The expression: CM+30D returns 06/30/96

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CanLoadType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Tests if the specified .NET Framework type can be loaded.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Ok :=   System.CanLoadType(DotNet: DotNet)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

DotNet
Type: DotNet
A variable of the DotNet data type to represent the .NET Framework type.

Return Value

Ok Type: Boolean

Example

The following code example is based on codeunit 5300 in the CRONUS International Ltd. demonstration database.

```
var
    OObjLibrary: DotNet
"Microsoft.Dynamics.NAV.OLSync.OLSyncSupplier.OutlookObjectLibrary.'Microsoft.Dynamics.NAV.OLSync.OLSyncSupp
lier, Version=7.1.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35'";
    MyError: Label 'Cannot access the specified type.';
if not CanLoadType(OObjLibrary) then
   Error(MyError);
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CaptionClassTranslate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a translated version of the caption string. The string is translated to the current local language.

Syntax

```
String :=   System.CaptionClassTranslate(CaptionClassText: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

CaptionClassText
Type: String
A literal string that defines the caption.

Return Value

String Type: String

Remarks

You can use the SelectLatestVersion Method (Database) to clear the current session's cache for the CaptionClassTranslate strings. The strings will then be reevaluated by the **CaptionClassTranslate** method trigger in codeunit 42.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Clear Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Clears the value of a single variable. Also, it clears all the filters that were set if the variable is a record and resets the key to the primary key and the company on a record variable.

Syntax

```
System.Clear(var Variable: Any)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Variable

Type: Any

The identifier (variable) of any AL data type, including simple and composite data types. The following rules apply when you run the Clear method:

- A number variable is set to 0 (zero)
- A string variable is set to empty string
- A date variable is set to 0D (undefined date)
- A time variable is set to 0T (undefined time)
- A Boolean variable is set to False

Remarks

Use the ClearAll Method to clear all internal variables, keys, and filters in the object and in any associated objects such as reports, pages, and codeunits that contain AL code. Note, however, that ClearAll Method does not affect or change values for variables in single instance codeunits.

For a composite data type, such as a record or an array, all elements are cleared. Furthermore, all fields in a record will be initialized with the InitValue Property of the field.

Clear can also be used on the Guid Data Type. It converts the GUID to zeros. Use the CreateGuid Method (Guid) to create a new unique GUID.

Clear can also be used to deselect a company. For more information, see ChangeCompany Method (Record).

If you use **Clear** on a codeunit, only the reference to the codeunit is deleted and not the codeunit itself, as with Automation objects. This means that the content of the codeunit stays intact.

If you have an array of controls, the whole array is destroyed when the array variable goes out of scope.

If you have an array of automation servers, you may clear the whole array at once, or clear each element individually.

Example 1

This example shows how to use the **Clear** method.

```
var
    Text000: Label 'Joe Raybon';
    Text001: Label 'Initially the variable "Name" contains: >;%1\>';
    Text002: Label 'After using Clear, the variable "Name" contains: >;%1\>';
begin
    Name := Text000;
    Message(Text001, Name);
    Clear(Name);
    Message(Text002, Name);
end;
```

The first message window displays the following:

Initially the variable "Name" contains: >;Joe Raybon>

The second message window displays the following:

After using Clear, the variable "Name" contains: >;>

Example 2

In the following example you will declare two variables:

```
var
    MyTextVar: Text;
    GuidVar: Guid;
```

These variables will be declared and cleared, first by using **Clear** and then by using ClearAll Method.

```
var
    Text000: Label 'My Text';
    Text001: Label 'Initially the variable "MyTextVar" contains >;%1> and "GuidVar" is defined as >;%2>';
    Text002: Label 'After using Clear(MyTextVar), the variable "MyTextVar" contains >;%1> and "GuidVar" is
still defined as >;%2>';
    Text003: Label 'After using Clear(GuidVar) the variable "GuidVar" becomes undefined';
    Text004: Label 'Giving the "MyTextVar" variable the initial value again and creating a new "GuidVar"
results in >;%1> and >;%2>';
    Text005: Label 'Using ClearAll results in an empty "MyTextVar" >;%1> and an undefined "GuidVar"';
begin
    MyTextVar : Text000;
    GuidVar := CreateGuid();
    Message(Text001,MyTextVar,GuidVar);
    Clear(MyTextVar);
    Message(Text002,MyTextVar,GuidVar);
    Clear(GuidVar);
    Message(Text003,GuidVar);
    MyTextVar := Text000;
    GuidVar := CreateGuid();
    Message(Text004,MyTextVar,GuidVar);
    ClearAll;
    Message(Text005,MyTextVar,GuidVar);
end;
```

The first message window displays the following:

Initially the variable "MyTextVar" contains: >;My Text> and "GuidVar" is defined as >;12345678-

1234-1234-1234-1234567890AB>

The second message window displays the following:

After using Clear(MyTextVar), the variable "MyTextVar" contains: >;> and GuidVar is still defined as >;12345678-1234-1234-1234-1234567890AB>

The third message window displays the following:

After using Clear(GuidVar) the variable "GuidVar" becomes undefined

The fourth message window displays the following:

Giving the "MyTextVar" variable the initial value again and creating a new "GuidVar" results in >;My Text> and >;87654321-4321-4321-4321-BA0987654321>

The fifth message window displays the following:

Using ClearAll results in an empty "MyTextVar" >;> and an undefined "GuidVar"

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Clear Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Clears the value of a single variable. Also, it clears all the filters that were set if the variable is a record and resets the key to the primary key and the company on a record variable.

Syntax

```
System.Clear(var Variable: Any)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Variable

Type: Any

The identifier (variable) of any AL data type, including simple and composite data types. The following rules apply when you run the Clear method:

- A number variable is set to 0 (zero)
- A string variable is set to empty string
- A date variable is set to 0D (undefined date)
- A time variable is set to 0T (undefined time)
- A Boolean variable is set to False

Remarks

Use the ClearAll Method to clear all internal variables, keys, and filters in the object and in any associated objects such as reports, pages, and codeunits that contain AL code. Note, however, that ClearAll Method does not affect or change values for variables in single instance codeunits.

For a composite data type, such as a record or an array, all elements are cleared. Furthermore, all fields in a record will be initialized with the InitValue Property of the field.

Clear can also be used on the Guid Data Type. It converts the GUID to zeros. Use the CreateGuid Method (Guid) to create a new unique GUID.

Clear can also be used to deselect a company. For more information, see ChangeCompany Method (Record).

If you use **Clear** on a codeunit, only the reference to the codeunit is deleted and not the codeunit itself, as with Automation objects. This means that the content of the codeunit stays intact.

If you have an array of controls, the whole array is destroyed when the array variable goes out of scope.

If you have an array of automation servers, you may clear the whole array at once, or clear each element individually.

Example 1

This example shows how to use the **Clear** method.

```
var
    Text000: Label 'Joe Raybon';
    Text001: Label 'Initially the variable "Name" contains: >;%1\>';
    Text002: Label 'After using Clear, the variable "Name" contains: >;%1\>';
begin
    Name := Text000;
    Message(Text001, Name);
    Clear(Name);
    Message(Text002, Name);
end;
```

The first message window displays the following:

Initially the variable "Name" contains: >;Joe Raybon>

The second message window displays the following:

After using Clear, the variable "Name" contains: >;>

Example 2

In the following example you will declare two variables:

```
var
    MyTextVar: Text;
    GuidVar: Guid;
```

These variables will be declared and cleared, first by using **Clear** and then by using ClearAll Method.

```
var
    Text000: Label 'My Text';
    Text001: Label 'Initially the variable "MyTextVar" contains >;%1> and "GuidVar" is defined as >;%2>';
    Text002: Label 'After using Clear(MyTextVar), the variable "MyTextVar" contains >;%1> and "GuidVar" is
still defined as >;%2>';
    Text003: Label 'After using Clear(GuidVar) the variable "GuidVar" becomes undefined';
    Text004: Label 'Giving the "MyTextVar" variable the initial value again and creating a new "GuidVar"
results in >;%1> and >;%2>';
    Text005: Label 'Using ClearAll results in an empty "MyTextVar" >;%1> and an undefined "GuidVar"';
begin
    MyTextVar : Text000;
    GuidVar := CreateGuid();
    Message(Text001,MyTextVar,GuidVar);
    Clear(MyTextVar);
    Message(Text002,MyTextVar,GuidVar);
    Clear(GuidVar);
    Message(Text003,GuidVar);
    MyTextVar := Text000;
    GuidVar := CreateGuid();
    Message(Text004,MyTextVar,GuidVar);
    ClearAll;
    Message(Text005,MyTextVar,GuidVar);
end;
```

The first message window displays the following:

Initially the variable "MyTextVar" contains: >;My Text> and "GuidVar" is defined as >;12345678-

1234-1234-1234-1234567890AB>

The second message window displays the following:

After using Clear(MyTextVar), the variable "MyTextVar" contains: >;> and GuidVar is still defined as >;12345678-1234-1234-1234-1234567890AB>

The third message window displays the following:

After using Clear(GuidVar) the variable "GuidVar" becomes undefined

The fourth message window displays the following:

Giving the "MyTextVar" variable the initial value again and creating a new "GuidVar" results in >;My Text> and >;87654321-4321-4321-4321-BA0987654321>

The fifth message window displays the following:

Using ClearAll results in an empty "MyTextVar" >;> and an undefined "GuidVar"

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ClearAll Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Clears all internal variables (except REC variables), keys, and filters in the object and in any associated objects, such as reports, pages, codeunits, and so on that contain AL code.

Syntax

```
System.ClearAll()
```

NOTE

This method can be invoked without specifying the data type name.

Remarks

ClearAll works by calling the Clear Method repeatedly on each variable. However, this is not the case with codeunits, where the ClearAll method works by calling ClearAll inside the codeunit. It deletes the contents of the codeunit, whereas Clear only deletes the reference to the codeunit.

ClearAll does not affect or change values for variables in single instance codeunits.

When a method is called repeatedly in the same transaction, the system retains all values for variables and filters in memory between calls. For example, this is used to assign numbers to entry numbers when posting. When you do not want to retain the values in memory, use the ClearAll method to clear them.

For information about the initial values of cleared variables, see the Clear Method. Take into consideration that fields in a record will be initialized with the InitValue Property of the field.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ClearLastError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the last error message from memory.

Syntax

```
System.ClearLastError()
```

NOTE

This method can be invoked without specifying the data type name.

Remarks

You can use the GetLastErrorText Method to determine whether an error has occurred and to see the text in the last error message that was generated. You can then use the ClearLastError method to remove the last error message from memory. If you subsequently call the GetLastErrorText Method, an empty string is returned.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ClosingDate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the closing date for a Date Data Type.

Syntax

```
ClosingDate :=   System.ClosingDate(Date: Date)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Date
Type: Date
The input date.

Return Value

ClosingDate Type: Date

Remarks

All dates have a corresponding closing date. A closing date is a period in time following the given date but before the next regular date. Closing dates are sorted immediately after the corresponding regular date but before the next regular date.

xxxxxxD: Regular date

xxxxxxC: Closing date

The compiler cannot convert the expression xxxxxxC to a Date data type. Therefore, you must use the ClosingDate method to create a closing date.

Example 1

The first example shows how to use the ClosingDate method. A regular date is given as input.

```
var
    Date1: Date;
    CloDate: Date;
    Text000: Label 'The closing date for %1 is %2.';
begin
    Date1 := 20140404D;
    CloDate := ClosingDate(Date1);
    Message(Text000, Date1, CloDate);
end;
```

The following message is displayed:

The closing date for 04/04/14 is C04/04/14.

Example 2

The second example shows some statements that do not work and explains why they do not work.

```
var
    Date1: Date;
    CloDate1: Date;
    CloDate2: Date;
    Text000: Label 'The closing date for %1 is %2.';
begin
    // Date1 := 20140404C;
    // The previous statement does not compile because the compiler
    // cannot convert '20140404C' to a Date data type.
    Date1 := 20140404D;
    // The previous statement compiles.
    // The compiler converts '20140404D' to a Date data type.
    // CloDate1 := ClosingDate(20140505C);
    // The previous statement does not compile because the compiler
    // cannot convert '20140505C' to a Date data type and the ClosingDate
    // method requires a Date data type for its parameter.
    CloDate1 := ClosingDate(Date1);
    // The previous statement compiles.
    // Date1 is a Date data type.
    CloDate2 := ClosingDate(CloDate1);
    // The previous statement compiles.
    // CloDate1 is a Date data type.
    Message(Text001, CloDate1, CloDate2);
end;
```

The following message is displayed:

The closing date for C04/04/14 is C04/04/14.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CodeCoverageInclude Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Includes the code that has been logged.

Syntax

```
System.CodeCoverageInclude(var ObjectRecord: Record)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

ObjectRecord
Type: Record

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CodeCoverageLoad Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads the code that has been logged.

Syntax

```
System.CodeCoverageLoad()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CodeCoverageLog Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Starts and stops the logging of code. You can also use this method to retrieve the current logging status.

Syntax

```
[IsActive := ]  System.CodeCoverageLog([NewIsActive: Boolean] [, MultiSession: Boolean])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

NewIsActive

Type: Boolean

true starts code logging;**false** stops code logging.

MultiSession

Type: Boolean

Return Value

IsActive Type: Boolean **true** is code logging is active;**false** otherwise.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CodeCoverageRefresh Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Refreshes the code that has been logged.

Syntax

```
System.CodeCoverageRefresh()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CompressArray Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Moves all non-empty strings (text) in an array to the beginning of the array. The resulting StringArray has the same number of elements as the input array, but empty entries appear at the end of the array.

Syntax

```
[Count := ]  System.CompressArray(StringArray: Array of [String])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

StringArray
Type: String
The string array that you want to compress.

Return Value

Count Type: Integer

Remarks

When compressing an array of strings, the non-empty strings in the resulting array are sorted just like in the original array.

The **CompressArray** method is useful for printing names and addresses. For instance, you can use this method to remove blank lines in account statements.

This method compresses an array of Text or Code, but not BigText. The compression is performed by moving empty strings to the end of the array.

For example, "Redmond", "Copenhagen", "", "Fargo", "Paris" is changed to "Redmond", "Copenhagen", "Fargo", "Paris", "".

In Dynamics 365 Business Central, it is not supported to use the **CompressArray** method on multidimensional arrays. In earlier versions, **CompressArray** works for arrays of arrays.

Example

This example shows how to use the **CompressArray** method. The input array has the following values: Joe Raybon, One Meca Way, Atlanta.

The output StringArray has the following values: Joe Raybon, One Meca Way, Atlanta.

All non-empty entries have been moved to the beginning of the array.

```
var
    CustomerData: array[6] of Text;
begin
    CustomerData[1] := 'Joe Raybon';
    CustomerData[2] := ''; // Empty string.
    CustomerData[3] :='One Meca Way';
    CustomerData[4] := '  '; // A non-empty string that contains blanks.
    CustomerData[5] := 'Atlanta';
    CustomerData[6] := ''; // Empty string.
    Message('Before compression the address is: \%1\%2\%3\%4\%5\%6', CustomerData[1], CustomerData[2],
CustomerData[3], CustomerData[4], CustomerData[5], CustomerData[6]);
    CompressArray(CustomerData); // The empty lines (strings) are removed.
    Message('After compression the address is: \%1\%2\%3\%4\%5\%6', CustomerData[1], CustomerData[2],
CustomerData[3], CustomerData[4], CustomerData[5], CustomerData[6]);
end;
```

The first message window displays the following:

Before compression, the address is:

Joe Raybon

One Meca Way

Atlanta

The second message window displays the following:

After compression, the address is:

Joe Raybon

One Meca Way

Atlanta

All empty strings, which cause blank lines, are moved to the end of the array.

> **NOTE**
> Empty lines are not printed if they occur on the first or last line in a message window.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CopyArray Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies one or more elements in an array to a new array.

Syntax

```
System.CopyArray(NewArray: Array of [Any], Array: Array of [Any], Position: Integer [, Length: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

NewArray
Type: Any
The array to copy to; the destination array.

Array
Type: Any

Position
Type: Integer
The position of the first array element to copy.

Length
Type: Integer
The number of array elements to copy. If you do not specify Length, all array elements are copied from Position to the last element. Use the following equation to determine valid values. 1 =< LENGTH =< MAXLEN(ARRAY) - POSITION + 1

Remarks

You can only copy from one-dimensional arrays. Repeat the CopyArray method to copy two-dimensional and three-dimensional arrays.

You cannot copy an array if the data type of the array is a complex data type. For more information about complex data types, see AL Data Types.

Example 1

The following example assigns values to Array1 and copies values from Array1 to Array2. Array1 is an integer array with the Dimensions Property set to 10. It contains integers from 1 to 10. The example code copies the numbers 6, 7, 8, 9, and 10 to Array2, an integer array with the **Dimensions** property set to 5.

```
var
    Array1: array[10] of Integer;
    Array2: array[5] of Integer;
begin
    Array1[1] := 1;
    Array1[2] := 2;
    Array1[3] := 3;
    Array1[4] := 4;
    Array1[5] := 5;
    Array1[6] := 6;
    Array1[7] := 7;
    Array1[8] := 8;
    Array1[9] := 9;
    Array1[10] := 10;
    COPYARRAY(Array2,Array1,6,5);
end;
```

Example 2

If Array1 is an integer array with dimension 10, and it contains the numbers from 1 to 10, and Array2 is an integer array with dimension 5, then the following command causes a run-time error.

```
CopyArray(Array2,Array1,3);
```

The error occurs because the code attempts to copy eight elements from Array1 to Array2, and Array2 has room for only five elements.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CopyStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies the information that is contained in an InStream to an OutStream.

Syntax

```
[Ok := ]  System.CopyStream(OutStream: OutStream, InStream: InStream [, BytesToRead: Integer])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

OutStream

Type: OutStream

The OutStream object to which you will copy the information; the destination stream.

InStream

Type: InStream

The InStream object from which you want to copy; the source stream.

BytesToRead

Type: Integer

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

```
var
    F1: File;
    F2: File;
    InS: InStream;
    OutS: OutStream;
begin
    F1.Open('c:\Test.txt');
    F1.CreateInStream(InS);
    F2.Create('c:\CopyTest.txt');
    F2.CreateOutStream(OutS);
    CopyStream(OutS,InS);
    F1.Close();
    F2.Close();
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CreateDateTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a DateTime object from a date and a time.

Syntax

```
Datetime :=   System.CreateDateTime(Date: Date, Time: Time)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Date

Type: Date

The date that you want to use to create a DateTime. You cannot use an undefined date to create a DateTime.

Time

Type: Time

The time that you want to use to create a DateTime. You cannot use an undefined time to create a DateTime.

Return Value

Datetime Type: DateTime

Example

```
var
    TestDate: Date;
    TestTime: Time;
    TestDateTime: DateTime;
begin
    TestDate := Today;
    TestTime := Time;
    TestDateTime := CreateDateTime(TestDate,TestTime);
    …
    TestDateTime := CreateDateTime(081111D,020000T);
    …
    TestDateTime := CreateDateTime(010101D,0T);
end;
```

See Also

System Data Type
Getting Started with AL

System.CreateEncryptionKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an encryption key for the current tenant.

Syntax

```
[Ok := ]  System.CreateEncryptionKey()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If a key already exists, the following error will be displayed: **Unable to create a new encryption key. An encryption key already exists**.

Example

This code example creates an encryption key for the current tenant. It uses the EncryptionEnabled method to perform a check.

```
if not EncryptionEnabled then
        CreateEncryptionKey();
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CreateGuid Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a new unique GUID. The value can then be assigned to a GUID data type or a text data type. Use the text data type if you want to compare the GUID to another text string.

Syntax

```
Guid :=   System.CreateGuid()
```

NOTE

This method can be invoked without specifying the data type name.

Return Value

Guid Type: Guid

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.CurrentDateTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current DateTime.

Syntax

```
Datetime :=   System.CurrentDateTime()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

Datetime Type: DateTime

Example

This example requires that you create a DateTime variable named TestDateTime.

```
TestDateTime := CurrentDateTime;
Message(Format(TestDateTime));
```

The message window displays the current date and time.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Date2DMY Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the day, month, or year of a Date Data Type.

Syntax

```
Number :=   System.Date2DMY(Date: Date, Value: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Date

Type: Date

The input date.

Value

Type: Integer

Return Value

Number Type: Integer

Example

This example shows how to use the **Date2DMY** method.

```
var
    InputDate: Date;
    Day: Integer;
    Month: Integer;
    Year: Integer;
    Text000: Label 'Today is day %1 of month %2 of the year %3.';
begin
    InputDate := Today;
    Day := Date2DMY(InputDate,1);
    Month := Date2DMY(InputDate,2);
    Year := Date2DMY(InputDate,3);
    Message(Text000,Day,Month,Year);
end;
```

The message window displays the following:

Today is day 16 of month 2 of the year 2014.

See Also

System.Date2DWY Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the day of the week, week number, or year of a Date Data Type.

Syntax

```
Number :=   System.Date2DWY(Date: Date, Value: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Date

Type: Date

The input date.

Value

Type: Integer

Specifies what the function returns. The valid options are 1, 2, and 3.

- The value 1 corresponds to day of the week (1-7, Monday = 1).
- The value 2 corresponds to week number (1-53).
- The value 3 corresponds to year.

Return Value

Number Type: Integer The resulting day of the week, week number, or year.

Remarks

When the input date to the **Date2DWY** method is in a week that spans two years, the **Date2DWY** method computes the output year as the year that has the most days. For example, the input date is 010114. This date is in a week that starts on Monday, December 29, 2013, and ends Sunday, January 4, 2014. The week has three days in 2008 and four days in 2014. So the output year is 2014.

> **IMPORTANT**
> Date2DWY always uses the **ISO week-numbering year** scheme for the week, regardless of the server or device configuration. This means that week 01 of a year is the week that includes the first Thursday of the Gregorian year. Or in other words, the week that includes 4 January.

Example

This example shows a special case that occurs when you use the **Date2DWY** method in a week that spans two years.

```
var
    InputDate: Date;
    DayOfWeek: Integer;
    WeekNumber: Integer;
    Year: Integer;
    Text000: Label 'The date %1 corresponds to:\\';
    Text001: Label 'The day of the week: %2\\';
    Text002: Label 'The week number: %3\\';
    Text003: Label 'The year: %4';
begin
    InputDate := 20140101D;
    DayOfWeek := Date2DWY(InputDate, 1);
    WeekNumber := Date2DWY(InputDate, 2);
    Year := Date2DWY(InputDate, 3);
    Message(Text000 + Text001 + Text002 + Text003, InputDate, DayOfWeek, WeekNumber, Year);
end;
```

The message window displays the following information:

The date 01/01/14 corresponds to:

The day of the week: 4

The week number: 1

The year: 2014

This example shows that the date 01/01/14 is regarded as day number 4 (Thursday) in week number 1 in the year 2014.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.DaTi2Variant Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a variant that contains an encapsulation of a COM VT_DATE.

Syntax

```
Variant :=   System.DaTi2Variant(Date: Date, Time: Time)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Date
Type: Date
The input date.

Time
Type: Time
The input time.

Return Value

Variant Type: Variant

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Decrypt Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Takes a string as input and returns the decrypted value of the string.

Syntax

```
PlainTextString :=   System.Decrypt(EncryptedString: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

EncryptedString
Type: String
The input string that will be decrypted.

Return Value

PlainTextString Type: String The output string that is decrypted.

Remarks

If encryption is not enabled or the encryption key does not exist, the following error will be displayed: **An encryption key is required to complete the request**. If decryption failed because input data could not be decrypted, the following error will be displayed: **Unable to decrypt data. The data was encrypted using a different key**.

Example

This code example checks whether the tenant is configured to allow encryption using the EncryptionEnabled method, and then it decrypts an encrypted text string.

This example requires that you create the following text constants: EncryptedText and PlainText.

```
if not EncryptionEnabled then
        Error('Encryption has not been enabled.');
      PlainText := Decrypt(EncryptedText);
```

See Also

System Data Type
Getting Started with AL

System.DeleteEncryptionKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes an encryption key for the current tenant.

Syntax

```
System.DeleteEncryptionKey()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Example

This code example checks if encryption is configured for the tenant using the EncryptionEnabled method and if so, it performs the deletion of the encryption key.

```
if not EncryptionEnabled then
  Error('Encryption has not been enabled.');
  DeleteEncryptionKey();
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.DMY2Date Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a Date object based on a day, month, and year.

Syntax

```
Date :=   System.DMY2Date(Day: Integer [, Month: Integer] [, Year: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Day
Type: Integer
The number of the day in the month (1-31)

Month
Type: Integer
The number of the month in the year (1-12). If you omit this optional parameter, the current month will be used as the default.

Year
Type: Integer
The four-digit number of the year. If you omit this optional parameter, the current year is used as the default.

Return Value

Date Type: Date

Example

```
var
    Day: Integer;
    Month: Integer;
    Year: Integer;
    OutputDate: Date;
    Text000: Label "Day number %1, month number %2, and year number %3 corresponds to the date %4.";
begin
    Day := 17;
    Month := 2;
    Year := 2014;
    OutputDate := DMY2Date(Day, Month, Year);
    Message(Text000, Day, Month, Year, OutputDate);
end;
```

On a computer that has the regional format set to English (United States), the message window displays the following:

Day number 17, month number 2, and year number 2014 corresponds to the date 02/17/14.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.DT2Date Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the date part of a DateTime object.

Syntax

```
Date :=   System.DT2Date(Datetime: DateTime)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Datetime
Type: DateTime
The DateTime of which to return the date part.

Return Value

Date Type: Date

Remarks

If *Datetime* is undefined, then this method returns an undefined date.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.DT2Time Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the time part of a DateTime object.

Syntax

```
Time :=   System.DT2Time(Datetime: DateTime)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Datetime
Type: DateTime
The DateTime of which to return the time part.

Return Value

Time Type: Time

Remarks

If *Datetime* is undefined, then this method returns an undefined time.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.DWY2Date Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a Date that is based on a week day, a week, and a year.

Syntax

```
Date :=   System.DWY2Date(WeekDay: Integer [, Week: Integer] [, Year: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

WeekDay
Type: Integer
The number of the day in the week (1-7). Monday is day number 1.

Week
Type: Integer
The number of the week. Week 1 is the first week of the year that has four or more days in the year. If you omit this optional parameter, the current week is used as the default.

Year
Type: Integer
The four-digit number of the year. If you omit this optional parameter, the year of the current week is used as the default.

Return Value

Date Type: Date

Remarks

A special situation occurs if the week (*Week*) that is input into DWY2Date spans two years. Depending on *Weekday*, the year of the output *Date* can differ from the input year. This scenario is shown in the following code example.

Example 1

The input week in this example spans two years.

```
var
    DayOfWeek: Integer;
    Week: Integer;
    Year: Integer;
    OutputDate: Date;
    Text000: Label"Day %1 of week %2 in the year %3 is the date %4.";
begin
    DayOfWeek := 1;
    Week := 1;
    Year := 2014;
    OutputDate := DWY2Date(DayOfWeek, Week, Year);
    Message(Text000, DayOfWeek, Week, Year, OutputDate);
end;
```

On a computer that has the regional format set to English (United States), the message window displays the following:

Day 1 of week 1 in the year 2014 is the date: 12/30/13.

The example shows that the first day of the week in the first week of the year 2014 is regarded as the date December 30, 2013. The first week of the year 2014 is the first week that has four or more days in the year 2014. That week starts on Monday, December 30, 2013, and ends on Sunday, January 5, 2014.

Example 2

The input week in this example spans two years.

```
var
    DayOfWeek: Integer;
    Week: Integer;
    Year: Integer;
    OutputDate: Date;
    Text000: Label"Day %1 of week %2 in the year %3 is the date %4.";
begin
    DayOfWeek := 1;
    Week := 1;
    Year := 2016;
    OutputDate := DWY2Date(DayOfWeek, Week, Year);
    Message(Text000, DayOfWeek, Week, Year, OutputDate);
end;
```

On a computer that has the regional format set to English (United States), the message window displays the following:

Day 1 of week 1 in the year 2016 is the date: 01/04/16.

The example shows that the first day of the week in the first week of the year 2016 is regarded as the date January 4, 2016. The first week of the year 2016 is the first week that has four or more days in the year 2016. That week starts on Monday, January 4, 2016, and ends on Sunday, January 11, 2015.

Example 3

This example shows how to use the DWY2Date method without specifying the optional *Year* parameter. The output in this example depends on the day on which you run the code.

```
var
    DayOfWeek: Integer;
    Week: Integer;
    OutputDate: Date;
    Text000: Label"Day %1 of week %2 is the date %3.";
begin
    DayOfWeek := 1;
    Week := 1;
    OutputDate := DWY2Date(DayOfWeek, Week);
    Message(Text000, DayOfWeek, Week, OutputDate);
end;
```

On a computer that has the regional format set to English (United States), if you ran the code on January 1, 2014, then the message window displays the following:

Day 1 of week 1 is the date: 12/30/13.

If you do not specify the year, then the year of the current week is used. On January 1, 2014, the current week is the week that starts on December 30, 2013 and ends on January 5, 2014. This week has four days in 2014 so the year of the current week is 2014. The first day of the first week of 2014 is 12/30/13.

On a computer that has the regional format set to English (United States), if you ran the code on January 1, 2016, then the message window displays the following:

Day 1 of week 1 is the date: 12/29/14.

On January 1, 2016, the current week is the week that starts on December 28, 2015 and ends on January 3, 2016. This week has four days in 2015 so the year of the current week is 2015. The first day of the first week of 2015 is 12/29/14.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Encrypt Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Takes a string as input and returns the encrypted value of the string.

Syntax

```
EncryptedString :=   System.Encrypt(PlainTextString: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

PlainTextString
Type: String
The input string that will be encrypted. The input string cannot exceed a length of 215 plain characters. If the input string includes special characters the length is even more reduced.

Return Value

EncryptedString Type: String The output string that is encrypted.

Remarks

If encryption is not enabled or the encryption key is not found, the following error will be displayed: **An encryption key is required to complete the request**.

Example

```
var
    Text000: Text;
begin
    Text000 := 'ABC123';
    Message('Value: ' + Text000);
    Text000 := Encrypt(Text000);
    Message('Value: ' + Text000);
end;
```

This code example takes the string value **ABC123** and outputs the encrypted value of the string. The encrypted value will vary from system to system due to differences in the encryption key.

See Also

System Data Type

System.EncryptionEnabled Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks if the tenant is configured to allow encryption.

Syntax

```
Ok :=   System.EncryptionEnabled()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

Ok Type: Boolean

Example

This code example checks if the tenant is configured for encryption.

```
if EncryptionEnabled then
        Message('Encryption has been enabled')
else
  Message('Encryption has not been enabled')
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.EncryptionKeyExists Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether an encryption key for the current tenant is present on the server tenant.

Syntax

```
Ok :=   System.EncryptionKeyExists()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

Ok Type: Boolean

Example

This code example performs checks to determine if an encryption key already exists.

```
if EncryptionEnabled then
        if EncryptionKeyExists then
          Message('Encryption has been enabled and the encryption key is present in this server instance')
        else
          Message('Encryption has been enabled but the encryption key is not present on this server
instance')
else
  Message('Encryption has not been enabled');
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Evaluate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Evaluates a string representation of a value into its typical representation. The result is assigned to a variable.

Syntax

```
[Ok := ]  System.Evaluate(var Variable: Any, String: String [, Number: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Variable
Type: Any
The value of the string is assigned to the variable.

String
Type: String
A string that contains a value of any simple AL data type.

Number
Type: Integer
This optional value can be used when exporting data with an XmlPort. The only valid value is 9, which indicates that the data must be converted from XML format to C/SIDE format.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

This example shows how to use the **Evaluate** method when it is called with four different types of variables.

```
var
    VarInteger: Integer;
    VarDate: Date;
    VarYesNo: Boolean;
    VarDuration: Duration;
    Value: Text;
    Ok1: Boolean;
    Ok2: Boolean;
    Ok3: Boolean;
    Ok4: Boolean;
    Text000: Label 'VarInteger = \#1\#\#\#\#\#. The return code is: %2.\\';
    Text001: Label 'VarDate = \#3\#\#\#\#\#. The return code is: %4.\\';
    Text002: Label 'VarYesNo = \#5\#\#\#\#\#. The return code is: %6.\\';
    Text003: Label 'VarDuration = %7. The return code is: %8.';
begin
    Value := '19960101';
    Ok1 := Evaluate(VarInteger, Value);
    Ok2 := Evaluate(VarDate, Value);
    Ok3 := Evaluate(VarYesNo, Value);
    Value := '2days 4hours 3.7 seconds 17 milliseconds';
    Ok4 := Evaluate(VarDuration, Value);
    Message(Text000 + Text001 + Text002 + Text003, VarInteger, Ok1, VarDate, Ok2, VarYesNo, Ok3,
VarDuration, Ok4);
end;
```

The message window displays the following:

VarInteger = 10196 . The return code is: Yes.

VarDate = 01/01/96. The return code is: Yes.

VarYesNo = No . The return code is: No.

VarDuration = 2 days 4 hours 3 seconds 717 milliseconds. The return code is: Yes.

This example shows that although Value ('010196') can be interpreted as both an integer and a date expression, it cannot be interpreted as a Boolean expression. This causes an error, shown in the return code Ok3 (=False).

This example also shows that when you evaluate a string as a duration data type, you can use certain words in the string to describe the duration. The following words or abbreviations are supported:

- day, days, d
- hour, hours, h
- minute, minutes, min, m
- second, seconds, sec, s
- millisecond, milliseconds, milli, millis

You can include decimal values in the string that you evaluate as a duration, except for milliseconds, which must be a whole number.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ExportEncryptionKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a password protected temporary filepath containing the encryption key. When encrypting or decrypting data in Dynamics 365 Business Central, an encryption key is used. A single key is used per tenant and every tenant will have a different key. Keys can be exported to a file which may be necessary in the case of upgrading or migrating a system from one set of hardware to another. The EXPORTEncryptIONKey method allows an administrator to specify a destination file for the key and specify a password protection for the file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Path :=   System.ExportEncryptionKey(Password: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Password
Type: String
Specifies the password for the encryption key file.

Return Value

Path Type: String A temporary filepath to where the key is exported.

Remarks

> **NOTE**
> This applies only to on-premises versions of Business Central. For online versions, encryption is always enabled and you cannot turn it off.

If encryption is not enabled or the encryption key is not found, the following error will be displayed: **An encryption key is required to complete the request**.

Example

This code example uses the ExportEncryptionKey method to return a password protected file that contains an

encryption key. With the Download method the file is sent from the Dynamics 365 Business Central service computer to the client computer.

This example requires that you create the following text constants: ExportFileName and ClientFileName.

```
if not EncryptionEnabled then
    Error('Encryption has not been enabled.');
    ExportFileName := ExportEncryptionKey('This is my personal secret');

    ClientFileName := 'ExportedKey.ekey';
    Download(ExportFileName,'Save the encrypted key file','','Encrypted Key File
(*.ekey)|*.ekey',ClientFileName);

    Erase(ExportFileName);
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ExportObjects Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Exports application objects to a file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
System.ExportObjects(FileName: String, var ObjectRecord: Record [, Format: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

FileName
Type: String
The path of the file to export to.

ObjectRecord
Type: Record
A record to the Object table.

Format
Type: Integer
The format to use when exporting.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Format Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Formats a value into a string.

Syntax

```
String :=   System.Format(Value: Any [, Length: Integer] [, FormatNumber: Integer])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Value

Type: Any

This is an AL variable (expression) of any simple data type, such as Option, Integer, BigInteger, Decimal, Char, Text, Code, Date, Time, DateTime, Boolean, or GUID. If, when the system formats Value, the result is a value larger than the maximum length MaxStrLen method (Code, Text) of String, a run-time error occurs.

Length

Type: Integer

This optional parameter specifies the length of String.

FormatNumber

Type: Integer

This optional parameter specifies the format that you want to use.

Return Value

String Type: String

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Format Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Formats a value into a string.

Syntax

```
String :=   System.Format(Value: Any, Length: Integer, FormatString: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Value

Type: Any

This is an AL variable (expression) of any simple data type, such as Option, Integer, BigInteger, Decimal, Char, Text, Code, Date, Time, DateTime, Boolean, or GUID. If, when the system formats Value, the result is a value larger than the maximum length MaxStrLen method (Code, Text) of String, a run-time error occurs.

Length

Type: Integer

This optional parameter specifies the length of String.

FormatString

Type: String

A literal string that defines a format as in the Format Property.

Return Value

String Type: String

Remarks

For the *Length* parameter, the following rules apply:

- If *Length* = 0 then the entire value is returned (default).
- If *Length* > 0 then the returned string will be exactly *Length* characters.

  If *Value* is less than *Length* characters, then either leading or trailing spaces are inserted, depending on the format that you select.

  If *Value* exceeds *Length* characters, then *String* is truncated to *Length* characters.
- If *Length* < 0 then the returned string will not have leading or trailing spaces.

  If *Value* is less than *Length* characters, the length of *String* will equal the length of *Value*.

If *Value* exceeds *Length* characters, then *String* is truncated to *Length* characters.

For the *Format* parameter, see Format Property for more information.

The *FormatNumber* parameter specifies the format that you want to use. The basic options for the Decimal data type are as follows:

- <*Sign*><*Integer Thousand*><*Decimals*> is Format 0
- <*Sign*><*Integer*><*Decimals*> is Format 1
- <*Sign*><*Integer*><*Decimals*><*Comma,.*> is Format 2
- <*Integer Thousand*><*Decimals*><*Sign,1*> is Format 3
- <*Integer*><*Decimals*><*Sign,1*> is Format 4

> **NOTE**
> You cannot use both *FormatNumber* and *FormatStr* at the same time.

Example 1

```
var
    Text000: Label 'The formatted value >%1<';
begin
    Message(Text000, Format(-123456.78, 12, 3));
    Message(Text000, Format(-123456.78, 12, '<Standard Format,3>'));
    Message(Text000, Format(-123456.78, 12, '<Integer Thousand><Decimals><Sign,1>'));
end;
```

The Regional and Language settings on the computer on which you run the code affect how the string is displayed. For example, on a computer that has the regional format set to English (United States), the message window displays the following:

The formatted value: > 123,456.78-<

The formatted value: > 123,456.78-<

The formatted value: > 123,456.78-<

On a computer that has the regional format set to Danish (Denmark), the message window displays the following:

The formatted value: > 123.456,78-<

The formatted value: > 123.456,78-<

The formatted value: > 123.456,78-<

Example 2

This example shows how to use a string to build a format.

```
var
    Text000: Label 'Today is %1';
begin
    Message(Text000, Format(Today,0,'<Month Text> <Day>'));
end;
```

The message window displays the following:

Today is April 15.

See Also

Format Property
System Data Type
Getting Started with AL
Developing Extensions

System.GetDocumentUrl Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the URL for the specified temporary media object ID.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Url :=   System.GetDocumentUrl(ID: Guid)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

ID

Type: Guid

The temporary media object ID.

Return Value

Url Type: String The URL for the specified temporary media object ID.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetDotNetType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the System.Type that corresponds to the given value.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Type :=   System.GetDotNetType(Expression: Any)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Expression
Type: Any
The value for which to retrieve the System.Type.

Return Value

Type Type: DotNet

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetLastErrorCallStack Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the call stack from where the last error occurred.

Syntax

```
String :=   System.GetLastErrorCallStack()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

String Type: String

Remarks

For some errors, such as divide by zero errors and overflow errors, **GetLastErrorCallStack** does not return every call in the stack. To get the complete call stack for these types of errors, use the debugger and specify that you want to break on errors. On the **Debugger** page, in the **Call Stack** FactBox, you can view all the method calls that led to the error.

Example

In this example, an error occurs in codeunit 50003. The text of the Message includes a call to the GetLastErrorCallStack method.

```
// Codeunit 50001, TestErrors1
// OnRun trigger
Error('Some error message')

// Codeunit 50002, TestErrors2
// OnRun trigger
ClearLastError;
if not Codeunit.Run(50001) then
  Message('The call stack for the last error is:\' + GetLastErrorCallStack);
```

When you run codeunit 50002, the message window displays the following:

The call stack for the last error is:

TestErrors1(CodeUnit 50001).OnRun(Trigger) line 1

TestErrors2(CodeUnit 50002).OnRun(Trigger) line 2

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetLastErrorCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the classification of the last error that occurred.

Syntax

```
String :=   System.GetLastErrorCode()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

String Type: String

Remarks

You use the GetLastErrorCode method to identify the type of the last error that occurred. You use the GetLastErrorText Method to get the details of the last error.

The result of the GetLastErrorCode method is not translated into the local language. The result of the GetLastErrorText Method is translated into the local language.

Example

This example opens a file and then causes an error to occur by trying to open the file again. If the code for the error that occurred is not the expected error code, then an error message is displayed.

```
var
    ErrorCode: Text[1024];
    ExpectedErrorCode: Text[1024];
    FileObj1: File;
    FileObj2: File;
    Text0001: Label 'The error code is not valid. Expected: %1. Actual: %2. Error message: %3';
begin
    ExpectedErrorCode := 'StreamIO';
    FileObj1.CreateTempFile;
    AssertError FileObj2.Create(FileObj1.Name);
    ErrorCode := GetLastErrorCode;
    if ErrorCode <> ExpectedErrorCode then
      Error(Text001, ExpectedErrorCode, ErrorCode, GetLastErrorText);
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetLastErrorObject Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the last System.Exception object that occurred.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
DotNet :=   System.GetLastErrorObject()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

DotNet Type: DotNet

Remarks

You use this method to retrieve and handle the last exception that occurred in the application. The System.Exception object exposes several members that enable you to get detailed information about the exception, such Exception.InnerException and Exception.Message.

Example

This example uses the GetLastErrorObject method to get an exception object that occurs. In this example, the Microsoft .NET Framework objects are executed by MyCodeunit. The AL code uses the InnerException property of the System.Exception object to identify whether the inner exception has the type WebException and returns an exception message accordingly.

```
var
    MyCodeunit: Codeunit MyCodeunit;
    Exception: DotNet "'System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089'.System.Net.WebException";
    WebException: DotNet "'mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089'.System.Exception";
begin
    if not MyCodeunit.Run then begin
        Exception := GetLastErrorObject;

        if not Exception.InnerException.GetType.Equals(WebException.GetType) then
            Error(Exception.Message);

        WebException := Exception.InnerException;
        Error(WebException.Message);
    end;
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetLastErrorText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the last error that occurred in the debugger.

Syntax

```
String :=   System.GetLastErrorText()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

String Type: String

Remarks

If you call the GetLastErrorText method immediately after you call the ClearLastError method, then an empty string is returned.

The result of the GetLastErrorCode Method is not translated into the local language. The result of the GetLastErrorText method is translated into the local language.

Example

If you call the Codeunit.Run method to run a codeunit and an error occurs in the codeunit, then the error is displayed. However, if you also use the return value of the Codeunit.Run method, then the error is not displayed. In this case, you can use the GetLastErrorText method to determine whether an error has occurred and to see the text of the last error message that was generated. This example shows how to use the GetLastErrorText method. This example requires that you create two codeunits. Codeunit 50001 generates an error. Codeunit 50002 runs codeunit 50001 and if an error occurs, then it displays the text of the error.

```
// Codeunit 50001
// OnRun trigger
Error('Some error message.');

// Codeunit 50002
// OnRun trigger
ClearLastError();
if not Codeunit.Run(50001) then
  Message('The last error was: ' + GetLastErrorText);
```

In this example, because the if statement uses the return value of the Codeunit.Run method, the error from codeunit 50001 is not displayed. Instead, you use the GetLastErrorText method to display the error.

When you run codeunit 50002, the message window displays the following:

The last error was: Some error message.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetUrl Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Generates a URL for the specified client target that is based on the configuration of the server instance. If the code runs in a multitenant deployment architecture, the generated URL will automatically apply to the tenant ID of the current user.

Syntax

```
String :=   System.GetUrl(ClientType: ClientType [, Company: String] [, ObjectType: ObjectType] [, ObjectId:
Integer] [, Record: Record] [, UseFilters: Boolean])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

ClientType

Type: ClientType

Specifies the client that you want to generate the URL for. If you want to generate a URL that depends on the client that the user is accessing the URL from, choose Current. A runtime error occurs if the ClientType is set to SOAP or OData but the specified object type and ID has not been published as a web service.

Company

Type: String

Specifies the company that the URL must contain. If you do not specify a company, the URL will run in the user's current company.

ObjectType

Type: ObjectType

Value: Table, Page, Report, Codeunit, Query, or XmlPort. Specifies the object type that the URL must open. If you specify an object type, you must also specify an object ID in the ObjectId parameter. Otherwise, the user will see a runtime error. If you set the ObjectType parameter to Page, you can also specify a record variable in the Record parameter.

ObjectId

Type: Integer

Specifies the ID of the specified object type that the URL must open.

Record

Type: Record

Specifies the Record variable that specifies which record to open.

UseFilters

Type: Boolean

Specifies whether to include filters that are defined on the object as a text string in the URL.

Return Value

String Type: String

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GetUrl Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Generates a URL for the specified client target that is based on the configuration of the server instance. If the code runs in a multitenant deployment architecture, the generated URL will automatically apply to the tenant ID of the current user.

Syntax

```
String :=   System.GetUrl(ClientType: ClientType, Company: String, ObjectType: ObjectType, ObjectId:
Integer, RecordRef: RecordRef [, UseFilters: Boolean])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

ClientType

Type: ClientType

Specifies the client that you want to generate the URL for. This parameter is required. If you want to generate a URL that depends on the client that the user is accessing the URL from, choose Current. The following table describes the options.> A runtime error occurs if the ClientType is set to SOAP or OData but the specified object type and ID has not been published as a web service.

Company

Type: String

Specifies the company that the URL must contain. If you do not specify a company, the URL will run in the user's current company.

ObjectType

Type: ObjectType

Value: Table, Page, Report, Codeunit, Query, or XmlPort. Specifies the object type that the URL must open. If you specify an object type, you must also specify an object ID in the ObjectId parameter. Otherwise, the user will see a runtime error. If you set the ObjectType parameter to Page, you can also specify a record variable in the Record parameter.

ObjectId

Type: Integer

Specifies the ID of the specified object type that the URL must open.

RecordRef

Type: RecordRef

Specifies the RecordRef variable that specifies which record to open.

UseFilters

Type: Boolean

Specifies whether to include filters that are defined on the object as a text string in the URL.

Return Value

String Type: String

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GlobalLanguage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the current global language setting.

Syntax

```
[LanguageID := ]  System.GlobalLanguage([NewLanguageID: Integer])
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Parameters

NewLanguageID

Type: Integer

The Microsoft language ID (LCID), such as 1033 for English (US).

Return Value

LanguageID Type: Integer The Microsoft language ID (LCID).

Remarks

The LanguageID is a standard Windows language ID. The Windows Language virtual table contains a list of these IDs and the corresponding names and short names.

For more information, see Multilanguage Development.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.GuiAllowed Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether the AL code can show any information on the screen.

Syntax

```
Ok :=   System.GuiAllowed()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

Ok Type: Boolean

Remarks

User Portal Application Server accepts GuiAllowed.

Example

This example shows how to use the GuiAllowed method.

```
var
    Text000: Label 'Code is running on a client.';
begin
    if GuiAllowed then
     Message(Text000);
end;
```

If the code runs on a client, which means that the user interface is available, a message box will appear with the following message.

Code is running on a client

If the code runs on Microsoft Dynamics NAV Application Server, then the message will not be displayed.

NOTE

If the Message Method (Dialog) or the Error Method (Dialog) is called when the code is running on Microsoft Dynamics NAV Application Server, then the message is written to the event log of the operating system.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Hyperlink Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Passes a URL as an argument to an Internet browser, such as Windows Internet Explorer.

Syntax

```
System.Hyperlink(URL: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

URL
Type: String
A URL that is passed to the Internet browser as an argument.

Remarks

The syntax must be a valid URL, such as *https://www.microsoft.com*, or path to a file on file share, such as *file://d$/myfiles/myfile.txt*. If you want the URL to be configurable, you can get the URL from a field or a variable instead. If you pass an empty string, then no browser window is opened.

At runtime, a new tab is opened in the same browser window where Dynamics 365 Business Central is running.

The `HyperLink` method works with different protocols and file types as along as the syntax is valid.

> **NOTE**
> When using the *file://* protocol to open a file, the file should be stored on a network file share, not locally; otherwise the file will not open. Browsers block hyperlinks to files from a web page for security reasons and the hyperlink must be manually copied and pasted into a manually opened tab page.

The HyperLink method does not work on NAS services.

Example

The following example shows two uses of the HyperLink method to open the specified URL in the default browser. In the first line of code, the URL is specified in code. The second line of code illustrates how you can get a URL that is stored in a field on the current table.

```
HyperLink('https://www.microsoft.com');
...
HyperLink(Rec.UrlField);
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ImportEncryptionKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Points to a password protected file that contains the key on the current server. When encrypting or decrypting data in Dynamics 365 Business Central, an encryption key is used. A single key is used per tenant, and every tenant will have a different key. Keys can be created or imported if one exists already, as may be the case if upgrading or migrating a system from one set of hardware to another. The IMPORTEncryptIONKey method allows an administrator to specify a file (password protected) which contains a key and imports it to the current Dynamics 365 Business Central service.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
[Ok := ]  System.ImportEncryptionKey(Path: String, Password: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Path
Type: String
Specifies the file that contains the encryption key.
Password
Type: String
Specifies the password the protects the file.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

If the import key file cannot be imported, the following error will be displayed: **Import Failed. The provided encryption key file contains invalid data and could not be imported**.

Example

This code example uses the EncryptionEnabled and EncryptionExists methods to do a check before importing the encryption key.

This example requires that you create a text constant ServerFileName.

```
if EncryptionEnabled then
        if EncryptionExists then
          if not Confirm('Encryption has been enabled and the server already contains an encryption key.\'
          + 'Importing a key will overwrite any existing key and may result in lost data.\\'
          + 'Do you wish to continue?') then
            exit
       else
          if not Confirm('Importing a key different from the already configured key will result in data
corruption.\\'
          + 'Do you wish to continue?') then
            exit

      if not Upload('Upload encrypted key','','Encrypted Key File
(*.ekey)|*.ekey','ExportedKey.ekey',ServerFileName) then
        Exit;

      ImportEncryptionKey(ServerFileName,'This is my personal secret');
      Erase(ServerFileName);
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ImportObjects Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Imports application objects from a file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
System.ImportObjects(FileName: String [, Format: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

FileName

Type: String

The path of the file from which the objects will be imported.

Format

Type: Integer

The format in which the objects are represented in the file.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.ImportStreamWithUrlAccess Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Imports an object into a media container to be used in a temporary URL with a default expiration time.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
ID :=   System.ImportStreamWithUrlAccess(InStream: InStream, Filename: String [, MinutesToExpire: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

InStream
Type: InStream
Input stream that contains the object to store as a media object.

Filename
Type: String
File name to associate with the created media object.

MinutesToExpire
Type: Integer
Number of minutes after which the object will expire.

Return Value

ID Type: Guid The ID of the media container, if the import is successful.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.IsNull Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets a value indicating whether a DotNet object has been created or not.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
Ok :=   System.IsNull(DotNet: DotNet)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

DotNet
Type: DotNet
A DotNet expression.

Return Value

Ok Type: Boolean **True** if the DotNet object is NULL, otherwise **false**.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.IsNullGuid Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether a value has been assigned to a GUID. A null GUID that consists only of zeros is valid but must never be used for references.

Syntax

```
Ok :=   System.IsNullGuid(Guid: Guid)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Guid

Type: Guid

The GUID that you want to check whether it is null.

Return Value

Ok Type: Boolean

Remarks

The Guid data type is useful when you want to uniquely identify data so that it can be exchanged with external applications. For example, if you want to transfer an item catalog to an external application, you add a Guid field to the record in the table and use it as the primary reference when you communicate with the external application. A Guid is a 16-byte binary data type that can be logically grouped into the following subgroups: 4byte-2byte-2byte-2byte-6byte.

Example

The following example initializes two variables named validGuid and nullGuid. The validGuid variable is assigned a valid Guid value, and the nullGuid variable is assigned a null GUID that consists of only zeros. The IsNullGuid method determines whether the Guid that is contained in the *validGuid* parameter is null. In this case, the Guid is not null so a message that states that the Guid is not null is displayed and the value is displayed. The method then checks the *nullGuid* parameter. This time, a message that states that the Guid is null is displayed because the Guid is a null Guid that consists of only zeros.

```
var
    validGuid: Guid;
    nullGuid: Guid;
    Text000: Label 'The Guid is null';
    Text0001: Label 'The Guid is not null.\\';
    Text0002: Label 'The value is %1.';
begin
    validGuid := '{FC841BE6-0ADA-46C4-A6A9-142AEC211613}';
    nullGuid  := '{00000000-0000-0000-0000-000000000000}';
    if IsNullGuid(validGuid) then
      Message(Text000)
    else
      Message(Text001 + Text002, validGuid);
    if IsNullGuid(nullGuid) then
      Message(Text000);
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.IsServiceTier Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Gets a value indicating whether the runtime is a service tier.

Syntax

```
Ok :=   System.IsServiceTier()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

Ok Type: Boolean **True** if the runtime is a service tier, otherwise **false**.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.NormalDate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the regular date (instead of the closing date) for the argument Date.

Syntax

```
NormalDate :=   System.NormalDate(Date: Date)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Date

Type: Date

The input date. You can enter a closing date or a normal date. A run-time error occurs if the value of Date is set to the undefined date (0D).

Return Value

NormalDate Type: Date

Example 1

The input date is a regular date.

```
var
    InputDate: Date;
    OutputDate: Date;
    Text000: Label 'The normal date for %1 is %2.';
begin
    InputDate := 20140404D;
    OutputDate := NormalDate(InputDate);
    Message(Text000, InputDate, OutputDate);
end;
```

On a computer that has the regional format set to English (United States), the message window displays the following:

The normal date for 04/04/14 is 04/04/14.

Example 2

The input date is a closing date.

```
var
    InputDate: Date;
    OutputDate: Date;
    Text000: Label 'The normal date for %1 is %2.';
begin
    InputDate := ClosingDate(20140404C);
    OutputDate := NormalDate(InputDate);
    Message(Text001, InputDate, OutputDate);
end;
```

On a computer that has the regional format set to English (United States), the message window displays the following:

The normal date for C04/04/14 is 04/04/14.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Power Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Raises a number to a power. For example, you can use this method to square the number 2 to get the result of 4.

Syntax

```
NewNumber :=   System.Power(Number: Decimal, Power: Decimal)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Number

Type: Decimal

The number you want to raise exponentially. This number is the base in the exponential method.

Power

Type: Decimal

The exponent in the exponential method.

Return Value

NewNumber Type: Decimal

Example 1

```
var
    Number1: Decimal;
    Power1: Decimal;
    Result1: Decimal;
    Text000: Label '%1 raised to the power of %2 = %3';
begin
    Number1 := 64;
    Power1 := 0.5;
    Result1 := Power(Number1, Power1);
    Message(Text000, Number1, Power1, Result1);
end;
```

On a computer that has the regional format set to English (United States), the first message window displays the following:

64 raised to the power of 0.5 = 8

This example shows that raising a number to the power of 0.5 corresponds to the square root of the number.

Example 2

This example shows a typical use for the POWER method.

If a principal amount P is deposited at interest rate R and compounded annually, then at the end of N years, the accumulated amount (A) is:

$A = P(1 + R)^N$

For example, you put LCY 2800 into a bank account that pays 5 percent, which is compounded quarterly. To determine what the amount will be in eight years, you must consider:

N = 32 payment periods (8 years times 4 quarterly periods)

R = 0.0125 per period (5 percent divided by 4 quarterly periods)

The accumulated amount A is:

$A = LCY\ 2800(1 + 0.0125)^{32} = LCY\ 2800(1.4881) = LCY\ 4166.77$

If a principal amount P is deposited at the end of each year at interest rate R (in decimal notation) compounded annually, then at the end of N years, the accumulated amount is:

$A = P[((1 + R)^N - 1)/R]$

This is typically called an *annuity*.

For example, you have an annuity in which a payment of LCY 500 is made at the end of each year. The interest on this annuity is 4 percent, which is compounded annually. To determine what the annuity will be worth in 20 years, you must consider:

R = 0.04

N = 20

The amount of the annuity A will be:

$A = LCY\ 500[((1 + 0.04)^{20} - 1)/0.04 = LCY\ 14,889.04$

```
var
    P: Decimal;
    R: Decimal;
    N: Decimal;
    A: Decimal;
    FormatString: Text;
    Text000: Label 'Principal $%1 at a 5 percent interest rate is compounded quarterly.\\';
    Text001: Label '(Rate = %2)\\';
    Text002: Label 'The amount after %3 years = $%4.';
    Text003: Label 'Principal $%1 is deposited at the end of each year at a 4 percent interest rate,
compounded annually.\\';
    Text004: Label '(Rate = %2)\\';
    Text005: Label 'The amount after %3 years = $%4.';
begin
    FormatString := '<Precision,2><Standard Format,1>';
    // Example 1
    P := 2800;
    R := 0.0125;
    N := 32;
    A = P * (Power(1 + R, N));
    Message(Text000 + Text001 + Text002, P, R, N, Format(A,0,FormatString);
    // Example 2
    P = 500;
    R = 0.04;
    N = 20;
    A = P * ((Power(1 + R, N) - 1)/R);
    Message(Text001, P, R, N, Format(A,0,FormatString));
end;
```

On a computer that has the regional format set to English (United States), the first message window displays the following:

Principal $2,800 at a 5 percent interest rate is compounded quarterly.

(Rate = 0.0125)

The amount after 32 years = $4166.77.

The second message window displays the following:

Principal $500 is deposited at the end of each year at a 4 percent interest rate, compounded annually.

(Rate = 0.04)

The amount after 20 years = $14889.04.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Random Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a pseudo-random number.

Syntax

```
Number :=   System.Random(MaxNumber: Integer)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

MaxNumber

Type: Integer

The largest acceptable number. In effect, you are setting a range from one (1) to the number that you specify with the MaxNumber parameter.

Return Value

Number Type: Integer

Remarks

If *MaxNumber* is negative it acts as a positive.

If *MaxNumber* is zero, this method always returns 1.

A number is always chosen from the same set of numbers. Use Randomize Method (Integer) to generate a new set of numbers.

Example

This example shows how to generate a pseudo-random number. The value of the variable Number2 is positive though the value of *MaxNumber* is negative and the value of the variable Number3 is always 1 because *MaxNumber* is 0.

```
var
    x: Integer;
    y: Integer;
    z: Integer;
    Number1: Integer;
    Number2: Integer;
    Number3: Integer;
    Text000: Label 'Number1 = %1, Number2 = %2, Number3 = %3';
begin
    x := 100; // x is assigned a positive value.
    y := -100; // y is assigned a negative value.
    z := 0; // z is assigned zero.
    Number1 := Random(x);
    Number2 := Random(y);
    Number3 := Random(z);
    Message(Text000, Number1, Number2, Number3);
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Randomize Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Generates a set of random numbers from which the RANDOM method (Integer) will select a random number.

Syntax

```
System.Randomize([Seed: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Seed

Type: Integer

A number used to create a unique set of numbers.

Remarks

If you use the same number as *Seed*, the same set of numbers is generated. If you omit this optional parameter, **Randomize** uses the current system time (total number of milliseconds since midnight). Calling the **Randomize** method before the Random Method makes the random numbers more unpredictable.

Furthermore, the random generator is specific to each connection so the sequence of numbers that is returned when you call the Random Method will be the same after each call to **Randomize** with a specific seed.

Example

The following example generates random numbers between 1 and 5 by using the *Seed* from the **Randomize** method to initialize the random number generator in the Random Method. The **Randomize** method uses the data from system clock as the *Seed* value.

```
var
    x: Integer;
    Text000: Label 'X=%1';
begin
    Randomize();
    x := Random(5);
    Message(Text000, x);
end;
```

See Also

System Data Type
Getting Started with AL

System.Round Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Rounds the value of a numeric variable.

Syntax

```
NewNumber :=   System.Round(Number: Decimal [, Precision: Decimal] [, Direction: String])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Number

Type: Decimal

The number that you want to round.

Precision

Type: Decimal

This optional parameter determines the precision used when rounding. If you do not specify a Precision parameter, then the following steps are used to specify the precision:

1. The method ReadRounding in Codeunit 45, ReadRounding, is called. ReadRounding returns a decimal value that is the precision. By default, the ReadRounding method returns the Amount Rounding Precision field from the GLSetup table.
2. If you have customized Codeunit 45 and it does not implement the ReadRounding method, then the precision is specified as 2 digits after the decimal.

Direction

Type: String

This optional parameter specifies how to round the Number parameter. The default rounding method is '='. The following are the options for rounding:

- '=' rounds up or down to the nearest value (default). Values of 5 or greater are rounded up. Values less than 5 are rounded down.
- '>' rounds up
- '<' rounds down

Return Value

NewNumber Type: Decimal

Example

This example shows how to use the **Round** method.

```
var
    DecimalToRound: Decimal;
    Direction: Text;
    Precision: Decimal;
    Result: Decimal;
    Text00: Label 'Round(%1, %2, %3) returns %4.';
begin
    DecimalToRound := 1234.56789;
    Direction := '>';
    Precision := 0.001;
    Result := Round(DecimalToRound, Precision, Direction);
    Message(Text000, Format(DecimalToRound,0,1), Precision, Direction, Result);
end;
```

On a computer that has the regional format set to English (United States), the message window displays the following:

Round(1234.56789, 0.001, '>') returns 1,234.568

The following table displays some additional Round examples.

NUMBER	PRECISION	DIRECTION	ROUNDED NUMBER
1234.56789	100	=	1200
1234.56789	0.1	=	1234.6
1234.56789	0.001	=	1234.568
1234.56789	0.001	<	1234.567
1234.56789	0.001	>	1234.568
-1234.56789	100	=	-1200
-1234.56789	0.1	=	-1234.6
-1234.56789	0.001	=	-1234.568
-1234.56789	0.001	<	-1234.567
-1234.56789	0.001	>	-1234.568

When you round down ('<') a negative number, such as -1234.56789, it is rounded down to -1234.567. However, -1234.567 is a mathematically greater value than -1234.56789.

When you round up ('>') a negative number, such as -1234.56789, it is rounded up to -1234.568. However, -1234.568 is a mathematically smaller value than -1234.56789.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.RoundDateTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Rounds a DateTime.

Syntax

```
NewDatetime :=   System.RoundDateTime(Datetime: DateTime [, Precision: BigInteger] [, Direction: String])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Datetime

Type: DateTime

The DateTime that you want to round.

Precision

Type: BigInteger

This optional parameter determines the precision used when rounding. The default value is 1000, which rounds to the nearest second. You can only use positive BigIntegers.

Direction

Type: String

This optional parameter specifies how to round the DateTime. The default rounding method is '='. You can change the method by using the following options:

- '=' rounds up or down to the nearest value (default). Values of 5 or greater are rounded up. Values less than 5 are rounded down.
- '>' rounds up
- '<' rounds down

Return Value

NewDatetime Type: DateTime The rounded result.

Remarks

A variable of type DateTime is stored as a 64 bit integer. The integer value is the number of milliseconds since year 0. When you call RoundDateTime, the DateTime integer value is rounded as a numeric variable.

If you use 1 for the *Precision* parameter, then the resulting DateTime is rounded to the nearest millisecond. If you use 1000 for the *Precision* parameter, which is the default value, then the resulting DateTime will be rounded to the nearest second.

We recommend that you do not use a value greater than 60*60*1000, which is the number of milliseconds in an

hour, for the *Precision* parameter. The Regional and Language Options in Windows affect how the hour and date parts of a DateTime are rounded. To display a DateTime in a specific format, we recommend that you use the Format Method (Code, Text) instead of the RoundDateTime method.

Example

This example shows how to use the RoundDateTime method to round to the nearest second.

```
var
    Text000: Label 'RoundDateTime(%1, %2) returns %3.';
begin
    DateToRound := 20081127D;
    TimeToRound := 093524.567T;
    DateTimeToRound := CreateDateTime(DateToRound, TimeToRound);
    Precision := 1000L;
    FormatString := '<Month,2>/<Day,2>/<Year> <Hours24,2>:<Minutes,2>:<Seconds,2>.<Thousands,3>';
    Result := RoundDateTime(DateTimeToRound, Precision);
    Message(TEXT000, Format(DateTimeToRound,0,FormatString), Precision, Format(Result,0,FormatString));
end;
```

The message window displays the following:

RoundDateTime(11/27/08 09:35:24.567, 1000) returns 11/27/08 09:35:25.000.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Sleep Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns control to the operating system for a specified time.

Syntax

```
System.Sleep(Duration: Integer)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Duration

Type: Integer

The number of milliseconds to return control to the operating system.

Remarks

When you use the Sleep method, control is guaranteed to return to the operating system for at least *Duration* milliseconds.

> **NOTE**
> The period may be longer than *Duration* milliseconds, depending on what the operating system is doing at the time that control is to return to the caller.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.TemporaryPath Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the path of the directory where the temporary file is stored.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
String :=   System.TemporaryPath()
```

> **NOTE**
> This method can be invoked using property access syntax.

> **NOTE**
> This method can be invoked without specifying the data type name.

Return Value

String Type: String

Remarks

This method returns a string that contains the path of the directory where the temporary file for Dynamics 365 is stored. This string ends with a backslash ('\') and does not contain the name of the temp file.

The string cannot contain more than 255 characters.

If this method is called from an application that is running on a Dynamics 365 Application Server, it returns the path of the directory where the Dynamics 365 Application Server is installed.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Time Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current time from the operating system.

Syntax

```
Time :=   System.Time()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

Time Type: Time

Remarks

The time that is returned is different for the Business Central Web client and Dynamics NAV Client connected to Business Central. For the Web client, the time is determined by the Regional Setting that is set under **MySetting** in the client. The very first time a user signs in, the system automatically determines the Regional Setting based on the user's browser/computer. For the Windows client, Time returns the current day of the computer that is running the client, as determined by the date and time settings of the operating system.

You can only use the Time method to retrieve the time from the operating system. You cannot use it to set the time in the operating system.

Example

```
var
    Text000: Label 'The current system time is %1.";
begin
    Message(Text000, Time);
end;
```

On a computer that has the regional format set to English (United States), the message window could display the following:

The current system time is 11:15:46 AM.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Today Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current date set in the operating system.

Syntax

```
Date :=   System.Today()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

Date Type: Date

Remarks

The date that is returned is different for the Business Central Web client and Dynamics NAV Client connected to Business Central. For the Web client, the day is determined by the Regional Setting that is set under **MySetting** in the client. The very first time a user signs in, the system automatically determines the Regional Setting based on the user's browser/computer. For the Windows client, Today returns the current day of the computer that is running the client, as determined by the date and time settings of the operating system.

You can only use the **Today** method to retrieve the current date from the operating system. You cannot use it to set the date in the operating system.

Example

This example shows how to use the **Today** method.

```
var
    Text000: Label 'The current date is: %1';
begin
    Message(Text000, Today);
end;
```

The message window could display the following:

The current date is: 05/27/08

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Variant2Date Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a date from a variant.

Syntax

```
Date :=   System.Variant2Date(Variant: Variant)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Variant
Type: Variant
The input variant.

Return Value

Date Type: Date

Example

```
var
    TextDate: Date;
    variant1: Variant;
begin
    variant1 := Today;
    TestDate := Variant2Date(variant1);
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.Variant2Time Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a time from a variant.

Syntax

```
Time :=   System.Variant2Time(Variant: Variant)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Variant
Type: Variant
The input variant.

Return Value

Time Type: Time

Example

```
var
    TextTime: Time;
    variant1: Variant;
begin
    variant1 := Time;
    TestTime := Variant2Time(variant1);
end;
```

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.WindowsLanguage Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current Windows language setting.

Syntax

```
LanguageID :=   System.WindowsLanguage()
```

NOTE

This method can be invoked using property access syntax.

NOTE

This method can be invoked without specifying the data type name.

Return Value

LanguageID Type: Integer

Remarks

The *LanguageID* is a standard Windows language ID. The Windows Language virtual table contains a list of these IDs and the corresponding names and short names.

For more information, see Multilanguage Development.

See Also

System Data Type
Getting Started with AL
Developing Extensions

System.WorkDate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the work date for the current session.

Syntax

```
[WorkDate := ]  System.WorkDate([NewDate: Date])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

NewDate

Type: Date

The new work date you want to set.

Return Value

WorkDate Type: Date

Remarks

To set the work date to follow the calendar day so that the work date is always the current date, set *NewDate* to `Today` or `0D`. If you explicitly set *NewDate* to the current date, then the work date will also follow the calendar day.

Example

This example shows how to use the WorkDate method.

```
var
    NewDate: Date;
    Text000: Label 'The new work date is: %1';
begin
    NewDate := WorkDate(20180101D);
    Message(Text000, NewDate);
end;
```

The code sets the work date to January 1, 2018, and returns the new date in a message. On a computer that has the regional format set to English (United States), the message window displays the following:

The work date is: 01/01/18

See Also

System Data Type
Getting Started with AL
Developing Extensions

Record Data Type

2/17/2021 • 7 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a complex data type.

The following methods are available on instances of the Record data type.

METHOD NAME	DESCRIPTION
AddLink(String [, String])	Adds a link to a record.
AddLoadFields([Any,...])	Specifies fields to be initially loaded when the record is retrieved from its data source. Subsequent calls to AddLoadFields will not overwrite fields already selected for the initial load.
AreFieldsLoaded(Any,...)	Checks whether the specified fields are all initially loaded.
Ascending([Boolean])	Gets or sets the order in which the system searches through a table.
CalcFields(Any [, Any,...])	Calculates the FlowFields in a record. You specify which fields to calculate by using parameters.
CalcSums(Any [, Any,...])	Calculates the total of a column in a table. You specify which fields to calculate by using parameters.
ChangeCompany([String])	Redirects references to table data from one company to another.
ClearMarks()	Removes all the marks from a record.
Consistent(Boolean)	Marks a table as being consistent or inconsistent.
Copy(var Record [, Boolean])	Copies a specified record's filters, views, automatically calculated FlowFields, marks, fields, and keys that are associated with the record from a table or creates a reference to a record.
CopyFilter(Any, Any)	Copies the filter that has been set for one field and applies it to another field.
CopyFilters(var Record)	Copies all the filters set by the SetFilter method (Record) or the SETRANGE method (Record) from one record to another.
CopyLinks(var Record)	Copies all the links from a specified record.
CopyLinks(RecordRef)	Copies all the links from a specified record.
Count()	Counts the number of records in a table.
CountApprox()	Returns an approximate count of the number of records in the table, for example, for updating progress bars or displaying informational messages.
CurrentCompany()	Gets the current company of a database table record.
CurrentKey()	Gets the current key of a database table.
Delete([Boolean])	Deletes a record in a table.
DeleteAll([Boolean])	Deletes all records in a table that fall within a specified range.
DeleteLink(Integer)	Deletes a specified link from a record in a table.
DeleteLinks()	Deletes all of the links that have been added to a record.

METHOD NAME	DESCRIPTION
FieldActive(Any)	Checks whether a field is enabled.
FieldCaption(Any)	Gets the current caption of the specified field as a string.
FieldError(Any [, String])	Stops the execution of the code causing a run-time error, and creates an error message for a field.
FieldName(Any)	Gets the name of a field as a string.
FieldNo(Any)	Gets the number assigned to a field in the table description.
FilterGroup([Integer])	Gets or sets the filter group that is applied to a table.
Find([String])	Finds a record in a table that is based on the values stored in keys.
FindFirst()	Finds the first record in a table based on the current key and filter.
FindLast()	Finds the last record in a table based on the current key and filter.
FindSet([Boolean] [, Boolean])	Finds a set of records in a table based on the current key and filter.
Get([Any,...])	Gets a record based on values stored in primary key fields.
GetAscending(Any)	Gets the sort order for the records returned. You can use GetASCENDING to identify the sort order of the specified field because fields can be sorted in ascending or descending order. For example, you can read data from an ODATA web service where the data is sorted in ascending order on the Name field but in descending order on the City field.
GetBySystemId(Guid)	Gets a record by its SystemId.
GetFilter(Any)	Gets a list of the filters within the current filter group that are applied to a field.
GetFilters()	Gets a string that contains a list of the filters within the current filter group for all fields in a record. In addition, this method also returns the state of the MARKEDONLY method (Record).
GetPosition([Boolean])	Gets a string that contains the primary key of the current record.
GetRangeMax(Any)	Gets the maximum value in a range for a field.
GetRangeMin(Any)	Gets the minimum value in a range for a field.
GetView([Boolean])	Gets a string that describes the current sort order, key, and filters on a table.
HasFilter()	Determines whether a filter is attached to a record within the current filter group.
HasLinks()	Determines whether a record contains any links.
Init()	Initializes a record in a table.
Insert()	Inserts a record into a table without executing the code in the OnInsert trigger.
Insert(Boolean)	Inserts a record into a table.
Insert(Boolean, Boolean)	Inserts a record into a table.
IsEmpty()	Determines whether a table or a filtered set of records is empty.

METHOD NAME	DESCRIPTION
IsTemporary()	Determines whether a record refers to a temporary table.
LoadFields(Any,...)	Accesses the table's corresponding data source and loads the values of the specified fields on the record.
LockTable([Boolean] [, Boolean])	Locks a table to protect it from write transactions that conflict with each other.
Mark([Boolean])	Marks a record. You can also use this method to determine whether a record is marked.
MarkedOnly([Boolean])	Activates a special filter. After you use this function, your view of the table includes only records marked by this function.
Modify([Boolean])	Modifies a record in a table.
ModifyAll(Any, Any [, Boolean])	Modifies a field in all records within a range that you specify.
Next([Integer])	Steps through a specified number of records and retrieves a record.
ReadConsistency()	Determines if the table supports read consistency.
ReadPermission()	Determines whether a user is granted read permission to the table that contains a record. This method can test for both full read permission and partial read permission that has been granted with a security filter.
RecordId()	
RecordLevelLocking()	Determines whether the table supports record-level locking.
Relation(Any)	Determines the table relationship of a given field.
Rename(Any [, Any,...])	Changes the value of a primary key in a table.
Reset()	Removes all filters, including any special filters set by MarkedOnly, changes fields select for loading back to all, and changes the current key to the primary key. Also removes any marks on the record and clears any AL variables defined on its table definition.
SecurityFiltering([SecurityFilter])	
SetAscending(Any, Boolean)	Sets the sort order for the records returned. Use this method after you have set the keys to sort after, using SetCurrentKey. The default sort order is ascending. You can use SetAscending to change the sort order to descending for a specific field, while the other fields in the specified key are sorted in ascending order.
SetAutoCalcFields([Any,...])	Sets the FlowFields that you specify to be automatically calculated when the record is retrieved from the database.
SetCurrentKey(Any [, Any,...])	Selects a key for a table.
SetFilter(Any, String [, Any,...])	Assigns a filter to a field that you specify.
SetLoadFields([Any,...])	Sets the fields to be initially loaded when the record is retrieved from its data source. This will overwrite fields previously selected for initial load.
SetPermissionFilter()	Applies the user's security filter.
SetPosition(String)	Sets the fields in a primary key on a record to the values specified in the supplied string. The remaining fields are not changed.
SetRange(Any [, Any] [, Any])	Sets a simple filter, such as a single range or a single value, on a field.

METHOD NAME	DESCRIPTION
SetRecFilter()	Sets the values in the current key of the current record as a record filter.
SetView(String)	Sets the current sort order, key, and filters on a table.
TableCaption()	Gets the current caption of a table as a string.
TableName()	Gets the name of a table.
TestField(Any)	Tests whether the contents of a field match a given value.
TestField(Any, Boolean)	Tests whether the contents of a field match a given value.
TestField(Any, Integer)	Tests whether the contents of a field match a given value.
TestField(Any, BigInteger)	Tests whether the contents of a field match a given value.
TestField(Any, Decimal)	Tests whether the contents of a field match a given value.
TestField(Any, Guid)	Tests whether the contents of a field match a given value.
TestField(Any, Text)	Tests whether the contents of a field match a given value.
TestField(Any, Label)	Tests whether the contents of a field match a given value.
TestField(Any, TextConst)	Tests whether the contents of a field match a given value.
TestField(Any, Code)	Tests whether the contents of a field match a given value.
TestField(Any, String)	Tests whether the contents of a field match a given value.
TestField(Any, Enum)	Tests whether the contents of a field match a given value.
TestField(Any, Any)	Tests whether the contents of a field match a given value.
TransferFields(var Record [, Boolean])	Copies all matching fields in one record to another record.
TransferFields(var Record, Boolean, Boolean)	Copies all matching fields in one record to another record.
Validate(Any [, Any])	Calls the OnValidate trigger for the field that you specify.
WritePermission()	Determines whether a user can write to a table. This method can test for both full write permission and partial write permission that has been granted with a security filter. A write permission consists of Insert, Delete, and Modify permissions.

See Also

Getting Started with AL
Developing Extensions

Record.AddLink Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a link to a record.

Syntax

```
[ID := ]  Record.AddLink(URL: String [, Description: String])
```

Parameters

Record Type: Record An instance of the Record data type.

URL
Type: String

Description
Type: String

Return Value

ID Type: Integer

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.AddLoadFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Specifies fields to be initially loaded when the record is retrieved from its data source. Subsequent calls to AddLoadFields will not overwrite fields already selected for the initial load.

Syntax

```
[Ok := ]  Record.AddLoadFields([Fields: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Fields

Type: Any

The FieldNo's of the fields to be loaded.

Return Value

Ok Type: Boolean **true** if all fields are selected for subsequent loads; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

It is not necessary to include the following fields, because they are always selected for loading: Primary key, SystemId, and data audit fields (SystemCreatedAt, SystemCreatedBy, SystemModifiedAt, SystemModifiedBy).

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

Example

The following example code shows how to use the AddLoadFields method to add a field for loading on a report. The example loads a field that is outside of the default fields selected by DataColumns of the report. The field is added by using the AddLoadFields on the OnPreDataItem trigger.

```
trigger OnPreDataItem()
begin
    CurrencyDataItem.AddLoadFields(CurrencyDataItem."ISO Numeric Code");
end;

trigger OnAfterGetRecord()
begin
    if (CurrencyDataItem."ISO Numeric Code" <> 'DKK') then begin
        CurrReport.Skip();
    end;
end;
```

See Also

Using Partial Records
Record Data Type
Getting Started with AL
Developing Extensions

Record.AreFieldsLoaded Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Checks whether the specified fields are all initially loaded.

Syntax

```
Ok :=   Record.AreFieldsLoaded(Fields: Any,...)
```

Parameters

Record Type: Record An instance of the Record data type.

Fields
Type: Any
The FieldNo's of the fields to check.

Return Value

Ok Type: Boolean **true** if all the fields specified by the Fields parameter are currently loaded; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

Example

This example shows how to use the AreFieldsLoaded method when you only need to load either the **Name** or the **Name 2** field on the **Customer** table. The procedure selects which ever field is actually loaded. If neither is loaded, this causes a JIT load.

```
procedureGetLoadedName(Cust:RecordCustomer):Text
begin
ifCust.AreFieldsLoaded(Cust.Name)then
exit(Cust.Name)
else
ifCust.AreFieldsLoaded(Cust."Name 2")then
exit(Cust."Name 2")
else
exit(Cust.Name);
end;
```

See Also

Using Partial Records
Record Data Type

Record.Ascending Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the order in which the system searches through a table.

Syntax

```
[Ascending := ]  Record.Ascending([Ascending: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Ascending
Type: Boolean

Return Value

Ascending Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CalcFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates the FlowFields in a record. You specify which fields to calculate by using parameters.

Syntax

```
[Ok := ]  Record.CalcFields(Field1: Any [, Field2: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Field1
Type: Any

Field2
Type: Any

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

FlowFields are virtual fields. The values in these fields are not saved in the table. This means that you must use either the **CalcFields** method or the SetAutoCalcFields Method (Record) to update them. For example, if you retrieve a record using the Find Method (Record) and Next Method (Record), the FlowFields in those records are set to zero (0). Then, when you call **CalcFields**, their values are updated.

When a FlowField is a direct source expression on a page or a report, the calculation is performed automatically.

You can also use the **CalcFields** method to retrieve binary large objects (BLOBs). For more information, see BLOB Data Type.

Dynamics 365 Business Central automatically maintains a count for all SIFT indexes. Therefore, SumIndexField Technology (SIFT) is used by default when the calculation method for a FlowField is **Count** or **Average**.

You can prevent the SIFT indexes from being updated by setting the MaintainSIFTIndex Property of the index in the base table to **False**. Then Business Central will traverse all records in the base table to perform the calculation instead of using SIFT. This can reduce the number of required SIFT indexes, which can improve performance. For more information, see SIFT and Performance.

In Dynamics 365 Business Central, an index is not required to support a certain sorting, but sorting without an index could lead to bad performance if a search returns a large result set, which would then have to be sorted before the first row is returned.

Example

This example shows how to use the **CalcFields** method to find the balance on December 31, 2008 and the net change for a customer in 2008.

```
var
    CustomerRec: Record Customer;
begin
    CustomerRec.SetRange("Date Filter",20080101D,20081231D);
    CustomerRec.CalcFields(Balance, "Net Change");
end;
```

The first line sets up a filter for the Date Filter field in the Customer record. This field is a FlowFilter field which is used in the filter definition for several FlowFields in the Customer record. In the second line, the FlowFields are calculated.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.CalcSums Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates the total of a column in a table. You specify which fields to calculate by using parameters.

Syntax

```
[Ok := ]  Record.CalcSums(Field1: Any [, Field2: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Field1
Type: Any

Field2
Type: Any

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
CALCSUM Method (FieldRef)
AL Database Methods and Performance on SQL Server

Record.ChangeCompany Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Redirects references to table data from one company to another.

Syntax

```
[Ok := ]  Record.ChangeCompany([CompanyName: String])
```

Parameters

Record Type: Record An instance of the Record data type.

CompanyName
Type: String
The name of the company to which you want to change. If you omit this parameter, you change back to the current company.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.ClearMarks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all the marks from a record.

Syntax

```
Record.ClearMarks()
```

Parameters

Record Type: Record An instance of the Record data type.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Consistent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Marks a table as being consistent or inconsistent.

Syntax

```
Record.Consistent(Consistent: Boolean)
```

Parameters

Record Type: Record An instance of the Record data type.

Consistent

Type: Boolean

The mark to be set on the table. If this parameter is true, the table is marked as consistent. If this parameter is false, the table is marked as inconsistent.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Copy Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies a specified record's filters, views, automatically calculated FlowFields, marks, fields, and keys that are associated with the record from a table or creates a reference to a record.

Syntax

```
Record.Copy(var FromRecord: Record [, ShareTable: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

FromRecord

Type: Record

The record to copy.

ShareTable

Type: Boolean

Specifies whether the method copies filters, views, automatically calculated FlowFields, marks, fields, and keys of the record or creates a reference to a temporary record. If FromRecord and Record are both temporary and ShareTable is true, then the COPY method causes Record to reference the same table as FromRecord. If ShareTable is true, then both Record and FromRecord must be temporary; otherwise an error will occur. The default value is false. If you specify false, only filters, marks, and keys are copied.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CopyFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies the filter that has been set for one field and applies it to another field.

Syntax

```
Record.CopyFilter(FromField: Any, Record.ToField: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

FromField
Type: Any
The field from which the filter will be copied.

Record.ToField
Type: Any

Remarks

The filter is copied and remains in the assigned group number. For example: `Rec.CopyFilter(FromRec);` disregards the current filter group on both `Rec` and `FromRec`, and copies the filter in `FromRec` (regardless of group number) into the same filter group assignment on `Rec`.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CopyFilters Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies all the filters set by the SetFilter method (Record) or the SETRANGE method (Record) from one record to another.

Syntax

```
Record.CopyFilters(var FromRecord: Record)
```

Parameters

Record Type: Record An instance of the Record data type.

FromRecord
Type: Record
The record from which you want to copy filters.

Remarks

All filters are copied and remain in their assigned group numbers. For example: `Rec.CopyFilters(FromRec);` disregards the current filter group on both `Rec` and `FromRec`, and copies all filters in `FromRec` (regardless of group number) into the same filter group assignments on `Rec`.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CopyLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies all the links from a specified record.

Syntax

```
Record.CopyLinks(var FromRecord: Record)
```

Parameters

Record Type: Record An instance of the Record data type.

FromRecord

Type: Record

Specifies the record from which you want to copy links.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CopyLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Copies all the links from a specified record.

Syntax

```
Record.CopyLinks(FromRecordRef: RecordRef)
```

Parameters

Record Type: Record An instance of the Record data type.

FromRecordRef
Type: RecordRef
Specifies the record from which you want to copy links.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Counts the number of records in a table.

Syntax

```
Number :=   Record.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Number Type: Integer

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CountApprox Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns an approximate count of the number of records in the table, for example, for updating progress bars or displaying informational messages.

Syntax

```
Number :=   Record.CountApprox()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Number Type: Integer

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CurrentCompany Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current company of a database table record.

Syntax

```
Company :=   Record.CurrentCompany()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Company Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.CurrentKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current key of a database table.

Syntax

```
CurrentKey :=   Record.CurrentKey()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

CurrentKey Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Delete Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes a record in a table.

Syntax

```
[Ok := ]  Record.Delete([RunTrigger: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

RunTrigger
Type: Boolean
Specifies whether to run the AL code in the OnDelete Trigger. If this parameter is true, then the code in the OnDelete trigger is executed. If this parameter is false, then the code in the OnDelete trigger is not executed. The default value is false. This parameter is optional.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.DeleteAll Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes all records in a table that fall within a specified range.

Syntax

```
Record.DeleteAll([RunTrigger: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

RunTrigger
Type: Boolean
Specifies whether to run the AL code in the OnDelete Trigger. If this parameter is true, then the code in the OnDelete trigger will be executed. If this parameter is false, then the code in the OnDelete trigger will not be executed. The default value is false.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.DeleteLink Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes a specified link from a record in a table.

Syntax

```
Record.DeleteLink(ID: Integer)
```

Parameters

Record Type: Record An instance of the Record data type.

ID

Type: Integer

The ID of the link to delete.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.DeleteLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes all of the links that have been added to a record.

Syntax

```
Record.DeleteLinks()
```

Parameters

Record Type: Record An instance of the Record data type.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FieldActive Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether a field is enabled.

Syntax

```
Ok :=   Record.FieldActive(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to check.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FieldCaption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current caption of the specified field as a string.

Syntax

```
Caption :=   Record.FieldCaption(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field for which you want to retrieve the caption.

Return Value

Caption Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FieldError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stops the execution of the code causing a run-time error, and creates an error message for a field.

Syntax

```
Record.FieldError(Field: Any [, Text: String])
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field for which you want to create an error message.

Text

Type: String

Use this optional parameter to include the text of the error message. If you do not use this parameter, then default text is used as shown in the following examples. You can use backslashes (\) to break lines.

Remarks

Like a run-time error, this method causes the system to automatically end any transaction.

Programming guidelines

The following guidelines for error messages are recommended:

- Describe what is wrong and how to solve the problem.
- Write a short descriptive message. Do not use more words than necessary.
- Note that a period is automatically inserted at the end of a FieldError.
- Use a label data type for the *Text* parameter.

For more information, see Progress Windows, Message, Error, and Confirm Methods.

Example 1

In the first example, there is no *Text* parameter and the field does not have a value.

```
var
    CustomerRec: Record Customer;

...
CustomerRec."No." := '';
CustomerRec.FieldError("No.");
```

The following message is displayed:

You must specify No. in Customer No.=''.

Example 2

In the next example, there is no *Text* parameter and the field has a value.

```
var
    CustomerRec: Record Customer;

...
CustomerRec."No." := 'NEW 3500';
CustomerRec.FieldError("No.");
```

The following message is displayed:

No. must not be NEW 3500 in Customer No.='NEW 3500'.

Example 3

The third example uses a non-empty string as the *Text* parameter.

```
var
    CustomerRec: Record Customer;
    Text001: Label 'is not valid';

...
CustomerRec."No." := 'NEW 3500';
CustomerRec.FieldError("No.", Text001);
```

The following message is displayed:

No. is not valid in Customer No.='NEW 3500'.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FieldName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of a field as a string.

Syntax

```
Name :=   Record.FieldName(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The name of the field in the record.

Return Value

Name Type: String

Remarks

The advantage of using the FieldName method call instead of a static assignment, like NameOfField := 'MyField', is that using the FieldName method dynamically adapts to any change to the field name made in the development environment.

Example

The following example gets the name of the **No.** field in the **Customer** table, and stores it in a string.

```
var
    NameOfField: Text;
    CustomerRec: Record Customer;

begin
    NameOfField := CustomerRec.FieldName("No.");
end;
```

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FieldNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number assigned to a field in the table description.

Syntax

```
Number :=   Record.FieldNo(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The name of the field in the record.

Return Value

Number Type: Integer

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FilterGroup Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the filter group that is applied to a table.

Syntax

```
[Group := ]  Record.FilterGroup([Group: Integer])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Group

Type: Integer

Return Value

Group Type: Integer

Remarks

A filter group can contain a filter for a Record that has been set earlier with the SetFilter Method (Record) or the SetRange Method (Record). The total filter applied is the combination of all the filters set in all the filtergroups.

When you select a filter group, subsequent filter settings by the SetFilter Method (Record) or the SetRange Method (Record) apply to that group.

All groups are active at all times. The only way to turn off a group is to remove the filters set in that group.

Filters in different groups are all effective simultaneously. For example, if in one group, a filter is set on customer numbers 1000 to 2000, while in another group, a filter is set on customer numbers 1800 to 3000, then only numbers in the range 1800 to 2000 are visible.

Dynamics 365 Business Central uses the following filter groups internally.

NUMBER	NAME	DESCRIPTION
-1	Cross-column	Used to support the cross-column search.

NUMBER	NAME	DESCRIPTION
0	Std	The default group where filters are placed when no other group has been selected explicitly. This group is used for filters that can be set from the filter dialogs by the end user.
1	Global	Used for filters that apply globally to the entire application.
2	Form	Used for the filtering actions that result from the following: - SetTableView Method (XMLport), SetTableView Method (Page) - SourceTableView Property - DataItemTableView Property.
3	Exec	Used for the filtering actions that result from the following: - SubPageView Property - RunPageView Property
4	Link	Used for the filtering actions that result from the following: - DataItemLink Property (Reports) - SubPageLink Property
5	Temp	Not currently used.
6	Security	Used for applying security filters for user permissions.
7	Factboxes	Used for clearing the state of factboxes.

A filter set in a group different from filter group 0 cannot be changed by a user that uses a filter dialog to set a filter. If, for example, a filter has been set on customer numbers 1000 to 2000 in group 4, then the user can set a filter that delimits this selection further, but cannot widen it to include customer numbers outside the range 1000 to 2000.

> **NOTE**
> It is possible to use one of the internally used groups. If you do this, you replace the filter that Dynamics 365 Business Central assumes is in this group. If, for example, you use filter group 4 in a page, you will replace the filtering that is actually the result of applying the SubPageLink Property. This could seriously alter the way pages and subpages interact.

> **IMPORTANT**
> Using filter group 7 may cause factboxes to not work as intended.

Reset filter

To reset the filters in filter group 1, you add an empty filter to the group. To add an empty filter, to filter group 1, you must first set the filter group.

```
Rec.FilterGroup(1);
```

Then, for each field in the table that to which the Rec variable refers, set an empty filter.

```
Rec.SetFilter(<field>,'');
```

Example 1

The following example uses the SetFilter Method (Record) to set a filter that selects records with No. field between 10000 and 20000. Then the **FilterGroup** method returns the number for the filter group. No filter group was selected explicitly so the filter is set in filter group 0. This value is stored in the `varOrigGroup` variable and displayed in a message box. Next, the **FilterGroup** method changes the filter group to 100. The new value is stored in the `varCurrGroup` variable and displayed in a message box.

```
var
        Customer: Record Customer;
        varOrigGroup: Integer;
        varCurrGroup: Integer;
        Text000: Label 'The original filtergroup is: %1';
        Text001: Label 'The current filtergroup is: %1';
begin
    Customer.SetFilter("No.", '10000..20000');
    varOrigGroup := Customer.FilterGroup;
    Message(Text000, varOrigGroup);
    varCurrGroup := Customer.FilterGroup(1);
    Message(Text001, varCurrGroup);
end;
```

Example 2

The following example finds all customers where the Customer Name or Contact Name contains the string **John**.

```
var
    SearchString: Text;
begin
    Customer.FilterGroup := -1;
    SearchString := '@*John*';
    Customer.SetFilter(Customer.Name, SearchString);
    Customer.SetFilter(Customer.Contact, SearchString);
end;
```

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Find Method

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds a record in a table that is based on the values stored in keys.

Syntax

```
[Ok := ]  Record.Find([Which: String])
```

Parameters

Record Type: Record An instance of the Record data type.

Which
Type: String
Specifies how to perform the search. The table is searched until either a record is found or there are no more records. Each character in this string can be present only one time. You can combine the '=', '<', and '>' characters. You can use the following characters:

- = to search for a record that equals the key values (default)
- > to search for a record that is larger than the key values
- < to search for a record that is less than the key values
- + to search for the last record in the table (+ can only be used alone)
- - to search for the first record in the table (- can only be used alone) If this parameter contains '=', '>' or '<', then you must assign value to all fields of the current and primary keys before you call Find.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Find retrieves the first record that meets the conditions set by *Which* and the filters associated with *Record*. The search path reflects the sort order defined by the current key. If the current key is not the primary key, several records might have the same values in current key fields. If this occurs, the sort order defined by the primary key as the search path is used.

Calling Find on an empty table from the OnNewRecord trigger causes the Business Central Server to throw an exception, and the AL execution stops. However, the client suppresses this error and does not show any error message to the user. Therefore, when using Find inside this trigger, you should add code that conditionally verifies whether a record was found, and if not, notify the user with a message. For example:

```
if not MyRecord.Find then
    Error('error message');
```

Example 1

The following example shows how use the **Find** method to find a record in a table. The code sets the number of the record to find to 1100, which is the primary key of the record to find. The **Find** method uses the '=' parameter to find the record that has a primary value that equals the specified primary key. If the record is found, then the item number, description, and unit price of the item are displayed in a message box. Otherwise, a message that specifies that the item is not found is displayed.

```
begin
    ItemRec."No." := '1100';
    if ItemRec.Find('=') then
        Message(Text000, ItemRec."No.", ItemRec.Description, ItemRec."Unit Price")
    else
        Message(Text001);
end;

var
    ItemRec: Record Item;
    Text000: Label 'Item No. %1.\Description: %2. Price: $%3.';
    Text001: Label 'The item was not found.';
```

Example 2

The following example defines a record variable that is named ItemRec. The **Find** method uses the '+' parameter to find the last record in the table. If the record is found, then the item number, description, and unit price of the item are displayed in a message box. Otherwise, the message that specifies that the item was not found is displayed.

```
begin
    if ItemRec.Find('+') then
        Message(Text000, ItemRec."No.", ItemRec.Description, ItemRec."Unit Price")
    else
        Message(Text001);
end;

var
    ItemRec: Record Item;
    Text000: Label 'Item No. %1.\Description: %2. Price: $%3.';
    Text001: Label 'The item was not found.';
```

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.FindFirst Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the first record in a table based on the current key and filter.

Syntax

```
[Ok := ]  Record.FindFirst()
```

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method should be used instead of **Find**('-') when you only need the first record.

You should only use this method when you explicitly want to find the first record in a table or set. Do not use this method in combination with **Repeat**.. **Until**.

Calling FindFirst on an empty table from the OnNewRecord trigger causes the Business Central Server to throw an exception, and the AL execution stops. However, the client suppresses this error and does not show any error message to the user. Therefore, when using FindFirst inside this trigger, you should add code that conditionally verifies whether a record was found, and if not, notify the user with a message. For example:

```
if not MyRecord.FindFirst then
    Error('error message');
```

Example

This example also assumes that you have a **ConfigurePost** method.

```
var
    SalesSetupRec: Record "Sales & Receivables Setup";
begin
    // Read the first record only.
    if SalesSetupRec.FindFirst then
        begin
           ConfigurePost(SalesSetupRec);
        end;
end;
```

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.FindLast Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the last record in a table based on the current key and filter.

Syntax

```
[Ok := ]  Record.FindLast()
```

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method should be used instead of Find('+') when you only need the last record.

You should only use this method when you explicitly want to find the last record in a table/set. Do not use this method in combination with Repeat...Until.

Calling FindLast on an empty table from the OnNewRecord trigger causes the Business Central Server to throw an exception, and the AL execution stops. However, the client suppresses this error and does not show any error message to the user. Therefore, when using FindLast inside this trigger, you should add code that conditionally verifies whether a record was found, and if not, notify the user with a message. For example:

```
if not MyRecord.FindLast then
    Error('error message');
```

Example

This example requires that you create a Record variable named GLEntryRec for the G/L Entry table.

```
// Read the last record only.
if GLEntryRec.FindLast then
  ...
```

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.FindSet Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds a set of records in a table based on the current key and filter.

Syntax

```
[Ok := ]  Record.FindSet([ForUpdate: Boolean] [, UpdateKey: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

ForUpdate
Type: Boolean
Set this parameter to true if you want to modify any records in the set; otherwise, set the parameter to false.If you set this parameter to true, then the LOCKTable method (Record) is immediately run on the table before the records are read. If you set this parameter to false, then you can still modify the records in the set, but these updates will not be performed optimally. The default value is false.

UpdateKey
Type: Boolean
Set this parameter to true if you want to modify any field value within the current key. This parameter only applies if the ForUpdate parameter is true. If you set this parameter to false, then you can still modify the records in the set, but these updates will not be performed optimally. The default value is false.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a record based on values stored in primary key fields.

Syntax

```
[Ok := ]  Record.Get([Value: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Value
Type: Any

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method always uses the primary key for the table. It ignores any filters that are set, except security filters. Security filters are applied or ignored based on the Security Filter Mode. The current key and filters are not changed after you call this method.

> **NOTE**
> `Get` does not require specifying all fields of the key in the call; any omitted field is treated as default value (for example, '' for text/code, false for boolean). You can only omit from the end of the key, not a field in the middle of a key. If a record with a blank/default value exists that is the one being returned, otherwise it will fail with a record does not exist error.

This method ignores any call to the SetAutoCalcFields Method. Therefore, a Get call on a record after a `SetAutoCalcFields` call does not automatically calculate FlowFields in the record.

> **NOTE**
> You cannot use the Get method to retrieve a record in a table by its primary key value if the primary key field in the table has the data type RecordID. In this case, you can retrieve the record by using the SetRange Method.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetAscending Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the sort order for the records returned. You can use GetASCENDING to identify the sort order of the specified field because fields can be sorted in ascending or descending order. For example, you can read data from an ODATA web service where the data is sorted in ascending order on the Name field but in descending order on the City field.

Syntax

```
IsAscending :=   Record.GetAscending(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to get the sort order for.

Return Value

IsAscending Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetBySystemId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Gets a record by its SystemId.

Syntax

```
[RecordExists := ]  Record.GetBySystemId(SystemId: Guid)
```

Parameters

Record Type: Record An instance of the Record data type.

SystemId
Type: Guid
The SystemId of the record to retrieve.

Return Value

RecordExists Type: Boolean **true** if the record exists; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Every record has a SystemId, which is stored in the SystemId field of the table. The SystemId cannot be changed.

Example

This example shows how to use the GetBySystemId method to retrieve a record.

```
var
    CustomerRec: Record Customer;
    Text000: Label 'Customer was found.';
begin
    If CustomerRec.GetBySystemId('{5286305A-08A3-E911-8180-001DD8B7338E}') then
    Message(Text000);
end;
```

See Also

SystemId Field
Record Data Type
Getting Started with AL
Developing Extensions

Record.GetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list of the filters within the current filter group that are applied to a field.

Syntax

```
String :=   Record.GetFilter(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The input field.

Return Value

String Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetFilters Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a string that contains a list of the filters within the current filter group for all fields in a record. In addition, this method also returns the state of the MARKEDONLY method (Record).

Syntax

```
String :=   Record.GetFilters()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

String Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetPosition Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a string that contains the primary key of the current record.

Syntax

```
String :=   Record.GetPosition([UseNames: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

UseNames
Type: Boolean

Return Value

String Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetRangeMax Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the maximum value in a range for a field.

Syntax

```
Value :=   Record.GetRangeMax(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field for which you want to find the maximum value. The current filter on Field must be a single range filter; otherwise, a run-time error occurs.

Return Value

Value Type: Any

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetRangeMin Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the minimum value in a range for a field.

Syntax

```
Value :=   Record.GetRangeMin(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field for which you want to find the minimum value.

Return Value

Value Type: Any

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.GetView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a string that describes the current sort order, key, and filters on a table.

Syntax

```
String :=   Record.GetView([UseNames: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

UseNames
Type: Boolean

Return Value

String Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.HasFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a filter is attached to a record within the current filter group.

Syntax

```
Ok :=   Record.HasFilter()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.HasLinks Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a record contains any links.

Syntax

```
Ok :=   Record.HasLinks()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Init Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Initializes a record in a table.

Syntax

```
Record.Init()
```

Parameters

Record Type: Record An instance of the Record data type.

Remarks

This method assigns default values to each field in the record, including the SystemId field. The values that are assigned in the record correspond to those defined when the table was created. If no value was assigned when the table was created, the values are assigned based on the data type, as shown in the following table.

DATA TYPE	DEFAULT VALUE
BigInteger	0
BigText	<Empty>
BLOB	<Empty>
Boolean	No
Code	'' (empty string)
Date	0d (Undefined date)
DateFormula	'' (empty string)
DateTime	0DT (Undefined datetime)
Decimal	0.0
Duration	0
GUID	00000000-0000-0000-0000-000000000000
Integer	0

DATA TYPE	DEFAULT VALUE
Option	0
RecordID	<Empty>
TableFilter	<Empty>
Text	'' (empty string)
Time	0T (Undefined time)

> **NOTE**
> Primary key and timestamp fields are not initialized.

After the method runs, you can change the values in any or all of the fields before you call the Insert Method (RecordRef) to enter the record in the table. Be sure that the fields that make up the primary key contain values that make the total primary key unique. If the primary key is not unique (such as the record already exists), then the record is rejected.

The method works in the same way as the Init Method (RecordRef).

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts a record into a table without executing the code in the OnInsert trigger.

Syntax

```
[Ok := ]  Record.Insert()
```

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The inserted record will automatically get assigned a SystemId by the platform. To assign a specific SystemId instead of the one assigned by the platform, use Insert(Boolean, Boolean) instead.

Example 1

This example shows how to use the Insert method without a return value.

```
Customer.Init;
Customer."No." := '1120';
Customer.Insert();
```

If customer 1120 already exists, then a run-time error occurs.

Example 2

This example shows how to use the Insert method with a return value.

```
var
    CustomerRec: Record Customer;
    Text000: Label 'Customer no: %1 inserted.';
    Text001: Label 'Customer no: %1 already exists.';
begin
    CustomerRec.Init();
    CustomerRec."No." := '1120';
    if CustomerRec.Insert() then
      Message(Text000, CustomerRec."No.")
    else
      Message(Text001, CustomerRec."No.");
end;
```

No run-time error occurs if customer 1120 already exists.

See Also

SystemId Field
Record Data Type
Getting Started with AL
Developing Extensions

Record.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts a record into a table.

Syntax

```
[Ok := ]  Record.Insert(RunTrigger: Boolean)
```

Parameters

Record Type: Record An instance of the Record data type.

RunTrigger
Type: Boolean
If this parameter is true, the code in the OnInsert Trigger is executed. If this parameter is false, the code in the OnInsert trigger is not executed. The default value is false.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Inserts a record into a table.

Syntax

```
[Ok := ]  Record.Insert(RunTrigger: Boolean, InsertWithSystemId: Boolean)
```

Parameters

Record Type: Record An instance of the Record data type.

RunTrigger
Type: Boolean
If this parameter is true, the code in the OnInsert Trigger is executed. If this parameter is false, the code in the OnInsert trigger is not executed. The default value is false.

InsertWithSystemId
Type: Boolean
If this parameter is true, the SystemId field of the record is given a value that you explicitly assign. If a value is not assigned, then the platform assigns one. If this parameter is false, the SystemId field is given a value that is auto-generated by the platform. The default value is false.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The inserted record will automatically get assigned a SystemId by the platform. To assign a specific SystemId instead of the one assigned by the platform, use Insert(Boolean, Boolean) instead.

Example

This example shows how to use the Insert method to insert a record with a specified SystemId.

```
var
    CustomerRec: Record Customer;
    Text000: Label 'Customer no: %1 inserted.';
    Text001: Label 'Customer no: %1 already exists.';
begin
    CustomerRec.Init();
    CustomerRec."No." := '1120';
    CustomerRec.SystemId := '{B6666666-F5A2-E911-8180-001DD8B7338E}';
    if CustomerRec.Insert(true, true) then
      Message(Text000, CustomerRec."No.")
    else
      Message(Text001, CustomerRec."No.");
end;
```

No run-time error occurs if customer 1120 already exists.

See Also

SystemId Field
Record Data Type
Getting Started with AL
Developing Extensions

Record.IsEmpty Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a table or a filtered set of records is empty.

Syntax

```
Empty :=   Record.IsEmpty()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Empty Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.IsTemporary Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a record refers to a temporary table.

Syntax

```
Temporary :=   Record.IsTemporary()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Temporary Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.LoadFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Accesses the table's corresponding data source and loads the values of the specified fields on the record.

Syntax

```
[Ok := ]  Record.LoadFields(Fields: Any,...)
```

Parameters

Record Type: Record An instance of the Record data type.

Fields
Type: Any
The FieldNo's of the fields to be loaded.

Return Value

Ok Type: Boolean **true** if all values were loaded on the record; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

This method will trigger a JIT load of the specified fields. The method allows for triggering the JIT load on multiple fields. If the fields are already loaded, another load will not be triggered. Using this method instead of relying on implicit JIT loads lets you develop for more explicit error handling when a load fails.

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

See Also

Using Partial Records
Record Data Type
Getting Started with AL
Developing Extensions

Record.LockTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Locks a table to protect it from write transactions that conflict with each other.

Syntax

```
Record.LockTable([Wait: Boolean] [, VersionCheck: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

Wait

Type: Boolean

Specifies what to do if the table is already locked. If this parameter is true and if another application has already locked the table, the system will wait until the table is unlocked. If this parameter is false and if another application has already locked the table, a run-time error occurs.

VersionCheck

Type: Boolean

If this parameter is true, the version of the Record will be checked. If this parameter is false, blank, or not used, the version will not be checked.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.Mark Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Marks a record. You can also use this method to determine whether a record is marked.

Syntax

```
[Marked := ]  Record.Mark([Mark: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

Mark
Type: Boolean
Specifies if a record is marked.

Return Value

Marked Type: Boolean **true** if the record is marked; otherwise, **false**.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.MarkedOnly Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Activates a special filter. After you use this function, your view of the table includes only records marked by this function.

Syntax

```
[MarkedOnly := ]  Record.MarkedOnly([MarkedOnly: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

MarkedOnly

Type: Boolean

Activates a special filter.

Return Value

MarkedOnly Type: Boolean **true** if the special filter is being used; otherwise, **false**.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Modify Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Modifies a record in a table.

Syntax

```
[Ok := ]  Record.Modify([RunTrigger: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

RunTrigger
Type: Boolean
Specifies whether to run the AL code in the OnModify Trigger. If this parameter is true, then the code in the OnModify trigger is executed. If this parameter is false (default), then the code in the OnModify trigger is not executed.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.ModifyAll Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Modifies a field in all records within a range that you specify.

Syntax

```
Record.ModifyAll(Field: Any, NewValue: Any [, RunTrigger: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to modify.

NewValue
Type: Any
The value that you want to assign to Field in all records. The data type of NewValue must match the data type of Field.

RunTrigger
Type: Boolean
If this parameter is true, the code in the OnModify Trigger is executed. If this parameter is false (default), the code in the OnModify trigger is not executed.

Remarks

If no filter is set, the field is modified in all records in the table. If a filter is set, the fields are modified only in the records which fall within the range specified by the filter. Records where the field is already equal to the new value are also updated.

The `OnValidate` field trigger is never run when `ModifyAll` is used. Using ModifyAll() is recommended if field validation is not wanted or needed. Otherwise, Record.Modify Method can be used.

Example

```
var
    customerRec: record Customer;

begin
    // The result of this statement:
    customerRec.ModifyAll("Statistics Group", 2);
    // is equivalent to:
    if customerRec.Find('-') then
        repeat
            customerRec."Statistics Group" := 2;
            customerRec.Modify;
        until customerRec.Next = 0;
end;
```

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.Next Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Steps through a specified number of records and retrieves a record.

Syntax

```
[Steps := ]  Record.Next([Steps: Integer])
```

Parameters

Record Type: Record An instance of the Record data type.

Steps
Type: Integer
Specifies the direction of the search and how many records to step over. This parameter follows the following rules:

- > 0 Search Steps records forward in the table.
- < 0 Search Steps records backward in the table.
- = 0 Stay on the same record in the table. If you do not specify this parameter, then the next record is found.

Return Value

Steps Type: Integer

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.ReadConsistency Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines if the table supports read consistency.

Syntax

```
Ok :=   Record.ReadConsistency()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.ReadPermission Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a user is granted read permission to the table that contains a record. This method can test for both full read permission and partial read permission that has been granted with a security filter.

Syntax

```
Ok :=   Record.ReadPermission()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.RecordId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Syntax

```
RecordID :=   Record.RecordId()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

RecordID Type: RecordId

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.RecordLevelLocking Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the table supports record-level locking.

Syntax

```
Ok :=   Record.RecordLevelLocking()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Relation Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines the table relationship of a given field.

Syntax

```
TableNumber :=   Record.Relation(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field for which you want to find the table relationship.

Return Value

TableNumber Type: Integer

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Rename Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Changes the value of a primary key in a table.

Syntax

```
[Ok := ]  Record.Rename(Value1: Any [, Value2: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Value1
Type: Any

Value2
Type: Any

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Reset Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all filters, including any special filters set by MarkedOnly, changes fields select for loading back to all, and changes the current key to the primary key. Also removes any marks on the record and clears any AL variables defined on its table definition.

Syntax

```
Record.Reset()
```

Parameters

Record Type: Record An instance of the Record data type.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SecurityFiltering Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Syntax

```
[SecurityFiltering := ]  Record.SecurityFiltering([SecurityFiltering: SecurityFilter])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

SecurityFiltering

Type: SecurityFilter

Return Value

SecurityFiltering Type: SecurityFilter

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SetAscending Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the sort order for the records returned. Use this method after you have set the keys to sort after, using SetCurrentKey. The default sort order is ascending. You can use SetAscending to change the sort order to descending for a specific field, while the other fields in the specified key are sorted in ascending order.

Syntax

```
Record.SetAscending(Field: Any, Ascending: Boolean)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to set the sort order for.

Ascending

Type: Boolean

The sort order. Specify false if data must be sorted in descending order; otherwise true.

Remarks

SetAscending is applicable to records that aren't displayed in a page in the client. For example, you can read data from an ODATA web service where data is sorted in ascending order on one field and descending on another. When the records are shown in a page, the method has no effect.

Example

The following code example shows how to use SetCurrentKey and SetAscending to sort data in two different directions based on two fields. It uses SetCurrentKey to specify the sort based on City and Name. Data will be sorted in ascending order based on those two fields, first by City, then by Name. Next, you use SetAscending to sort City in descending order instead.

```
page 50100 MyCustomerList
{
    PageType = List;
    ApplicationArea = All;
    UsageCategory = Lists;
    SourceTable = Customer;

    layout
    {
        area(Content)
        {
            repeater(GroupName)
            {
                field("City"; City)
                {
                    ApplicationArea = All;
                }
                field(Name; Name)
                {
                    ApplicationArea = All;
                }
            }
        }

    }

    trigger OnOpenPage()
    begin
        SetCurrentKey(City, Name);
        SetAscending(Name, false);
    end;

}
```

If you publish the page as a web service, and read the OData feed, you'll see the records sorted in ascending alphabetical order by city and descending alphabetical order by name. However, if you open the page in the client the city and name are both sorted in ascending order.

See Also

SetCurrentKey Method Record Data Type
Getting Started with AL
Developing Extensions

Record.SetAutoCalcFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the FlowFields that you specify to be automatically calculated when the record is retrieved from the database.

Syntax

```
[Ok := ]  Record.SetAutoCalcFields([Field1: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Field1

Type: Any

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

See AL Database Methods and Performance on SQL Server - SetAutoCalcFields.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
AL Database Methods and Performance on SQL Server

Record.SetCurrentKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a key for a table.

Syntax

```
[Ok := ]  Record.SetCurrentKey(Field1: Any [, Field2: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Field1
Type: Any

Field2
Type: Any

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

You can use SetCurrentKey for selecting a key and sorting When you use SetCurrentKey the following rules apply:

- Inactive fields are ignored. Only active keys are scanned.
- When searching for a key, the first occurrence of the specified fields is selected. This means the following:
  - If you specify only one field as a parameter when you call SetCurrentKey, the key that is actually selected may consist of more than one field.
  - If the field that you specify is the first component of several keys, the key that is selected may not be the key that you expect.
  - If no keys can be found that include the fields that you specify, the return value is **false**. If you do not test the return value, a run-time error occurs. If you do test the return value, the program will continue to run even though no key was found.

See Also

Record Data Type
Getting Started with AL
Developing Extensions
SetCurrentKey, SetRange, SetFilter, GetRangeMin, and GetRangeMax Methods

Record.SetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Assigns a filter to a field that you specify.

Syntax

```
Record.SetFilter(Field: Any, String: String [, Value: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to filter.

String

Type: String

The filter expression. A valid expression consists of alphanumeric characters and one or more of the following operators: <, >, \, &, |, and =. You can use replacement fields (%1, %2, and so on) to insert values at run-time.

Value

Type: Any

Replacement values to insert in replacement fields in the filter expression. The data type of Value must match the data type of Field.

Remarks

`SetFilter` does not filter for empty values. For example, if you set `MyRecord.SetFilter(MyTextField, '');` no filter is applied.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SetLoadFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 6.0.*

Sets the fields to be initially loaded when the record is retrieved from its data source. This will overwrite fields previously selected for initial load.

Syntax

```
[Ok := ]  Record.SetLoadFields([Fields: Any,...])
```

Parameters

Record Type: Record An instance of the Record data type.

Fields

Type: Any

The FieldNo's of the fields to be loaded.

Return Value

Ok Type: Boolean **true** if all fields are selected for subsequent loads; otherwise, **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

Calling SetLoadFields on a record without passing any fields will reset the fields selected to load to the default, where all readable normal fields are selected for load.

It is not necessary to include the following fields, because they are always selected for loading: Primary key, SystemId, and data audit fields (SystemCreatedAt, SystemCreatedBy, SystemModifiedAt, SystemModifiedBy).

This method is part of the partial records capability for improving performance. For more information, see Using Partial Records.

Example

This example uses the SetLoadFields method to speedup the calculation of the mean for values of the **Standard Cost** field in the **Item** table. Instead of loading all fields, only the **Standard Cost** is loaded. The other fields aren't needed for the calculation, so they're not loaded.

```
procedure ComputeArithmeticMean(): Decimal;
var
    Item: Record Item;
    SumTotal: Decimal;
    Counter: Integer;
begin
    Item.SetLoadFields(Item."Standard Cost");
    if Item.FindSet() then begin
        repeat
            SumTotal += Item."Standard Cost";
            Counter += 1;
        until Item.Next() = 0;
        exit(SumTotal / Counter);
    end;
end;
```

See Also

Using Partial Records
Record Data Type
Getting Started with AL
Developing Extensions

Record.SetPermissionFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies the user's security filter.

Syntax

```
Record.SetPermissionFilter()
```

Parameters

Record Type: Record An instance of the Record data type.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SetPosition Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the fields in a primary key on a record to the values specified in the supplied string. The remaining fields are not changed.

Syntax

```
Record.SetPosition(String: String)
```

Parameters

Record Type: Record An instance of the Record data type.

String

Type: String

The string that is used to set the primary key. This string contains the primary key value to set.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SetRange Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a simple filter, such as a single range or a single value, on a field.

Syntax

```
Record.SetRange(Field: Any [, FromValue: Any] [, ToValue: Any])
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to filter.

FromValue

Type: Any

The lower limit of the range. The data type of this parameter must match the data type of Field.

ToValue

Type: Any

The upper limit of the range. If you omit this parameter, then the value that you specified for FromValue is used. The data type of this parameter must match the data type of Field.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SetRecFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the values in the current key of the current record as a record filter.

Syntax

```
Record.SetRecFilter()
```

Parameters

Record Type: Record An instance of the Record data type.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.SetView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current sort order, key, and filters on a table.

Syntax

```
Record.SetView(String: String)
```

Parameters

Record Type: Record An instance of the Record data type.

String
Type: String
A string that contains the sort order, key, and filter to set. The string format is the same as the SourceTableView Property on pages.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TableCaption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current caption of a table as a string.

Syntax

```
Caption :=   Record.TableCaption()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Caption Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TableName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name of a table.

Syntax

```
Name :=   Record.TableName()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Name Type: String

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to test.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Boolean)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to test.

Value

Type: Boolean

The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Integer)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: Integer
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: BigInteger)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: BigInteger
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Decimal)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: Decimal
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Guid)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: Guid
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Text)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.
Value
Type: Text
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Label)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.
Value
Type: Label
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: TextConst)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.
Value
Type: TextConst
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Code)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: Code
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: String)
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

The field that you want to test.

Value

Type: String

The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Enum)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: Enum
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TestField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Tests whether the contents of a field match a given value.

Syntax

```
Record.TestField(Field: Any, Value: Any)
```

Parameters

Record Type: Record An instance of the Record data type.

Field
Type: Any
The field that you want to test.

Value
Type: Any
The value that you want to compare to Field. The data type of this parameter must match the data type of Field. If you include this optional parameter and the contents of Field do not match, then an error message is displayed. If you omit this parameter and the contents of Field is zero or blank (empty string), then an error message is displayed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TransferFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies all matching fields in one record to another record.

Syntax

```
Record.TransferFields(var FromRecord: Record [, InitPrimaryKeyFields: Boolean])
```

Parameters

Record Type: Record An instance of the Record data type.

FromRecord
Type: Record
The record from which to copy.

InitPrimaryKeyFields
Type: Boolean
Default: true If this parameter is true and the records are in the same table, both the timestamp and the Primary Key fields of the destination record will be changed. If this parameter is true and the records are not in the same table, then the Primary Key fields of the destination record will be changed but the timestamp of the destination record will not be changed. If this parameter is false, then neither the timestamp nor the Primary Key fields of the destination record are changed.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.TransferFields Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.2.*

Copies all matching fields in one record to another record.

Syntax

```
Record.TransferFields(var FromRecord: Record, InitPrimaryKeyFields: Boolean, SkipFieldsNotMatchingType:
Boolean)
```

Parameters

Record Type: Record An instance of the Record data type.

FromRecord
Type: Record
The record from which to copy.

InitPrimaryKeyFields
Type: Boolean
Default: true If this parameter is true and the records are in the same table, both the timestamp and the Primary Key fields of the destination record will be changed. If this parameter is true and the records are not in the same table, then the Primary Key fields of the destination record will be changed but the timestamp of the destination record will not be changed. If this parameter is false, then neither the timestamp nor the Primary Key fields of the destination record are changed.

SkipFieldsNotMatchingType
Type: Boolean
Specifies whether fields where the field type on the source record do not match the field type on the target record should be ignored.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.Validate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calls the OnValidate trigger for the field that you specify.

Syntax

```
Record.Validate(Field: Any [, NewValue: Any])
```

Parameters

Record Type: Record An instance of the Record data type.

Field

Type: Any

A field together with associated triggers.

NewValue

Type: Any

The value to insert into Field.

See Also

Record Data Type
Getting Started with AL
Developing Extensions

Record.WritePermission Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether a user can write to a table. This method can test for both full write permission and partial write permission that has been granted with a security filter. A write permission consists of Insert, Delete, and Modify permissions.

Syntax

```
Ok :=   Record.WritePermission()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Record Type: Record An instance of the Record data type.

Return Value

Ok Type: Boolean

See Also

Record Data Type
Getting Started with AL
Developing Extensions

TaskScheduler Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Is a complex data type for creating and managing tasks in the task scheduler, which runs codeunits at scheduled times.

The following methods are available on the TaskScheduler data type.

METHOD NAME	DESCRIPTION
CancelTask(Guid)	Cancels and deletes a scheduled task that runs a specific codeunit.
CanCreateTask()	Checks whether it is possible to schedule tasks in this session.
CreateTask(Integer, Integer [, Boolean] [, String] [, DateTime] [, RecordId])	Adds a task to ensure that a codeunit is not run before the specified time.
SetTaskReady(Guid [, DateTime])	Sets a task that runs a codeunit to the ready state. The task will not run unless it is in the ready state.
TaskExists(Guid)	Checks whether a specific task exists.

See Also

Getting Started with AL
Developing Extensions

TaskScheduler.CancelTask Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Cancels and deletes a scheduled task that runs a specific codeunit.

Syntax

```
[Ok := ]  TaskScheduler.CancelTask(Task: Guid)
```

Parameters

Task

Type: Guid

Specifies the unique identifier of the task. The unique identifier is returned by the CreateTASK method.

Return Value

Ok Type: Boolean

Remarks

Scheduled tasks are recorded in table **2000000175 Scheduled Task**. CancelTask removes the task entry from the table.

CancelTask can only cancel pending tasks. It cannot cancel a task that is in process. To see an example of CancelTask in use, refer to AL code of table **472 Job Queue Entry**.

For more information about tasks and TaskScheduler data type methods, see managing tasks Task Scheduler.

See Also

TaskScheduler Data Type
Getting Started with AL
Developing Extensions

TaskScheduler.CanCreateTask Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether it is possible to schedule tasks in this session.

Syntax

```
Ok :=   TaskScheduler.CanCreateTask()
```

Return Value

Ok Type: Boolean

See Also

TaskScheduler Data Type
Getting Started with AL
Developing Extensions

TaskScheduler.CreateTask Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds a task to ensure that a codeunit is not run before the specified time.

Syntax

```
[Task := ]  TaskScheduler.CreateTask(CodeunitId: Integer, FailureCodeunitId: Integer [, IsReady: Boolean] [,
Company: String] [, NotBefore: DateTime] [, RecordID: RecordId])
```

Parameters

CodeunitId

Type: Integer

Specifies the ID of the codeunit to run.

FailureCodeunitId

Type: Integer

Specifies the ID of the codeunit to run if the task fails. If you do not want to provide a failure codeunit, then use 0.

IsReady

Type: Boolean

Sets the task to the ready state. A task cannot run unless it is ready.

Company

Type: String

Specifies the company to run the task for. If you do not specify a company, the task will run in the user's current company.

NotBefore

Type: DateTime

Specifies the date and time that you want to run the task. When the task actually runs will depend on whether other tasks are running at the same time. The task will run the first opportunity on or after the date and time that you specify.

RecordID

Type: RecordId

Specifies the recordID of the record that you want to run the task on.

Return Value

Task Type: Guid

Remarks

Scheduled tasks are recorded in table **2000000175 Scheduled Task**. For more information about tasks and task scheduler, see managing tasks Task Scheduler.

Example

The following example schedules a task to run the **Job Queue Dispatcher** and uses codeunit **Job Queue Error Handler** as the failure codeunit.

```
var
    JobQueueEntry: Record "Job Queue Entry";
begin
    TaskScheduler.CreateTASK(CodeUnit::"Job Queue Dispatcher", CodeUnit::"Job Queue Error Handler", True,
CompanyName, CurrentDateTime + 1000 + Random(3000), JobQueueEntry.RecordID);
end;
```

See Also

TaskScheduler Data Type
Getting Started with AL
Developing Extensions

TaskScheduler.SetTaskReady Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a task that runs a codeunit to the ready state. The task will not run unless it is in the ready state.

Syntax

```
[Ok := ]  TaskScheduler.SetTaskReady(Task: Guid [, NotBefore: DateTime])
```

Parameters

Task
Type: Guid

NotBefore
Type: DateTime

Return Value

Ok Type: Boolean

Remarks

For more information about tasks and **TaskScheduler** data type methods, see managing tasks Task Scheduler.

Example

The following example creates a task, and then uses the SetTaskReady method to set the task to ready.

```
var
    TaskID: GUID;
begin
    TaskID := TaskScheduler.CreateTASK(CodeUnit::"Job Queue Dispatcher", CodeUnit::"Job Queue Error
Handler");
    TaskScheduler.SetTaskReady(taskID);
end;
```

See Also

TaskScheduler Data Type
Getting Started with AL
Developing Extensions

TaskScheduler.TaskExists Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Checks whether a specific task exists.

Syntax

```
Exists :=   TaskScheduler.TaskExists(Task: Guid)
```

Parameters

Task

Type: Guid

The unique identifier of the task. The unique identifier is returned by the CreateTASK method.

Return Value

Exists Type: Boolean

Remarks

Scheduled tasks are recorded in table **2000000175 Scheduled Task**. To see an example of TaskExists in use, refer to AL code of table **472 Job Queue Entry**.

For more information about tasks and TaskScheduler data type methods, see managing tasks Task Scheduler.

See Also

TaskScheduler Data Type
Getting Started with AL
Developing Extensions

TestAction Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a test action on a page.

The following methods are available on instances of the TestAction data type.

METHOD NAME	DESCRIPTION
Enabled()	Enables an action on a test page.
Invoke()	Invokes an action on a test page.
Visible()	Sets whether to display the action on a test page.

See Also

Getting Started with AL
Developing Extensions

TestAction.Enabled Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Enables an action on a test page.

Syntax

```
Enabled :=   TestAction.Enabled()
```

Parameters

TestAction Type: TestAction An instance of the TestAction data type.

Return Value

Enabled Type: Boolean

See Also

TestAction Data Type
Getting Started with AL
Developing Extensions

TestAction.Invoke Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Invokes an action on a test page.

Syntax

```
TestAction.Invoke()
```

Parameters

TestAction Type: TestAction An instance of the TestAction data type.

Remarks

All actions that are available on the page are also available on the test page.

See Also

TestAction Data Type
Getting Started with AL
Developing Extensions

TestAction.Visible Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets whether to display the action on a test page.

Syntax

```
Visible :=   TestAction.Visible()
```

Parameters

TestAction Type: TestAction An instance of the TestAction data type.

Return Value

Visible Type: Boolean

See Also

TestAction Data Type
Getting Started with AL
Developing Extensions

TestField Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a testable field on a page.

The following methods are available on instances of the TestField data type.

METHOD NAME	DESCRIPTION
Activate()	Activates a field on a test page.
AsBoolean()	Converts the value in a field on a test page to a Boolean data type.
AsDate()	Converts the value in a field on a test page to a Date data type.
AsDateTime()	Converts the value in a field on a test page to a DateTime data type.
AsDecimal()	Converts the value in a field on a test page to a Date data type.
AsInteger()	Converts the value of the field on a test page to an Integer data type.
AssertEquals(Any)	Asserts that the value in a field on a test page equals a specified value.
AssistEdit()	Provides assist-edit functionality to a field on a test page.
AsTime()	Converts the value in a field on a test page to a Time data type.
Caption()	Gets the current caption of the field as a String.
Drilldown()	Applies drill-down capability for a field on a test page.
Editable()	Gets the editable state for the field.
Enabled()	Gets the enabled state for the field.
GetOption([Integer])	Gets the options for a field on a test page.
GetValidationError([Integer])	Gets the validation error that occurred on a test page.
HideValue()	Gets the hide value state for the field.

METHOD NAME	DESCRIPTION
Invoke()	Invokes the default action on the field.
Lookup()	Provides a lookup window for a text box on a test page.
Lookup(RecordRef)	Calls the OnAfterLook trigger with the selected record
OptionCount()	Gets the number of options in a field on a test page.
SetValue(Any)	Sets a value for a field on a test page.
ShowMandatory()	Gets the ShowMandatory state for the field.
ValidationErrorCount()	Gets the number of validation errors that occurred on the test page.
Value([String])	Gets or sets the value of this field.
Visible()	Gets the visible state for the field.

See Also

Getting Started with AL
Developing Extensions

TestField.Activate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Activates a field on a test page.

Syntax

```
TestField.Activate()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AsBoolean Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a field on a test page to a Boolean data type.

Syntax

```
Result :=   TestField.AsBoolean()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Boolean The value of the field as a Boolean.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AsDate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a field on a test page to a Date data type.

Syntax

```
Result :=   TestField.AsDate()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Date The value of the field as a Date.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AsDateTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a field on a test page to a DateTime data type.

Syntax

```
Result :=   TestField.AsDateTime()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: DateTime The value of the field as a DateTime.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AsDecimal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a field on a test page to a Date data type.

Syntax

```
Result :=   TestField.AsDecimal()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Decimal The value of the field as a Decimal.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AsInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value of the field on a test page to an Integer data type.

Syntax

```
Result :=   TestField.AsInteger()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Integer The value of the field as an Integer.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AssertEquals Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Asserts that the value in a field on a test page equals a specified value.

Syntax

```
TestField.AssertEquals(Value: Any)
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Value
Type: Any
Specifies the value that the field should equal.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AssistEdit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides assist-edit functionality to a field on a test page.

Syntax

```
TestField.AssistEdit()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.AsTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value in a field on a test page to a Time data type.

Syntax

```
Result :=   TestField.AsTime()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Time The value of the field as a Time.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current caption of the field as a String.

Syntax

```
Result :=   TestField.Caption()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: String The current caption of the field as a String.

Remarks

Caption returns the caption of a field. Caption first looks for a CaptionML Property.

If it does not find one, it will use the Name Property. This means that Caption is enabled for multilanguage functionality.

This method is similar to the FieldCaption Method (Record) method.

Example

The following example opens table 18 (Customer) as a RecordRef variable that is named CustomerRecref. The code uses the Field Method (RecordRef) to loop through field 1 through 9 and creates a FieldRef variable that is named MyFieldRef. For each field, the Caption method retrieves the caption of the field, stores it in the varCaption variable and displays it in a message box.

```
var
    CustomerRecRef: RecordRef;
    MyFieldRef: FieldRef;
    varCaption: Text;
    i: Integer;
    Text000: Label 'The caption for field %1 is "%2".';
begin
    CustomerRecref.Open(18);
    for i := 1 to 9 do begin
      MyFieldRef := CustomerRecref.Field(i);
      varCaption := MyFieldRef.Caption;
      Message(Text000, i, varCaption);
    end;
    CustomerRecref.Close;
end;
```

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Drilldown Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies drill-down capability for a field on a test page.

Syntax

```
TestField.Drilldown()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Editable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the editable state for the field.

Syntax

```
Result :=   TestField.Editable()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Boolean The editable state for the field.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Enabled Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the enabled state for the field.

Syntax

```
Result :=   TestField.Enabled()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Boolean The enabled state for the field.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.GetOption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the options for a field on a test page.

Syntax

```
Result :=   TestField.GetOption([Index: Integer])
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Index
Type: Integer
The index of the field that you want to get the options from. This parameter is optional.

Return Value

Result Type: String The options for a field on a test page.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.GetValidationError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the validation error that occurred on a test page.

Syntax

```
Result :=   TestField.GetValidationError([Index: Integer])
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Index
Type: Integer
The index of the validation error that occurred on the test page.

Return Value

Result Type: String The validation error that occurred on a test page.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.HideValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the hide value state for the field.

Syntax

```
Result :=   TestField.HideValue()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Boolean The hide value state for the field.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Invoke Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Invokes the default action on the field.

Syntax

```
TestField.Invoke()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Lookup Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Provides a lookup window for a text box on a test page.

Syntax

```
TestField.Lookup()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

See Also

TestField Data Type Getting Started with AL Developing Extensions

TestField.Lookup Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 7.0.*

Calls the OnAfterLook trigger with the selected record

Syntax

```
TestField.Lookup(Selected: RecordRef)
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Selected

Type: RecordRef

Specifies the record that is selected during lookup.

See Also

TestField Data Type Getting Started with AL Developing Extensions

TestField.OptionCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of options in a field on a test page.

Syntax

```
Result :=   TestField.OptionCount()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Integer The number of options in a field on a test page.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.SetValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets a value for a field on a test page.

Syntax

```
TestField.SetValue(Value: Any)
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Value
Type: Any
The value to set.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.ShowMandatory Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the ShowMandatory state for the field.

Syntax

```
Result :=   TestField.ShowMandatory()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Boolean The ShowMandatory state for the field.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.ValidationErrorCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of validation errors that occurred on the test page.

Syntax

```
Result :=   TestField.ValidationErrorCount()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Integer The number of validation errors that occurred on the test page.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Value Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the value of this field.

Syntax

```
[Value := ]  TestField.Value([Value: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestField Type: TestField An instance of the TestField data type.

Value

Type: String

The new value to set for this field.

Return Value

Value Type: String The value of this field.

Remarks

If you omit NewValue, the method returns the current value of the field.

If you want to format the value of a FieldRef you must use `Format(FieldRef)` instead of `Format(FieldRef.Value)`.

Example

The following example opens table 18, the **Customer** table, a RecordRef variable that is named CustomerRecref. The Field Method (RecordRef) creates a FieldRef for the first field (No.). The reference to the field is stored in the MyFieldRef variable. The Active Method (FieldRef) method determines whether the field is enabled. If the field is enabled, then the record that is identified by the number in the CustomerNo variable is selected. The MyFieldRef variable is updated to refer to the second field (Name). Then the value in the second field is changed to Contoso. The Modify Method (RecordRef) modifies the record in the table to reflect the change. `MyFieldRef.Value` retrieves the new name and displays it in message box.

```
var
    CustomerRecRef: RecordRef;
    MyFieldRef: FieldRef;
    CustomerNo: Code;
    Text000: Label 'Customer name has changed to %1.';
begin
    CustomerNo := '50000';
    CustomerRecref.Open(18);
    MyFieldRef := CustomerRecref.Field(1);
    if MyFieldRef.Active then begin
      MyFieldRef.Value(CustomerNo);
      MyFieldRef := CustomerRecref.Field(2);
      MyFieldRef.Value('Contoso');
      CustomerRecref.Modify;
      Message(Text000, MyFieldRef.Value);
    end;
end;
```

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestField.Visible Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the visible state for the field.

Syntax

```
Result :=   TestField.Visible()
```

Parameters

TestField Type: TestField An instance of the TestField data type.

Return Value

Result Type: Boolean The visible state for the field.

See Also

TestField Data Type
Getting Started with AL
Developing Extensions

TestFilter Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a test filter on a page.

The following methods are available on instances of the TestFilter data type.

METHOD NAME	DESCRIPTION
Ascending([Boolean])	Gets or sets the order in which to search through a data set on a test page.
CurrentKey()	Gets the current key of a data set that is displayed on a test page.
GetFilter(TestFilterField)	Gets the filter that is applied to the specified field in a data set that is displayed on a test page.
SetCurrentKey(TestFilterField [, TestFilterField,...])	Sets the specified fields in a data set on a test page as the current key.
SetFilter(TestFilterField, String)	Applies a filter to the specified field on a test page.

See Also

Getting Started with AL
Developing Extensions

TestFilter.Ascending Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the order in which to search through a data set on a test page.

Syntax

```
[Ascending := ]  TestFilter.Ascending([Value: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

TestFilter Type: TestFilter An instance of the TestFilter data type.

Value
Type: Boolean
Sets the order in which to search through a data set on a test page.

Return Value

Ascending Type: Boolean

See Also

TestFilter Data Type
Getting Started with AL
Developing Extensions

TestFilter.CurrentKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current key of a data set that is displayed on a test page.

Syntax

```
CurrentKey :=   TestFilter.CurrentKey()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestFilter Type: TestFilter An instance of the TestFilter data type.

Return Value

CurrentKey Type: String

See Also

TestFilter Data Type
Getting Started with AL
Developing Extensions

TestFilter.GetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the filter that is applied to the specified field in a data set that is displayed on a test page.

Syntax

```
String :=   TestFilter.GetFilter(Field: TestFilterField)
```

Parameters

TestFilter Type: TestFilter An instance of the TestFilter data type.

Field
Type: TestFilterField
The field that you want to get the filter from.

Return Value

String Type: String

See Also

TestFilter Data Type
Getting Started with AL
Developing Extensions

TestFilter.SetCurrentKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the specified fields in a data set on a test page as the current key.

Syntax

```
[Ok := ]  TestFilter.SetCurrentKey(Field1: TestFilterField [, Field2: TestFilterField,...])
```

Parameters

TestFilter Type: TestFilter An instance of the TestFilter data type.

Field1
Type: TestFilterField
The field that you want to set as the current key.

Field2
Type: TestFilterField
Additional field that you want to set as the current key. This parameter is optional.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestFilter Data Type
Getting Started with AL
Developing Extensions

TestFilter.SetFilter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies a filter to the specified field on a test page.

Syntax

```
TestFilter.SetFilter(Field: TestFilterField, String: String)
```

Parameters

TestFilter Type: TestFilter An instance of the TestFilter data type.

Field

Type: TestFilterField

The field that you want to apply the filter to.

String

Type: String

The filter to apply to the specified field.

See Also

TestFilter Data Type
Getting Started with AL
Developing Extensions

TestFilterField Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the type of a field filter in a test filter on a page or on a request page.

See Also

Getting Started with AL
Developing Extensions

TestPage Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a variable type that can be used to test Page Application Objects.

The following methods are available on instances of the TestPage data type.

METHOD NAME	DESCRIPTION
Cancel()	Gets the Cancel system action.
Caption()	Gets the caption of the test page.
Close()	Closes an open test page.
Edit()	Gets the Edit system action.
Editable()	Gets the runtime value of the Editable property on a test page.
Expand(Boolean)	Expands rows on a test page.
FindFirstField(TestField, Any)	Finds the first field in the data set that is displayed on a test page.
FindNextField(TestField, Any)	Finds the next field in the data set that is displayed on a test page.
FindPreviousField(TestField, Any)	Finds the previous field in the data set that is displayed on a test page.
First()	Sets the current row of the test page as the first row in the data set.
GetField(Integer)	Gets a field on a test page.
GetValidationError([Integer])	Gets the list of all validation error that occurred on a test page as a string.
GoToKey([Any,...])	Finds the row in a data set on the test page that is identified by the specified values.
GoToRecord(Record)	Finds the specified record in a data set on a test page.
IsExpanded()	Specifies if rows on a test page are expanded.
Last()	Sets the current row of the test page as the last row in the data set.

METHOD NAME	DESCRIPTION
New()	Sets the current row of the test page to an empty row in a data set.
Next()	Sets the current row of the test page as the next row in the data set.
No()	Gets the No system action.
OK()	Gets the OK system action.
OpenEdit()	Opens a test page in edit mode.
OpenNew()	Opens a blank test page in edit mode.
OpenView()	Opens a test page in view mode.
Prev()	Sets the current row of the test page as the previous row in the data set.
Previous()	Sets the current row of the test page as the previous row in the data set.
RunPageBackgroundTask(Integer [, var Dictionary of [Text, Text]] [, Boolean])	Runs the page background task codeunit in the current session. Note that by default, no triggers are invoked at this point.
Trap()	Traps the next test page that is invoked and assigns it to the test page variable.
ValidationErrorCount()	Gets the number of validation errors that occurred on the test page.
View()	Gets the View system action.
Yes()	Gets the Yes system action.

See Also

Getting Started with AL
Developing Extensions

TestPage.Cancel Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the Cancel system action.

Syntax

```
Action :=   TestPage.Cancel()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Action Type: TestAction The Cancel system action.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the caption of the test page.

Syntax

```
String :=   TestPage.Caption()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

String Type: String The caption of the test page.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Close Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Closes an open test page.

Syntax

```
TestPage.Close()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Remarks

If *TestPage* has never been opened or is already closed, then a run-time error occurs, and the outcome of the test method is FAILURE.

Example

This example requires that you create a TestPage variable named CustTestPage with a Subtype of Customer List and that the codeunit in which you write the code is a test codeunit.

```
// Open the test page.
CustPage.OpenEdit;
// Add code to test the Customer List page.
// …
// Close the Customer Card
CustPage.Close;
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Edit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the Edit system action.

Syntax

```
Action :=   TestPage.Edit()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Action Type: TestAction The Edit system action.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Editable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the runtime value of the Editable property on a test page.

Syntax

```
Editable :=   TestPage.Editable()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Editable Type: Boolean The runtime value of the Editable property on the test page.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Expand Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Expands rows on a test page.

Syntax

```
TestPage.Expand(Expand: Boolean)
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Expand
Type: Boolean
The rows that you want to expand on the test page.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.FindFirstField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the first field in the data set that is displayed on a test page.

Syntax

```
[Ok := ]  TestPage.FindFirstField(Field: TestField, Value: Any)
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Field
Type: TestField
The field to find.

Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.FindNextField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the next field in the data set that is displayed on a test page.

Syntax

```
[Ok := ]  TestPage.FindNextField(Field: TestField, Value: Any)
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Field
Type: TestField
The field to find.

Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.FindPreviousField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the previous field in the data set that is displayed on a test page.

Syntax

```
[Ok := ]  TestPage.FindPreviousField(Field: TestField, Value: Any)
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Field
Type: TestField
The field to find.

Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.First Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the first row in the data set.

Syntax

```
[Ok := ]  TestPage.First()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**.

Remarks

If *TestPage* is closed or has never been opened, then the method call fails.

Example

This example sets the current row to the first customer in the dataset. It requires that you create a TestPage variable named CustomerList with a Subtype of Customer List.

```
CustomerList.OpenView;
CustomerList.First;
Message(CustomerList.Name.Value);
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.GetField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0 until version 3.0 where it was deprecated.*

Gets a field on a test page.

Syntax

```
TestField :=   TestPage.GetField(Id: Integer)
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Id

Type: Integer

The ID of the field that you want to get.

Return Value

TestField Type: TestField The field on the test page.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.GetValidationError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the list of all validation error that occurred on a test page as a string.

Syntax

```
Error :=   TestPage.GetValidationError([Index: Integer])
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Index
Type: Integer
The index of the validation error that occurred on the test page.

Return Value

Error Type: String A string where each line represents a validation error that occured on the test page.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.GoToKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the row in a data set on the test page that is identified by the specified values.

Syntax

```
[Ok := ]  TestPage.GoToKey([Value: Any,...])
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Value
Type: Any
The value or list of values to use to find the row. If this parameter is omitted, the value of the primary key that is defined for the underlying table is used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The GoToKey method loops over all records until it finds the identifies row. For each record, the OnAfterGetCurrentRecord Trigger is executed.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.GoToRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the specified record in a data set on a test page.

Syntax

```
[Ok := ]  TestPage.GoToRecord(Rec: Record)
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Rec
Type: Record
The record to find.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Remarks

The GoToRecord method loops over all records until it finds the identifies record. For each record, the OnAfterGetCurrentRecord Trigger is executed.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.IsExpanded Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies if rows on a test page are expanded.

Syntax

```
Expanded :=   TestPage.IsExpanded()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Expanded Type: Boolean **true** if the rows on the test page are expanded, otherwise **false**.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Last Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the last row in the data set.

Syntax

```
[Ok := ]  TestPage.Last()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**.

Remarks

If *TestPage* is closed or has never been opened, then the method call fails.

The Last method loops over all records until it sets the identifies the current record. For each record, the OnAfterGetCurrentRecord Trigger is executed.

Example

This example sets the current row to the last customer in the dataset. It requires that you create a TestPage variable named CustomerList with a Subtype of Customer List.

```
CustomerList.OpenView;
CustomerList.Last;
Message(CustomerList.Name.Value);
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.New Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page to an empty row in a data set.

Syntax

```
TestPage.New()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Next Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the next row in the data set.

Syntax

```
[Ok := ]  TestPage.Next()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**.

Remarks

If *TestPage* is closed or has never been opened, then the method call fails.

Example

This example requires that you create a TestPage variable named CustomerList with a Subtype of Customer List.

```
CustomerList.OpenView;
// Loops through all customers and displays the customer name.
if CustomerList.First then
  repeat
    Message('%1';CustomerList.Name);
until not CustomerList.NEXT;
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.No Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the No system action.

Syntax

```
Action :=   TestPage.No()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Action Type: TestAction The No system action.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.OK Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the OK system action.

Syntax

```
Action :=   TestPage.OK()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Action Type: TestAction The OK system action.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.OpenEdit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Opens a test page in edit mode.

Syntax

```
TestPage.OpenEdit()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Remarks

If *TestPage* is already open, then a run-time error occurs, and the outcome of the test method is FAILURE.

If the page to which *TestPage* refers is a card page, then the data that is loaded on the page is from the first row in the dataset.

Example

This example requires that you create a TestPage variable named CustTestPage with a Subtype of Customer List and that the codeunit in which you write the code is a test codeunit.

```
// Opens the TestPage in edit mode.
CustTestPage.OpenEdit()
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.OpenNew Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Opens a blank test page in edit mode.

Syntax

```
TestPage.OpenNew()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Remarks

If *TestPage* is already open, then a run-time error occurs, and the outcome of the test method is FAILURE.

Example

This example requires that you create a TestPage variable named CustTestPage with a Subtype of Customer List and that the codeunit in which you write the code is a test codeunit.

```
// Opens a blank Customer Card test page.
CustTestPage.OpenNEW;
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.OpenView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Opens a test page in view mode.

Syntax

```
TestPage.OpenView()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Remarks

If *TestPage* is already open, then a run-time error occurs, and the outcome of the test method is FAILURE.

If the page to which *TestPage* refers is a card page, then the data that is loaded on the page is from the first row in the dataset.

Example

This example requires that you create a variable named CustTestPage with a Subtype of Customer List and that the codeunit in which you write the code is a test codeunit.

```
// Opens the Customer Card on the first Customer in the dataset.
CustTestPage.OpenView;
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Prev Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0 until version 3.0 where it was deprecated.*

Sets the current row of the test page as the previous row in the data set.

Syntax

```
[Ok := ]  TestPage.Prev()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Previous Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the previous row in the data set.

Syntax

```
[Ok := ]  TestPage.Previous()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**.

Remarks

If *TestPage* is closed or has never been opened, then the method call fails.

Example

This example requires that you create a TestPage variable named CustomerList with a Subtype of Customer List.

```
CustomerList.OpenView;
…
if CustomerList.Last then repeat
  Message(CustomerList.Name);
until not CustomerList.Previous;
```

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.RunPageBackgroundTask Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 4.0.*

Runs the page background task codeunit in the current session. Note that by default, no triggers are invoked at this point.

Syntax

```
Results :=   TestPage.RunPageBackgroundTask(CodeunitId: Integer [, var Parameters: Dictionary of [Text, Text]] [, RunCompletionTriggers: Boolean])
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

CodeunitId
Type: Integer
Specifies the ID of the codeunit to run.
Parameters
Type: Dictionary of [Text, Text]
Specifies a collection of keys and values that are passed to the OnRun trigger of the codeunit that runs when the page background task session is started.
RunCompletionTriggers
Type: Boolean
Runs the completion triggers after the completion of the code unit. Default value is **false**.

Return Value

Results Type: Dictionary of [Text, Text] The dictionary of results for the page background task.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Trap Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Traps the next test page that is invoked and assigns it to the test page variable.

Syntax

```
TestPage.Trap()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Remarks

Only non-modal pages can be trapped.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.ValidationErrorCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of validation errors that occurred on the test page.

Syntax

```
Count :=   TestPage.ValidationErrorCount()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Count Type: Integer The number of validation errors that occurred on the test page.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.View Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the View system action.

Syntax

```
Action :=   TestPage.View()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Action Type: TestAction The view system action.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPage.Yes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the Yes system action.

Syntax

```
Action :=   TestPage.Yes()
```

Parameters

TestPage Type: TestPage An instance of the TestPage data type.

Return Value

Action Type: TestAction The Yes system action.

See Also

TestPage Data Type
Getting Started with AL
Developing Extensions

TestPart Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a variable type that can be used to test Page Application Objects of type Part.

The following methods are available on instances of the TestPart data type.

METHOD NAME	DESCRIPTION
Caption()	Gets the caption of the test page.
Editable()	Gets the runtime value of the Editable property on a test page.
Expand(Boolean)	Expands rows on a test page.
FindFirstField(TestField, Any)	Finds the first field in the data set that is displayed on a test page.
FindNextField(TestField, Any)	Finds the next field in the data set that is displayed on a test page.
FindPreviousField(TestField, Any)	Finds the previous field in the data set that is displayed on a test page.
First()	Sets the current row of the test page as the first row in the data set.
GetField(Integer)	Gets a field on a test page.
GetValidationError([Integer])	Gets the list of all validation error that occurred on a test page as a string.
GoToKey([Any,...])	Finds the row in a data set on the test page that is identified by the specified values.
GoToRecord(Record)	Finds the specified record in a data set on a test page.
IsExpanded()	Specifies if the current row on the test page is expanded.
Last()	Sets the current row of the test page as the last row in the data set.
New()	Sets the current row of the test page to an empty row in a data set.
Next()	Sets the current row of the test page as the next row in the data set.

METHOD NAME	DESCRIPTION
Prev()	Sets the current row of the test page as the previous row in the data set.
Previous()	Sets the current row of the test page as the previous row in the data set.
ValidationErrorCount()	Gets the number of validation errors that occurred on the test page.

See Also

Getting Started with AL
Developing Extensions

TestPart.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the caption of the test page.

Syntax

```
String :=   TestPart.Caption()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

String Type: String The caption of the test page.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.Editable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the runtime value of the Editable property on a test page.

Syntax

```
Editable :=   TestPart.Editable()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Editable Type: Boolean The runtime value of the Editable property on a test page.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.Expand Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Expands rows on a test page.

Syntax

```
TestPart.Expand(Expand: Boolean)
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Expand
Type: Boolean
The rows that you want to expand on the test page.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.FindFirstField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the first field in the data set that is displayed on a test page.

Syntax

```
[Ok := ]  TestPart.FindFirstField(Field: TestField, Value: Any)
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Field
Type: TestField
The field to find.
Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.FindNextField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the next field in the data set that is displayed on a test page.

Syntax

```
[Ok := ]  TestPart.FindNextField(Field: TestField, Value: Any)
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Field
Type: TestField
The field to find.
Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.FindPreviousField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the previous field in the data set that is displayed on a test page.

Syntax

```
[Ok := ]  TestPart.FindPreviousField(Field: TestField, Value: Any)
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Field
Type: TestField
The field to find.
Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.First Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the first row in the data set.

Syntax

```
[Ok := ]  TestPart.First()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Ok Type: Boolean True, if a first row is present.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.GetField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0 until version 3.0 where it was deprecated.*

Gets a field on a test page.

Syntax

```
TestField :=   TestPart.GetField(Id: Integer)
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Id

Type: Integer

The ID of the field that you want to get.

Return Value

TestField Type: TestField The field on the test page.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.GetValidationError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the list of all validation error that occurred on a test page as a string.

Syntax

```
Error :=   TestPart.GetValidationError([Index: Integer])
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Index
Type: Integer
The index of the validation error that occurred on the test page.

Return Value

Error Type: String A string where each line represents a validation error that occured on the test page.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.GoToKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the row in a data set on the test page that is identified by the specified values.

Syntax

```
[Ok := ]  TestPart.GoToKey([Value: Any,...])
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Value
Type: Any
The value or list of values to use to find the row. If this parameter is omitted, the value of the primary key that is defined for the underlying table is used.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.GoToRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the specified record in a data set on a test page.

Syntax

```
[Ok := ]  TestPart.GoToRecord(Rec: Record)
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Rec
Type: Record
The record to find.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.IsExpanded Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies if the current row on the test page is expanded.

Syntax

```
Expanded :=   TestPart.IsExpanded()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Expanded Type: Boolean **true** if the current row on the test page is expanded, otherwise **false**.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.Last Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the last row in the data set.

Syntax

```
[Ok := ]  TestPart.Last()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Ok Type: Boolean True, if a last row is present.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.New Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page to an empty row in a data set.

Syntax

```
TestPart.New()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.Next Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the next row in the data set.

Syntax

```
[Ok := ]  TestPart.Next()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Ok Type: Boolean True, if the current row has changed.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.Prev Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 3.0 until version 3.0 where it was deprecated.*

Sets the current row of the test page as the previous row in the data set.

Syntax

```
[Ok := ]  TestPart.Prev()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Ok Type: Boolean True, if the current row has changed.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.Previous Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the previous row in the data set.

Syntax

```
[Ok := ]  TestPart.Previous()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Ok Type: Boolean True, if the current row has changed.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestPart.ValidationErrorCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of validation errors that occurred on the test page.

Syntax

```
Count :=   TestPart.ValidationErrorCount()
```

Parameters

TestPart Type: TestPart An instance of the TestPart data type.

Return Value

Count Type: Integer Number of validation errors that occured on the test page.

See Also

TestPart Data Type
Getting Started with AL
Developing Extensions

TestRequestPage Data Type

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Stores test request pages. A test request page part is a logical representation of a request page on a report. A test request page does not display a user interface (UI). The subtype of a test request page is the report whose request page you want to test.

The following methods are available on instances of the TestRequestPage data type.

METHOD NAME	DESCRIPTION
Cancel()	Gets a TestAction representing an action on the Page under Test.
Caption()	Gets the caption of the test page.
Editable()	Gets the runtime value of the Editable property on a test page.
Expand(Boolean)	Expands rows on a test page.
FindFirstField(TestField, Any)	Finds the first field in the data set that is displayed on a test page. The row is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.
FindNextField(TestField, Any)	Finds the next field in the data set that is displayed on a test page.The row is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.
FindPreviousField(TestField, Any)	Finds the previous field in the data set that is displayed on a test page. The row is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.
First()	Sets the current row of the test page as the first row in the data set.
GetValidationError([Integer])	Gets the validation error that occurred on a test page.

METHOD NAME	DESCRIPTION
GoToKey([Any,...])	Finds the row in a data set on the test page that is identified by the specified values. The key is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.
GoToRecord(Record)	Finds the specified record in a data set on a test page. The record is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.
IsExpanded()	Specifies if rows on a test page are expanded.
Last()	Sets the current row of the test page as the last row in the data set.
New()	Sets the current row of the test page to an empty row in a data set.
Next()	Sets the current row of the test page as the next row in the data set.
OK()	Gets a TestAction representing an action on the Page under Test.
Preview()	Gets a TestAction representing an action on the Page under Test.
Previous()	Sets the current row of the test page as the previous row in the data set.
Print()	Gets a the Print representing an action on the Page under Test.
SaveAsExcel(String)	Saves a report as a Microsoft Excel (.xls) file.
SaveAsPdf(String)	Saves a report as an Adobe Acrobat (.pdf) file.
SaveAsWord(String)	Saves a report as a Microsoft Word (.doc) file.
SaveAsXml(String, String)	Saves a report data set and the labels on a report as two XML (.xml) files.
Schedule()	Gets a TestAction representing an action on the Page under Test.
ValidationErrorCount()	Gets the number of validation errors that occurred on the test page.

See Also

TestRequestPage.Cancel Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a TestAction representing an action on the Page under Test.

Syntax

```
Action :=   TestRequestPage.Cancel()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Action Type: TestAction TestAction representing an action on the Page under Test.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Caption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the caption of the test page.

Syntax

```
String :=   TestRequestPage.Caption()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

String Type: String The caption of the test page.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Editable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the runtime value of the Editable property on a test page.

Syntax

```
Editable :=   TestRequestPage.Editable()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Editable Type: Boolean The runtime value of the Editable property.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Expand Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Expands rows on a test page.

Syntax

```
TestRequestPage.Expand(Expand: Boolean)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Expand
Type: Boolean
The rows that you want to expand on the test page.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.FindFirstField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the first field in the data set that is displayed on a test page. The row is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.

Syntax

```
[Ok := ]  TestRequestPage.FindFirstField(Field: TestField, Value: Any)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Field
Type: TestField
The field to find.

Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the row is found, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.FindNextField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the next field in the data set that is displayed on a test page.The row is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.

Syntax

```
[Ok := ]  TestRequestPage.FindNextField(Field: TestField, Value: Any)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Field
Type: TestField
The field to find.
Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the row is found, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.FindPreviousField Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the previous field in the data set that is displayed on a test page. The row is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.

Syntax

```
[Ok := ]  TestRequestPage.FindPreviousField(Field: TestField, Value: Any)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Field
Type: TestField
The field to find.

Value
Type: Any
The value of the field.

Return Value

Ok Type: Boolean **true** if the row is found, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.First Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the first row in the data set.

Syntax

```
[Ok := ]  TestRequestPage.First()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Ok Type: Boolean **true** if a first row is present, otherwise **false**.Throws a NavTestPageNotOpenedException if the page is not opened.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.GetValidationError Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the validation error that occurred on a test page.

Syntax

```
Error :=   TestRequestPage.GetValidationError([Index: Integer])
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Index

Type: Integer

The index of the validation error that occurred on the test page.

Return Value

Error Type: String The validation error that occured at the specified index.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.GoToKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the row in a data set on the test page that is identified by the specified values. The key is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.

Syntax

```
[Ok := ]  TestRequestPage.GoToKey([Value: Any,...])
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Value
Type: Any
The value or list of values to use to find the row. If this parameter is omitted, the value of the primary key that is defined for the underlying table is used.

Return Value

Ok Type: Boolean **true** if the key is found, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.GoToRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the specified record in a data set on a test page. The record is searched from the beginning of the dataset defined by the current filter. The search is performed by iterating across the rows, comparing the primary key with the primary key fields of the record. For large dataset, use SetFilter to limit the dataset.

Syntax

```
[Ok := ]  TestRequestPage.GoToRecord(Rec: Record)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Rec
Type: Record
The record to find.

Return Value

Ok Type: Boolean **true** if the record is found, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.IsExpanded Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Specifies if rows on a test page are expanded.

Syntax

```
Expanded :=   TestRequestPage.IsExpanded()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Expanded Type: Boolean **true** if the rows on the test page are expanded, otherwise **false**.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Last Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the last row in the data set.

Syntax

```
[Ok := ]  TestRequestPage.Last()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Ok Type: Boolean **true** if a last row is present, otherwise **false**. Throws a NavTestPageNotOpenedException if the page is not opened.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.New Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page to an empty row in a data set.

Syntax

```
TestRequestPage.New()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Next Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the next row in the data set.

Syntax

```
[Ok := ]  TestRequestPage.Next()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**. Throws a NavTestPageNotOpenedException if the page is not opened.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.OK Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a TestAction representing an action on the Page under Test.

Syntax

```
Action :=   TestRequestPage.OK()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Action Type: TestAction TestAction representing an action on the Page under Test.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Preview Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a TestAction representing an action on the Page under Test.

Syntax

```
Action :=   TestRequestPage.Preview()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Action Type: TestAction TestAction representing an action on the Page under Test.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Previous Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the current row of the test page as the previous row in the data set.

Syntax

```
[Ok := ]  TestRequestPage.Previous()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Ok Type: Boolean **true** if the current row has changed, otherwise **false**.Throws a NavTestPageNotOpenedException if the page is not opened.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Print Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a the Print representing an action on the Page under Test.

Syntax

```
Action :=   TestRequestPage.Print()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Action Type: TestAction TestAction representing an action on the Page under Test.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.SaveAsExcel Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as a Microsoft Excel (.xls) file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
TestRequestPage.SaveAsExcel(FileName: String)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

FileName
Type: String
The path and file name to which the report is saved. The file name extension should be .xls.

Remarks

All filters and options that have been set on the TestRequestPage are respected in the saved report.

After you run this method, you cannot continue to interact with the *TestRequestPage*. If you want to continue to use the *TestRequestPage* variable, you must run a report again.

Example

The following example shows the code for a test method to run a report and a request page handler method to test the request page. This example requires that you create the following:

- A test codeunit called SaveAsExcel.
- A test method in the test codeunit called TestSaveAsExcel.
- A handler method of type RequestPageHandler called ReqPageHandler. This handler method has one parameter called RequestPage of Type TestRequestPage and Subtype Customer – Top 10 List. The RequestPage parameter is specified as VAR and is passed by reference to the handler method.

```
var
    Filename: Text;
begin
    //Test method: TestSaveAsExcel
    Filename := TemporaryPath + 'MyRep.xls';
    Message(Filename);
    if not File.Erase(Filename) then
      Error('Cannot erase %1',Filename);
    Report.Run(111);
    if not File.Exists(Filename) then
      Error('File should exist!');

    //Request Page Handler method
    RequestPage.Customer.SetFilter("No.", '20000');
    RequestPage.ChartType.Value('Pie chart');
    RequestPage.SaveAsExcel(Filename);
end;
```

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.SaveAsPdf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as an Adobe Acrobat (.pdf) file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
TestRequestPage.SaveAsPdf(FileName: String)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

FileName
Type: String
The path and file name to which the report is saved. The file name extension should be .pdf.

Remarks

All filters and options that have been set on the TestRequestPage are respected in the saved report.

After you run this method, you cannot continue to interact with the *TestRequestPage*. If you want to continue to use the *TestRequestPage* variable, you must run a report again.

Example

The following example shows the code for a test method to run a report and a request page handler method to test the request page. This example requires that you create the following:

- A test codeunit called SaveAsPDF.
- A test method in the test codeunit called TestSaveAsPDF.
- A handler method of type RequestPageHandler called ReqPageHandler. This handler method has one parameter called RequestPage of Type TestRequestPage and Subtype Customer – Top 10 List. The RequestPage parameter is specified as VAR and is passed by reference to the handler method.

```
var
    Filename: Text;
begin
    //Test method: TestSaveAsPDF
    Filename := TemporaryPath + 'MyRep.pdf';
    Message(Filename);
    if not File.Erase(Filename) then
      Error('Cannot erase %1',Filename);
    Report.Run(111);
    if not File.Exists(Filename) then
      Error('File should exist!');

    //Request Page Handler method
    RequestPage.Customer.SetFilter("No.", '20000');
    RequestPage.ChartType.Value('Pie chart');
    RequestPage.SaveAsPDF(Filename);
end;
```

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.SaveAsWord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report as a Microsoft Word (.doc) file.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
TestRequestPage.SaveAsWord(FileName: String)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

FileName
Type: String
The path and file name to which the report is saved. The file name extension should be .doc.

Remarks

All filters and options that have been set on the TestRequestPage are respected in the saved report.

After you run this method, you cannot continue to interact with the *TestRequestPage*. If you want to continue to use the *TestRequestPage* variable, you must run a report again.

Example

The following example shows the code for a test method to run a report and a request page handler method to test the request page. This example requires that you create the following:

- A test codeunit called SaveAsWord.
- A test method in the test codeunit called TestSaveAsWord.
- A handler method of type RequestPageHandler called ReqPageHandler. This handler method has one parameter called RequestPage of Type TestRequestPage and Subtype Customer – Top 10 List. The RequestPage parameter is specified as VAR and is passed by reference to the handler method.

```
var
    Filename: Text;
begin
    //Test method: TestSaveAsWord
    Filename := TemporaryPath + 'MyRep.doc';
    Message(Filename);
    if not File.Erase(Filename) then
      Error('Cannot erase %1',Filename);
    Report.Run(111);
    if not File.Exists(Filename) then
      Error('File should exist!');

    //Request Page Handler method
    RequestPage.Customer.SetFilter("No.", '20000');
    RequestPage.ChartType.Value('Pie chart');
    RequestPage.SaveAsExcel(Filename);
end;
```

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.SaveAsXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Saves a report data set and the labels on a report as two XML (.xml) files.

> **NOTE**
> This method is supported only in Business Central on-premises.

Syntax

```
TestRequestPage.SaveAsXml(ParameterFileName: String, DataSetFileName: String)
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

ParameterFileName
Type: String
The path and file name to which the paramater file is saved.

DataSetFileName
Type: String
The path and file name to which the data set file is saved.

Remarks

All filters and options that have been set on the TestRequestPage are respected in the saved report.

After you run this method, you cannot continue to interact with the *TestRequestPage*. If you want to continue to use the *TestRequestPage* variable, you must run a report again.

Example

The following example shows the code for a test method to run a report and a request page handler method to test the request page. This example requires that you create the following:

- A test codeunit called SaveAsXML.
- A test method in the test codeunit called TestSaveAsXML.
- A handler method of type RequestPageHandler called ReqPageHandler. This handler method has one parameter called RequestPage of Type TestRequestPage and Subtype Customer – Top 10 List. The RequestPage parameter is specified as VAR and is passed by reference to the handler method.

```
var
    LabelsFilename: Text;
    DatasetFilename: Text;
begin
    // Add the following code to the TestSaveAsXML test method.
    LabelsFilename := TemporaryPath + 'MyLabels.xml';
    DatasetFilename := TemporaryPath + 'MyDataset.xml';
    Report.Run(111);
    if not File.Exists(LabelsFilename) then
      Error('Labels file should exist!');
    if not File.Exists(DatasetFilename) then
      Error('Dataset file should exist!');

    // Add the following code to the ReqPageHandler method.
    RequestPage.Customer.SetFilter("No.", '20000');
    RequestPage.ChartType.Value('Pie chart');
    RequestPage.SaveAsXML(LabelsFilename,DatasetFilename);
end;
```

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.Schedule Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a TestAction representing an action on the Page under Test.

Syntax

```
Action :=   TestRequestPage.Schedule()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Action Type: TestAction TestAction representing an action on the Page under Test.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

TestRequestPage.ValidationErrorCount Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of validation errors that occurred on the test page.

Syntax

```
Count :=   TestRequestPage.ValidationErrorCount()
```

Parameters

TestRequestPage Type: TestRequestPage An instance of the TestRequestPage data type.

Return Value

Count Type: Integer The number of validation errors that occurred on the test page.

See Also

TestRequestPage Data Type
Getting Started with AL
Developing Extensions

Text Data Type

2/17/2021 • 5 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes a text string.

The following methods are available on the Text data type.

METHOD NAME	DESCRIPTION
ConvertStr(String, String, String)	Replaces all chars in source found in FromCharacters with the corresponding char in ToCharacters and returns the converted string. If the length of the FromCharacters parameter and the ToChars parameter are different, an exception is thrown. If the parameter FromCharacters or the parameter ToChars is empty, the source is returned unmodified. Each element in source is only converted ONCE a double-replacement cannot happen.
CopyStr(String, Integer [, Integer])	Copies a substring of any length from a specific position in a string (text or code) to a new string.
DelChr(String [, String] [, String])	Deletes chars contained in the which parameter in a string based on the contents on the where parameter. If the where parameter contains an equal-sign, then all occurrences of characters in which is deleted from the current value. If the where parameter contains a less-than, then the characters are only deleted when they are first in the string. If the where parameter contains a greater-than, then the characters are only deleted when they are the last in the string. If the where parameter contains any other char, an exception is thrown. If the where parameter or the which parameter is empty, the source is returned unmodified. The which parameter is to be considered as an array of chars to delete where the order does not matter.
DelStr(String, Integer [, Integer])	Deletes a substring inside a string (text or code).
IncStr(String)	Increases a positive number or decrease a negative number inside a string by one (1).
InsStr(String, String, Integer)	Inserts a substring into a string.
LowerCase(String)	Converts all letters in a string to lowercase.
MaxStrLen(String)	Gets the maximum defined length of a string variable.
MaxStrLen(Variant)	Gets the maximum defined length of a variant variable.
PadStr(String, Integer [, String])	Changes the length of a string to a specified length. If the string is shorter than the specified length, length spaces are added at the end of the string to match the length. If the string is longer than the specified length, the string is truncated. If the specified length is less than 0, an exception is thrown.
SelectStr(Integer, String)	Retrieves a substring from a comma-separated string.

METHOD NAME	DESCRIPTION
StrCheckSum(String [, String] [, Integer])	Calculates a checksum for a string that contains a number. If the source is empty, 0 is returned. Each char in the source and in the weight must be a numeric character 0-9, otherwise an exception is thrown. If the WeightString parameter is shorter then the source, it is padded with '1' up until the length of source. If the WeightString parameter is longer than the source, an exception is thrown.
StrLen(String)	Gets the length of a string you define.
StrPos(String, String)	Searches for the first occurrence of substring inside a string.
StrSubstNo(String [, Any,...])	Replaces %1, %2, %3... and #1, #2, #3... fields in a string with the values you provide as optional parameters.
UpperCase(String)	Converts all letters in a string to uppercase.

The following methods are available on instances of the Text data type.

METHOD NAME	DESCRIPTION
Contains(Text)	Returns a value indicating whether a specified substring occurs within this string.
EndsWith(Text)	Determines whether the end of this string instance matches the specified string.
IndexOf(Text [, Integer])	Reports the one-based index of the first occurrence of the specified string in this instance.
IndexOfAny(Text [, Integer])	Reports the one-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position.
IndexOfAny(List of [Char] [, Integer])	Reports the one-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position.
LastIndexOf(Text [, Integer])	Reports the one-based index position of the last occurrence of a specified string in this instance.
PadLeft(Integer [, Char])	Returns a new Text that right-aligns the characters in this instance by padding them on the left, for a specified total length.
PadRight(Integer [, Char])	Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.
Remove(Integer [, Integer])	Returns a new Text in which a specified number of characters from the current string are deleted.
Replace(Text, Text)	Returns a new Text in which all occurrences of a specified string in the current instance are replaced with another specified string.
Split([Text,...])	Splits a string into a maximum number of substrings based on a collection of separators.
Split(List of [Text])	Splits a string into a maximum number of substrings based on a collection of separators.

METHOD NAME	DESCRIPTION
Split(List of [Char])	Splits a string into a maximum number of substrings based on a collection of separators.
StartsWith(Text)	Determines whether the beginning of this instance matches a specified string.
Substring(Integer [, Integer])	Retrieves a substring from this instance.
ToLower()	Returns a copy of this string converted to lowercase.
ToUpper()	Returns a copy of this string converted to uppercase.
Trim()	Returns a new Text in which all leading and trailing white-space characters from the current Text object are removed.
TrimEnd([Text])	Removes all trailing occurrences of a set of characters specified in an array from the current Text object.
TrimStart([Text])	Removes all leading occurrences of a set of characters specified in an array from the current Text object.

Remarks

The **Text** data type is a value type, such that every time you use a method on it, you create a new string object in memory. This requires a new allocation of space. In situations where you need to perform repeated modifications to a string, the overhead associated with creating a **Text** data type can be costly.

The TextBuilder Data Type is a reference type, which holds a pointer elsewhere in memory. For performance reasons, we recommend you to use it when you want to modify a string without creating a new object. For example, using TextBuilder Data Type can boost performance when concatenating many strings together in a loop.

See Also

Getting Started with AL
Developing Extensions
TextBuilder Data Type

Text.ConvertStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces all chars in source found in FromCharacters with the corresponding char in ToCharacters and returns the converted string. If the length of the FromCharacters parameter and the ToChars parameter are different, an exception is thrown. If the parameter FromCharacters or the parameter ToChars is empty, the source is returned unmodified. Each element in source is only converted ONCE a double-replacement cannot happen.

Syntax

```
NewString :=   Text.ConvertStr(String: String, FromCharacters: String, ToCharacters: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string that you want to convert.

FromCharacters

Type: String

The characters that you want to replace. This function is case-sensitive.

ToCharacters

Type: String

The new characters with which you want to replace the FromCharacters. This function is case-sensitive.

Return Value

NewString Type: String The input string with the converted characters.

Remarks

The characters in the *FromCharacters* parameter are replaced by the characters in the *ToCharacters* parameter.

If the lengths of the *FromCharacters* and *ToCharacters* strings are not equal, then a run-time error occurs.

If either the *FromCharacters* or the *ToCharacters* strings are empty, then the source is returned unchanged.

Example

```
var
    OriginalString: Text[30];
    FromChars: Text[30];
    ToChars: Text[30];
    NewString: Text[30];
    Text000: Label 'Do you want to leave without saving?';
    Text001: Label 'lws';
    Text002: Label 'LWS';
    Text003: Label 'The original sentence is \\: %1';
    Text004: Label 'The sentence is converted to:\\ %1';
begin
    OriginalString := Text000;
    FromChars := Text001;
    ToChars := Text002;
    NewString := ConvertStr(OriginalString, FromChars, ToChars);
    Message(Text003, OriginalString);
    Message(Text004, NewString);
end;
```

The first message window shows:

The original sentence is:

Do you want to leave without saving?

The second message window shows:

The sentence is converted to:

Do you Want to Leave Without Saving?

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.CopyStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Copies a substring of any length from a specific position in a string (text or code) to a new string.

Syntax

```
NewString :=   Text.CopyStr(String: String, Position: Integer [, Length: Integer])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string that you want to copy from.

Position

Type: Integer

The position of the first character to copy. If the value of Position is less than 1, then the CopyStr function returns an error. If Position is greater than the length of the string, then the CopyStr function returns an empty string.

Length

Type: Integer

The number of characters to copy. If the value of Length is less than 0, then the CopyStr function returns an error. If the value of Length causes Position + Length to be > (total length of the string), then the result includes all the characters from Position to the end of the string. If you omit Length, then the resulting string includes all the characters from Position to the end of the string.

Return Value

NewString Type: String The copied string.

Remarks

If *Position* combined with *Length* exceeds the length of the string, all the characters from *Position* to the end of the string are returned.

Example

```
var
    Str: Text[30];
    Position: Integer;
    Length: Integer;
    NewStr: Text[30];
    Text000: Label 'Using the CopyStr method';
    Text001: Label 'The original string is:>%1<';
    Text002: Label 'The copied string is:>%1<';
begin
    Str := Text000;
    Position := 7;
    Length := 8;
    Message(Text001, Str);
    NewStr := CopyStr(Str, Position, Length);
    Message(Text002, NewStr);
end;
```

The first message window shows the original string:

The original string is:

>Using the CopyStr method<

The second message window shows the copied string:

The copied string is:

>the COPY<

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.DelChr Method

2/17/2021 • 4 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes chars contained in the which parameter in a string based on the contents on the where parameter. If the where parameter contains an equal-sign, then all occurrences of characters in which is deleted from the current value. If the where parameter contains a less-than, then the characters are only deleted when they are first in the string. If the where parameter contains a greater-than, then the characters are only deleted when they are the last in the string. If the where parameter contains any other char, an exception is thrown. If the where parameter or the which parameter is empty, the source is returned unmodified. The which parameter is to be considered as an array of chars to delete where the order does not matter.

Syntax

```
NewString :=   Text.DelChr(String: String [, Where: String] [, Which: String])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The input string.

Where

Type: String

Specifies where to delete characters.

Which

Type: String

Specifies the characters that you want to delete.

Return Value

NewString Type: String The end result String.

Remarks

The DelChr method is case-sensitive.

If you omit the *Which* parameter, then the method deletes spaces from *String* based on the contents of the *Where* parameter as follows:

- If *Where* contains =, then all the spaces are deleted from *String*.
- If *Where* contains <, then all the spaces at the start of *String* are deleted.
- If *Where* contains >, then all the spaces at the end of *String* are deleted.

- If *Where* contains any other character, then an error is returned.
- If *Where* is empty, then *String* is returned unchanged.

If you use the *Where* and the *Which* parameters, then the method deletes from *String* the characters that are contained in the *Which* parameter based on the contents of the *Where* parameter as follows:

- If *Where* contains =, then every occurrence of the characters in *Which* are deleted from *String*.
- If *Where* contains <, then the characters in *Which* are only deleted if they occur at the start of *String*.
- If *Where* contains >, then the characters in *Which* are deleted only if they occur at the end of *String*.
- If *Where* contains any other character, then an error is returned.
- If *Where* is empty, then *String* is returned unchanged.
- If *Which* is empty, then *String* is returned unchanged.

The *Which* parameter contains an array of the characters that you want to delete. The order of the characters is of no significance. If *String* contains a character that is specified in *Which*, it is deleted from *String*.

Example 1

```
var
    String: Text;
    Where: Text;
    Which: Text;
    NewString: Text;
    Text000: TexConst ENU='Windy West Solutions';
    Text001: TexConst ENU='>%1< is transformed to >%2<';
begin
    String := Text000;
    Where := '<>';
    Which := 'Ws';
    NewString := DelChr(String, Where, Which);
    Message(Text001, String,NewString);
end;
```

The message window displays the following:

>Windy West Solutions< is transformed to >indy West Solution<

The method deletes every W and s that is either the first or last character in *String*.

Example 2

```
var
    String: Text;
    Where: Text;
    Which: Text;
    NewString: Text;
    Text000: TexConst ENU='This is an example';
    Text001: TexConst ENU='>%1< is transformed to >%2<';
begin
    String := Text000;
    Where := '=';
    Which := 'sx';
    NewString := DelChr(String, Where, Which);
    Message(Text001, String,NewString);
end;
```

The message window displays the following:

>This is an example< is transformed to >Thi i an eample<

The method deletes every s and x from *String*.

Example 3

```
var
    String: Text;
    Where: Text;
    Which: Text;
    NewString: Text;
    Text000: TexConst ENU='This is an example';
    Text001: TexConst ENU='>%1< is transformed to >%2<';
begin
    String := Text000;
    Where := '>';
    Which := 'Tely';
    NewString := DelChr(String, Where, Which);
    Message(Text001, String,NewString);
end;
```

The message window displays the following:

>This is an example< is transformed to >This is an examp<

If T, e, l, or y is the last character in *String*, the method deletes them.

Example 4

```
var
    String: Text;
    Where: Text;
    Which: Text;
    NewString: Text;
    Text000: TexConst ENU='This is an example';
    Text001: TexConst ENU='>%1< is transformed to >%2<';
begin
    String := Text000;
    Where := '<';
    Which := 'This ';
    NewString := DelChr(String, Where, Which);
    Message(Text001, String,NewString);
end;
```

The message window displays the following:

>This is an example< is transformed to >an example<

If T, h, s, i, or space is the first character in *String*, the method deletes them.

Example 5

```
var
    String: Text;
    Where: Text;
    Which: Text;
    NewString: Text;
    Text000: TexConst ENU='This is an example';
    Text001: TexConst ENU='>%1< is transformed to >%2<';
begin
    String := Text000;
    Where := '<';
    NewString := DelChr(String, Where);
    Message(Text001, String,NewString);
end;
```

The message window displays the following:

> This is an example< is transformed to >This is an example<

The method removes any spaces from the start of *String*.

Example 6

```
var
    String: Text;
    NewString: Text;
    Text000: Label ' Windy West Solutions '; // note that there can be multiple leading and trailing spaces
    Text001: Label '>%1< is transformed to >%2<';
begin
    String := Text000;
    NewString := DelChr(String);
    Message(Text001, String, NewString);
end;
```

The message window displays the following:

> Windy West Solutions < is transformed to >WindyWestSolutions<

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.DelStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Deletes a substring inside a string (text or code).

Syntax

```
NewString :=   Text.DelStr(String: String, Position: Integer [, Length: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

String
Type: String
The input string.

Position
Type: Integer
The position of the first character that you want to delete. Position must be greater than zero (0). If Position exceeds the length of String, DelStr returns the original string, unchanged.

Length
Type: Integer
Specifies how many characters to delete. Length must be greater than zero (0).

Return Value

NewString Type: String The input string without the specified substring.

Remarks

If you omit *Length*, all the characters starting with *Position* are deleted until the end of the string.

If you omit *Length* and *Position* is less than 1, then an error is returned.

If you omit *Length* and *Position* is greater than the length of *String*, then *String* is returned unchanged.

Example

```
var
    Str: Text[40];
    NewStr: Text[40];
    Position: Integer;
    Length: Integer;
    Text000: TexConst ENU='Adjusting prices - Please wait.';
    Text001: TexConst ENU='The original string:>%1<';
    Text002: TexConst ENU='The original modified:>%2<';
begin
    Str := Text000;
    Position := 11; // Remove the word 'prices' and a blank.
    Length := 7;
    NewStr := DelStr(Str, Position, Length);
    Message(Text001, Str);
    Message(Text002, NewStr);
end;
```

The first message window displays the following:

The original string:

>Adjusting prices - Please wait.<

The second message window displays the following:

The modified string:

>Adjusting - Please wait.<

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.IncStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Increases a positive number or decrease a negative number inside a string by one (1).

Syntax

```
NewString :=   Text.IncStr(String: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string that you want to increase or decrease.

Return Value

NewString Type: String The incremented string.

Remarks

If *String* contains more than one number, then only the number closest to the end of the string is changed. For example, 'A10B20' is changed to 'A10B21' and 'a12b12c' to 'a12b13c'.

If *String* contains a negative number, then it is decreased by one. For example, '-55' is changed to '-56'.

Zero (0) is considered a positive number. Therefore, it is increased it by one. For example, 'A0' is changed to 'A1'.

When *String* contains a number such as 99, it is increased to 100 and the length of the output string is: LEN(String) + 1. For example, 'a12b99c' is changed to 'a12b100c'.

If *String* does not contain any number, the output string is an empty string. For example, 'aaa' is changed to ''.

IncStr only increments integer numbers within strings, not decimals. For example, if you call IncStr on the string **a99.99b** then the result is **a99.100b**.

Example

```
var
    Account: Text[60];
    NegAccount: Text[60];
    EmptyAccount: Text[60];
    MyAccount: Text[60];
    ResultAccount: Text[60];
    ResultNegAccount: Text[60];
    ResultEmptyAccount: Text[60];
    ResultMyAccount: Text[60];
    Text000: TexConst ENU='Account no. 99 does not balance.';
    Text001: TexConst ENU='Account no. 2342 shows a total of $-452.';
    Text002: TexConst ENU='My bank account is empty.';
    Text003: TexConst ENU='My bank account shows a total of $0.';
    Text004: TexConst ENU='The text strings before IncStr is called:\\%1\\%2\\%3\\%4';
    Text005: TexConst ENU='The text strings after IncStr is called:\\%1\\%2\\%3\\%5';
begin
    Account := Text000;
    NegAccount := Text001;
    EmptyAccount := Text002;
    MyAccount := Text003;
    Message(Text004, Account, NegAccount, EmptyAccount, MyAccount);
    ResultAccount := IncStr(Account);
    ResultNegAccount := IncStr(NegAccount);
    ResultEmptyAccount := IncStr(EmptyAccount);
    ResultMyAccount := IncStr(MyAccount);
    Message(Text005, ResultAccount, ResultNegAccount, ResultEmptyAccount, ResultMyAccount);
end;
```

The first message displays the following:

The text strings before IncStr is called:

Account no. 99 does not balance.

Account no. 2342 shows a total of $ -452.

My bank account is empty.

My bank account shows a total of $ 0.

The second message displays the following:

The text strings after IncStr has been called:

Account no. 100 does not balance.

Account no. 2342 shows a total of $ -453.

My bank account shows a total of $ 1.

The example shows that if the string contains more than one number, only the last number is changed. Furthermore, positive numbers and zero are increased and negative numbers are decreased. Finally, if there are no numbers in the string, then an empty string is returned.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.InsStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts a substring into a string.

Syntax

```
NewString :=   Text.InsStr(String: String, SubString: String, Position: Integer)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string into which you want to insert a substring.

SubString

Type: String

The substring that you want to insert into String.

Position

Type: Integer

Specifies where to insert SubString. Position must be greater than or equal to 1. If Position is greater than the length of String, then the result is concatenated and copied to NewString.

Return Value

NewString Type: String The input string including the specified substring

Remarks

If *SubString* is empty, then *String* is returned unchanged.

If *Position* is less than 1, an error is returned.

If *Position* is greater than the length of *String*, *SubString* is added at the end of *String*. For example, `InsStr("Thomas","AAA",999)` returns 'ThomasAAA'.

Example

```
var
    Str: Text[60];
    SubString: Text[60];
    NewString: Text[60];
    Text000: Label 'Press ENTER to continue.';
    Text001: Label 'or ESC';
    Text002: Label ' The test string before InsStr is called:>%1<';
    Text003: Label ' The resulting string after InsStr is called:>%1<';
begin
    Str := Text000;
    SubString := Text001;
    Message(Text002, Str);
    NewString := InsStr(Str, SubString, 13);
    Message(Text003, NewString);
end;
```

The first message window displays the following:

The test string before InsStr is called:

>Press ENTER to continue.<

The second message window displays the following:

The resulting string after InsStr is called:

>Press ENTER or ESC to continue.<

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.LowerCase Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts all letters in a string to lowercase.

Syntax

```
NewString :=   Text.LowerCase(String: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string that you want to convert to lowercase. Only letters in the range A to Z and, if applicable, special language characters are converted.

Return Value

NewString Type: String The string converted to lowercase.

Example

```
var
    Str: Text[60];
    Lower: Text[60];
    Text000: Label 'The Entries are Sorted by Name.';
    Text001: Label 'The string before LowerCase is:>%1<';
    Text002: Label 'The string after LowerCase is:>%1<';
begin
    Str := Text000;
    Message(Text001, Str);
    Lower := LowerCase(Str);
    Message(Text002, Lower);
end;
```

The first message window displays the following:

The string before LowerCase is:

>The Entries are Sorted by Name.<

The second message window displays the following:

The string after LowerCase is:

>the entries are sorted by name.<

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.MaxStrLen Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the maximum defined length of a string variable.

Syntax

```
MaxLength :=   Text.MaxStrLen(String: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string variable for which you want to find the maximum length.

Return Value

MaxLength Type: Integer The maximum length of the string variable.

Remarks

If you call this method on a Variant, it returns an error.

Example

```
var
    City: Text[30];
    MaxLength: Integer;
    Length: Integer;
    Text000: Label 'Vedbaek';
    Text001: Label 'The MaxStrLen method returns %1,\\';
    Text002: Label 'whereas the StrLen method returns %2';
begin
    City := Text000;
    MaxLength := MaxStrLen(City);
    Length := StrLen(City);
    Message(Text001 + Text002, MaxLength, Length);
end;
```

The message window displays the following:

The MaxStrLen method returns 30,

whereas the StrLen method returns 7.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.MaxStrLen Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0 until version 1.0 where it was deprecated.*

Gets the maximum defined length of a variant variable.

Syntax

```
MaxLength :=   Text.MaxStrLen(Variant: Variant)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Variant

Type: Variant

The source variant for which you want to find the maximum length.

Return Value

MaxLength Type: Integer The maximum length of the string variable.

Remarks

If you call this method on a Variant, it returns an error.

Example

```
var
    City: Text[30];
    MaxLength: Integer;
    Length: Integer;
    Text000: Label 'Vedbaek';
    Text001: Label 'The MaxStrLen method returns %1,\\';
    Text002: Label 'whereas the StrLen method returns %2';
begin
    City := Text000;
    MaxLength := MaxStrLen(City);
    Length := StrLen(City);
    Message(Text001 + Text002, MaxLength, Length);
end;
```

The message window displays the following:

The MaxStrLen method returns 30,

whereas the StrLen method returns 7.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.PadStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Changes the length of a string to a specified length. If the string is shorter than the specified length, length spaces are added at the end of the string to match the length. If the string is longer than the specified length, the string is truncated. If the specified length is less than 0, an exception is thrown.

Syntax

```
NewString :=   Text.PadStr(String: String, Length: Integer [, FillCharacter: String])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

String
Type: String
The string that you want to increase or decrease.

Length
Type: Integer
The new length of the output string. If Length is less than the length of String, then String is truncated. Otherwise String is expanded with filler characters. If Length is less than 0, then an error is returned.

FillCharacter
Type: String
This is a string of length 1. This character is used to fill empty spaces at the end of the output string. If not specified, spaces are used as default. If the length of FillCharacter is not 1, an error is returned.

Return Value

NewString Type: String A copy of the string with the expected length.

Remarks

If you omit *FillCharacter* and *String* is shorter than *Length*, then spaces are added at the end of *String* to match *Length*.

If you omit *FillCharacter* and *String* is longer than *Length*, then *String* is truncated.

Example

```
var
    Str1: Text[30];
    Str2: Text[30];
    Len1: Integer;
    Len2: Integer;
    Text000: Label '13 characters';
    Text001: Label 'Four';
    Text002: Label 'Before PadStr is called:\\';
    Text003: Label '>%1< has the length %2\\';
    Text004: Label '>%3< has the length %4\\';
    Text005: Label 'After PadStr is called:\\';
begin
    Str1 := Text000;
    Str2 := Text001;
    Len1 := StrLen(Str1);
    Len2 := StrLen(Str2);
    Message(Text002 + Text003 + Text004, Str1, Len1, Str2, Len2);
    Str1 := PadStr(Str1, 5); // Truncate the length to 5
    Str2 := PadStr(Str2, 15, 'w'); // Concatenate w until length = 15
    Len1 := StrLen(Str1);
    Len2 := StrLen(Str2);
    Message(Text005 + Text003 + Text004, Str1, Len1, Str2, Len2);
end;
```

The first message window displays the following:

Before PadStr is called:

>13 characters< has the length 13

>Four< has the length 4

The second message window displays the following:

After PadStr is called:

>13 ch< has the length 5

>Fourwwwwwwwwwww< has the length 15

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.SelectStr Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves a substring from a comma-separated string.

Syntax

```
NewString :=   Text.SelectStr(Number: Integer, CommaString: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Number

Type: Integer

Specifies which substring to retrieve. The substrings in the comma-separated string are numbered 1, 2, 3, and so on. If Number is greater than the actual number of substrings, then a run-time error occurs.

CommaString

Type: String

A string that contains substrings separated by commas. The maximum length of this string is 391 characters.

Return Value

NewString Type: String The substring from the comma-separated string at the index specified.

Remarks

SelectStr treats string values as OPTIONS. This means that identical values in different strings are not allowed.

Any trailing commas are removed before the operation starts.

If *Number* is less than 1 or greater than the number of real values (excluding trailing commas) in the string, then an error is returned.

Quotes are not supported. For example, a,b,"c,d",e is treated as a five-element substring where substring 4 is d".

Example 1

```
var
    CommaStr: Text[60];
    CommaStr2: Text[60];
    SubStr1: Text[60];
    SubStr2: Text[60];
    SubStr3: Text[60];
    SubStr4: Text[60];
    Text000: Label 'This,is a comma,separated,string';
    Text001: Label 'The calls to SelectStr return:\\';
    Text002: Label '11,22,33,,55,,,';
begin
    CommaStr := Text000;
    CommaStr2 := Text002;
    SubStr1 := SelectStr(2, CommaStr); // Pick out the 2nd substring.
    SubStr2 := SelectStr(4, CommaStr); // Pick out the 4th substring.
    SubStr3 := SelectStr(1, CommaStr2);
    SubStr4 := SelectStr(3, CommaStr2);
    Message(Text001 + '>%1<\' + '>%2<\' + '>%3<\' + '>%4<\', SubStr1, SubStr2, SubStr3, SubStr4);
end;
```

The message window displays the following text:

The calls to SelectStr return:

>is a comma<

>string<

>11<

>33<

Example 2

```
var
    CommaStr2: Text[60];
    SubStr5: Text[60];
    Text002: Label '11,22,33,,55,,,';
begin
    CommaStr2 := Text002;
    SubStr5 := SelectStr(6,CommaStr2);
    Message('>%1<\', SubStr5);
end;
```

This example returns an error.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.StrCheckSum Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Calculates a checksum for a string that contains a number. If the source is empty, 0 is returned. Each char in the source and in the weight must be a numeric character 0-9, otherwise an exception is thrown. If the WeightString parameter is shorter then the source, it is padded with '1' up until the length of source. If the WeightString parameter is longer than the source, an exception is thrown.

Syntax

```
CheckNumber :=   Text.StrCheckSum(String: String [, WeightString: String] [, Modulus: Integer])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

This string contains the number for which you want to calculate a checksum. You can only enter the numeric characters 0-9 in this string. If you enter anything else, a run-time error will occur. If String is empty, 0 is returned.

WeightString

Type: String

This string contains numbers that you want to use as weights when calculating the checksum. The default value is a string that contains StrLen(String) '1'-characters.

Modulus

Type: Integer

The number that you want to use in the checksum formula. The default value is 10.

Return Value

CheckNumber Type: Integer The resulting checksum value.

Example 1

This example shows how to use the StrCheckSum method to calculate a checksum.

```
var
    StrNumber: Text[30];
    Weight: Text[30];
    Modulus: Integer;
    Text000: Label 'The number: %1\\';
    Text001: Label 'has the checksum: %2';
begin
    StrNumber := '4378';
    Weight := '1234';
    Modulus := 7;
    CheckSum := StrCheckSum(StrNumber, Weight, Modulus);
    Message(Text000 + Text001, StrNumber, CheckSum);
end;
```

The formula is:

(7 - (4x1 + 3x2 + 7x3 + 8x4) MOD 7) MOD 7=0

The message window displays the following:

The number: 4378

has the checksum: 0

Example 2

This example shows how to use the StrCheckSum method to calculate a modulus 10 checksum for a bar code.

The StrCheckSum method can be used to calculate checksums for 13- and 8-digit European Article Number (EAN) and EAN-compatible bar codes such as a Universal Product Code (UPC) or Japanese Article Number (JAN).

A 13-digit EAN code has the following format:

1. The 12 digits in positions 13 to 2 are used to calculate the checksum at position 1.
2. Starting with position 2, all even values are totaled. The result is then multiplied by three. This value is called Even.
3. Starting with position 3, all odd values are totaled. The result is called Odd.
4. Total=Even + Odd.
5. The modulus 10 checksum is then (10 - Total MOD 10) MOD 10.

```
var
    StrNumber: Text[30];
    Weight: Text[30];
    Modulus: Integer;
    Text000: Label 'The EAN code: %1\\';
    Text001: Label 'has the checksum: %2';
begin
    StrNumber := '577622135746';
    Weight := '131313131313';
    CheckSum := StrCheckSum(StrNumber, Weight);
    Message(Text000 + Text001, StrNumber, CheckSum);
end;
```

The message window displays the following:

The EAN code: 577622135746

has the checksum: 3

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.StrLen Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the length of a string you define.

Syntax

```
Length :=   Text.StrLen(String: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

String
Type: String
The string for which you want to determine the length.

Return Value

Length Type: Integer The length of the string.

Remarks

The difference between the StrLen method and the MaxStrLen Method (Code, Text) is that the StrLen returns the actual number of characters in the input string, whereas MaxStrLen returns the maximum defined length of the input string.

In Dynamics 365 Business Central, if you call StrLen on a Variant, then you get an error that the contents of the parameter are not valid. In earlier versions of Dynamics 365, if you call StrLen on a Variant, then 0 is returned.

Example

This example shows the difference between the StrLen and the MaxStrLen methods.

```
var
    City: Text[30];
    MaxLength: Integer;
    Length: Integer;
    Text000: Label 'Atlanta';
    Text001: Label 'The MaxStrLen method returns: %1,\\';
    Text002: Label 'whereas the StrLen method returns: %2';
begin
    City := Text000;
    MaxLength := MaxStrLen(City);
    Length := StrLen(City);
    Message(Text001 + Text002, MaxLength, Length);
end;
```

The message window displays the following:

The MaxStrLen method returns: 30

whereas the StrLen method returns: 7

This shows that the MaxLength method returns the maximum possible length according to the definition of the string variable, whereas StrLen returns the actual length of the text.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.StrPos Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Searches for the first occurrence of substring inside a string.

Syntax

```
Position :=   Text.StrPos(String: String, SubString: String)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

String
Type: String
The string in which you want to search.

SubString
Type: String
The substring for which you want to search.

Return Value

Position Type: Integer The one-based index of the first occurrence of the substring inside the string.

Remarks

The StrPos method returns the position of the first occurrence of the substring.

If *SubString* cannot be found, then the method returns zero.

If *String* or *SubString* is empty, then the method returns zero.

Example 1

This example shows how to use the StrPos method.

```
var
    Text000: Label 'ABC abc abc xy';
    Text001: Label 'abc';
    Text002: Label 'The search for the substring: >%1<\\';
    Text003: Label 'in the string: >%2<,\\';
    Text004: Label 'returns the position: %3';
begin
    String := Text000;
    SubStr := Text001
    Pos := StrPos(String, SubStr);
    Message(Text002 + Text003 + Text004, SubStr, String, Pos);
    // The StrPos method is case-sensitive. Furthermore, it only
    // returns the position of the 1st occurrence of the substring.
end;
```

The message window displays the following:

The search for the substring: >abc<

in the string: >ABC abc abc xy<

returns the position: 5

Example 2

```
Pos1 := StrPos("abc",""); // Returns 0.
Pos2 := StrPos("abc","c"); // Returns 3.
Pos3 := StrPos("abc","bc"); // Returns 2.
Pos4 := StrPos("abc","x"); // Returns 0.
```

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.StrSubstNo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces %1, %2, %3... and #1, #2, #3... fields in a string with the values you provide as optional parameters.

Syntax

```
NewString :=   Text.StrSubstNo(String: String [, Value1: Any,...])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

A string containing '#' and/or '%' fields.

Value1

Type: Any

One or more values (expressions) that you want to insert into String. You can specify up to 10 values.

Return Value

NewString Type: String Returns a new string with the provided values inserted into the specified string.

Remarks

This method replaces the numbered specifiers in a string with a string representation of the equivalent numbered value.

The specifiers in the string can be either %*n* or #######*n*, where *n* represents a 1-based number. When you use the # specifier, the number can be anywhere in the specifier. For example, ###2#### is allowed.

The %*n* specifier is replaced with the string representation of the value with their full lengths.

The ##*n* specifier is replaced with the same number of characters as the length of the specifier (including the number). The values are truncated to the length of the # field.

If the string representation is shorter than the length of the specifier, then it is left aligned.

For example, `StrSubstNo('Test %1 >#2##< >#3<', 1,2,3)` returns "Test 1 >2 < >3 <".

In this example, the following substitutions are made:

- %1 is replaced by '1' because a % field is replaced by the specified value in its full length.
- '#2##' is replaced by '2 space space space' because the value is shorter than the field and therefore, the 2

is left aligned and the field is four characters long.

- #3 is replaced by '3 space' because the 3 is left aligned and the field is two characters long

If the string is longer, then asterisks are inserted to indicate overflow.

For example, `StrSubstNo('Test %1 >###2< >#3<', 'Thomas','Thomas',0)` returns "Test Thomas >****< >0 <".

In this example the following substitutions are made:

- %1 is replaced by 'Thomas' because a %1 field is replaced by the specified value in its full length.
- '###2' is replaced by '****' because the string 'Thomas' is longer than the ###2 field. Each character in the field is replaced by an asterisk.
- #3 is replaced by '0 space' because the 0 is left aligned and the field is two characters long.

You can have several references to the same value.

For example, `StrSubstNo('Test %1 %3 %1', 555, 666, 777)` returns "Test 555 777 555".

If one of the values is null, then it is treated as an empty string.

Example

The following example shows how to use the StrSubstNo method.

```
var
    Str: Text[1024];
    AccountNo: Integer;
    Balance: Decimal;
    Text000: Label 'The balance of account %1 is $ %2';
    Text001: Label 'The test string before StrSubstNo is called:\\%1';
    Text002: Label 'The string after StrSubstNo is called:\\%1';
begin
    Str := Text000;
    AccountNo := 3452;
    Balance := 2345 + 5462;
    Message(Text001, Str);
    Str := StrSubstNo(Str, AccountNo, Balance);
    Message(Text002, Str);
end;
```

The first message window displays the following text:

The string before StrSubstNo has been called:

The balance of account %1 is $ %2

The second message window displays the following text:

The string after StrSubstNo has been called:

The balance of account 3452 is $ 7,807

> **NOTE**
> This example is run on a computer that has the regional format set to English (United States).

See Also

Text.UpperCase Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts all letters in a string to uppercase.

Syntax

```
NewString :=   Text.UpperCase(String: String)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

String

Type: String

The string that you want to convert to uppercase.

Return Value

NewString Type: String The string converted to uppercase.

Example

The following example shows how to use the **UpperCase** method.

```
var
    Text000: Label 'Outstanding Order Status';
    Text001: Label 'The test string before UpperCase is called:\\%1';
    Text002: Label 'The string after UpperCase is called:\\%1';
begin
    Lower := Text000;
    Message(Text001, Lower);
    Upper := UpperCase(Lower);
    Message(Text002, Upper);
end;
```

The first message window displays the following:

The test-string before UpperCase is called:

Outstanding Order Status

The second message window displays the following:

The string after UpperCase is called:

OUTSTANDING ORDER STATUS

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Contains Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a value indicating whether a specified substring occurs within this string.

Syntax

```
Ok :=   Text.Contains(Value: Text)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Value
Type: Text
The string to seek.

Return Value

Ok Type: Boolean **true** if the specified substring occurs in the specified string, otherwise **false**.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.EndsWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the end of this string instance matches the specified string.

Syntax

```
Ok :=   Text.EndsWith(Value: Text)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Value
Type: Text
The string to compare to the substring at the end of this instance.

Return Value

Ok Type: Boolean **true** if the end of this string instance matches the specified string, otherwise **false**.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.IndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reports the one-based index of the first occurrence of the specified string in this instance.

Syntax

```
Index :=   Text.IndexOf(Value: Text [, StartIndex: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Value
Type: Text
The string to seek.

StartIndex
Type: Integer
The one-based search starting position.

Return Value

Index Type: Integer The one-based index of the first occurrence of the specified string in this instance. If the index returned is 0, the value is not present in the string.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.IndexOfAny Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reports the one-based index of the first occurrence of the specified string in this instance. The search starts at a specified character position.

Syntax

```
Index :=   Text.IndexOfAny(Values: Text [, StartIndex: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Values
Type: Text
The collection of characters to seek.

StartIndex
Type: Integer
The one-based search starting position.

Return Value

Index Type: Integer The one-based index of the first occurrence of the specified string in this instance. If the index returned is 0, the value is not present in the string.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.IndexOfAny Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reports the one-based index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position.

Syntax

```
Index :=   Text.IndexOfAny(Values: List of [Char] [, StartIndex: Integer])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Values

Type: List of [Char]

The collection of characters to seek.

StartIndex

Type: Integer

The one-based search starting position.

Return Value

Index Type: Integer The one-based index of the first occurrence of the specified string in this instance. If the index returned is 0, the value is not present in the string.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.LastIndexOf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reports the one-based index position of the last occurrence of a specified string in this instance.

Syntax

```
Index :=   Text.LastIndexOf(Value: Text [, StartIndex: Integer])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Value

Type: Text

The string to seek.

StartIndex

Type: Integer

The search starting position. The search proceeds from startIndex toward the beginning of this instance.

Return Value

Index Type: Integer The one-based index of the last occurrence of the specified string in this instance. If the index returned is 0, the value is not present in the string.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.PadLeft Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a new Text that right-aligns the characters in this instance by padding them on the left, for a specified total length.

Syntax

```
Result :=   Text.PadLeft(Count: Integer [, Char: Char])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Count
Type: Integer
The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

Char
Type: Char
A padding character.

Return Value

Result Type: Text The end result Text.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.PadRight Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.

Syntax

```
Result :=   Text.PadRight(Count: Integer [, Char: Char])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Count

Type: Integer

The number of characters in the resulting string, equal to the number of original characters plus any additional padding characters.

Char

Type: Char

A padding character.

Return Value

Result Type: Text The end result Text.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a new Text in which a specified number of characters from the current string are deleted.

Syntax

```
Result :=   Text.Remove(StartIndex: Integer [, Count: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

StartIndex
Type: Integer
The one-based position to begin deleting characters.

Count
Type: Integer
The number of characters to delete.

Return Value

Result Type: Text The end result Text.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a new Text in which all occurrences of a specified string in the current instance are replaced with another specified string.

Syntax

```
Result :=   Text.Replace(OldValue: Text, NewValue: Text)
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

OldValue
Type: Text
The string to replace all occurrences of OldValue.

NewValue
Type: Text
The string to be replaced.

Return Value

Result Type: Text The end result Text.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Split Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Splits a string into a maximum number of substrings based on a collection of separators.

Syntax

```
Result :=   Text.Split([Separators: Text,...])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Separators
Type: Text
A collection of separators that delimit the substrings in this string.

Return Value

Result Type: List of [Text] The collection of substrings from the original string based on the collection of separators.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Split Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Splits a string into a maximum number of substrings based on a collection of separators.

Syntax

```
Result :=   Text.Split(Separators: List of [Text])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Separators
Type: List of [Text]
A collection of separators that delimit the substrings in this string.

Return Value

Result Type: List of [Text] The collection of substrings from the original string based on the collection of separators.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Split Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Splits a string into a maximum number of substrings based on a collection of separators.

Syntax

```
Result :=   Text.Split(Separators: List of [Char])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Separators
Type: List of [Char]
A collection of separators that delimit the substrings in this string.

Return Value

Result Type: List of [Text] The collection of substrings from the original string based on the collection of separators.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.StartsWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines whether the beginning of this instance matches a specified string.

Syntax

```
Ok :=   Text.StartsWith(Value: Text)
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Value

Type: Text

The string to compare.

Return Value

Ok Type: Boolean **true** if the beginning of this instance matches the specified string, otherwise **false**.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Substring Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Retrieves a substring from this instance.

Syntax

```
Substring :=   Text.Substring(StartIndex: Integer [, Count: Integer])
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

StartIndex
Type: Integer
The one-based starting character position of a substring in this instance.

Count
Type: Integer
The number of characters in the substring.

Return Value

Substring Type: Text The substring extracted from this instance.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.ToLower Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a copy of this string converted to lowercase.

Syntax

```
Result :=   Text.ToLower()
```

> **NOTE**
> This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Return Value

Result Type: Text A copy of this string converted to lowercase.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.ToUpper Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a copy of this string converted to uppercase.

Syntax

```
Result :=   Text.ToUpper()
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Return Value

Result Type: Text A copy of this string converted to uppercase.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.Trim Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns a new Text in which all leading and trailing white-space characters from the current Text object are removed.

Syntax

```
Result :=   Text.Trim()
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Return Value

Result Type: Text A copy of this string without all leading and trailing white-space characters.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.TrimEnd Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all trailing occurrences of a set of characters specified in an array from the current Text object.

Syntax

```
Result :=   Text.TrimEnd([Chars: Text])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Chars

Type: Text

A string containing the characters to remove.

Return Value

Result Type: Text A copy of this string without all trailing white-space characters.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

Text.TrimStart Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all leading occurrences of a set of characters specified in an array from the current Text object.

Syntax

```
Result :=   Text.TrimStart([Chars: Text])
```

NOTE

This method can be invoked without specifying the data type name.

Parameters

Text Type: Text An instance of the Text data type.

Chars

Type: Text

A string containing the characters to remove.

Return Value

Result Type: Text A copy of this string without all leading white-space characters.

See Also

Text Data Type
Getting Started with AL
Developing Extensions

TextConst Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes a multi-language string constant.

Remarks

The `TextConst` data type is typically used for UI messages; process or error messages. Keeping the `TextConst` data type in global scope, makes it easier to reuse the same message for several situations. For information about naming, see CodeCop Rule AA0074.

> **IMPORTANT**
>
> The `TextConst` data type is not included in the .xlf files for translation. Make sure to use the Label Data Type instead.

Example

The data type can be declared with the syntax as shown in the example below.

```
codeunit 50100 MyCodeunit
{
    procedure MyProcedure()
    var
        localTextConst: TextConst ENU = 'My text', DAN = 'Min tekst';
    begin
        Message(localTextConst);
    end;

    var
        globalTextConst: TextConst ENU = 'My text', DAN = 'Min tekst';
}
```

See Also

Getting Started with AL
Developing Extensions

TextBuilder Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a lighweight wrapper for the .Net implementation of StringBuilder.

The following methods are available on instances of the TextBuilder data type.

METHOD NAME	DESCRIPTION
Append(Text)	Appends a copy of the specified string to this TextBuilder instance.
AppendLine([Text])	Appends a copy of the specified string followed by the default line terminator to the end of the current TextBuilder object. If this parameter is omitted, only the line terminator will be appended.
Capacity([Integer])	Gets or sets the maximum number of characters that can be contained in the memory allocated by the current instance.
Clear()	Removes all characters from the current TextBuilder instance.
EnsureCapacity(Integer)	Ensures that the capacity of this TextBuilder instance is at least the specified value.
Insert(Integer, Text)	Inserts a string into this TextBuilder instance at the specified character position.
Length([Integer])	Gets or sets the length of this TextBuilder instance.
MaxCapacity()	Gets the maximum capacity of this TextBuilder instance.
Remove(Integer, Integer)	Removes the specified range of characters from this TextBuilder instance.
Replace(Text, Text)	Replaces all occurrences of a specified string in this TextBuilder instance with another specified string.
Replace(Text, Text, Integer, Integer)	Replaces, within a substring of this instance, all occurrences of a specified string in this TextBuilder instance with another specified string.
ToText()	Converts the value of this TextBuilder instance to a Text.
ToText(Integer, Integer)	Converts the value of a substring of this TextBuilder instance to a Text.

Remarks

The **TextBuilder** data type is one-based indexed, that is, the indexing begins with 1.

The Text Data Type is a value type, such that every time you use a method on it, you create a new string object in memory. This requires a new allocation of space. In situations where you need to perform repeated modifications to a string, the overhead associated with creating a Text Data Type can be costly.

The **TextBuilder** data type is a reference type, which holds a pointer elsewhere in memory. For performance reasons, we recommend you to use it when you want to modify a string without creating a new object. For example, using **TextBuilder** data type can boost performance when concatenating many strings together in a loop.

See Also

Getting Started with AL
Developing Extensions
Text Data Type

TextBuilder.Append Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Appends a copy of the specified string to this TextBuilder instance.

Syntax

```
[Ok := ]  TextBuilder.Append(Text: Text)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

Text
Type: Text
The text to append.

Return Value

Ok Type: Boolean **true** if the copy of the specified string succeeded, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.AppendLine Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Appends a copy of the specified string followed by the default line terminator to the end of the current TextBuilder object. If this parameter is omitted, only the line terminator will be appended.

Syntax

```
[Ok := ]  TextBuilder.AppendLine([Text: Text])
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

Text
Type: Text
The string to append.

Return Value

Ok Type: Boolean **true** if the copy of the specified string with the default line terminator succeeded, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Capacity Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the maximum number of characters that can be contained in the memory allocated by the current instance.

Syntax

```
[OldCapacity := ]  TextBuilder.Capacity([NewCapacity: Integer])
```

NOTE

This method can be invoked using property access syntax.

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

NewCapacity

Type: Integer

The maximum number of characters that can be contained in the memory allocated by the current instance. Its value can range from Length to MaxCapacity.

Return Value

OldCapacity Type: Integer The maximum number of characters that can be contained in the memory allocated by the current instance.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Clear Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all characters from the current TextBuilder instance.

Syntax

```
TextBuilder.Clear()
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.EnsureCapacity Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Ensures that the capacity of this TextBuilder instance is at least the specified value.

Syntax

```
[Ok := ]  TextBuilder.EnsureCapacity(NewCapacity: Integer)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

NewCapacity
Type: Integer
The minimum capacity to ensure.

Return Value

Ok Type: Boolean **true** if the capacity of the TextBuilder is at least the specified value, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Insert Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Inserts a string into this TextBuilder instance at the specified character position.

Syntax

```
[Ok := ]  TextBuilder.Insert(Position: Integer, Text: Text)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

Position
Type: Integer
The position in this TextBuilder instance where insertion begins.

Text
Type: Text
The string to insert.

Return Value

Ok Type: Boolean **true** if the insertion of the specified succeeded, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Length Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the length of this TextBuilder instance.

Syntax

```
[OldLength := ]  TextBuilder.Length([NewLength: Integer])
```

NOTE

This method can be invoked using property access syntax.

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

NewLength

Type: Integer

The new length of this TextBuilder instance.

Return Value

OldLength Type: Integer The length of this TextBuilder instance.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.MaxCapacity Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the maximum capacity of this TextBuilder instance.

Syntax

```
MaxCapacity :=   TextBuilder.MaxCapacity()
```

NOTE

This method can be invoked using property access syntax.

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

Return Value

MaxCapacity Type: Integer The maximum capacity of this TextBuilder instance.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified range of characters from this TextBuilder instance.

Syntax

```
[Ok := ]  TextBuilder.Remove(StartIndex: Integer, Count: Integer)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

StartIndex
Type: Integer
The one-based position in this TextBuilder instance where removal begins.

Count
Type: Integer
The number of characters to remove.

Return Value

Ok Type: Boolean **true** if the specified range of characters was successfully removed, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces all occurrences of a specified string in this TextBuilder instance with another specified string.

Syntax

```
[Ok := ]  TextBuilder.Replace(OldText: Text, NewText: Text)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

OldText
Type: Text
The string to replace.

NewText
Type: Text
The string that replaces OldText.

Return Value

Ok Type: Boolean **true** if all occurrences of a specified string were successfully replaced, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.Replace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces, within a substring of this instance, all occurrences of a specified string in this TextBuilder instance with another specified string.

Syntax

```
[Ok := ]  TextBuilder.Replace(OldText: Text, NewText: Text, StartIndex: Integer, Count: Integer)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

OldText
Type: Text
The string to replace.

NewText
Type: Text
The string that replaces OldText.

StartIndex
Type: Integer
The position in this TextBuilder instance where the substring begins.

Count
Type: Integer
The length of the substring.

Return Value

Ok Type: Boolean **true** if all occurrences of a specified string were successfully replaced, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.ToText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value of this TextBuilder instance to a Text.

Syntax

```
Result :=   TextBuilder.ToText()
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

Return Value

Result Type: Text The result text.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

TextBuilder.ToText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the value of a substring of this TextBuilder instance to a Text.

Syntax

```
Result :=   TextBuilder.ToText(StartIndex: Integer, Count: Integer)
```

Parameters

TextBuilder Type: TextBuilder An instance of the TextBuilder data type.

StartIndex
Type: Integer
The starting position of the substring in this TextBuilder instance.

Count
Type: Integer
The number of characters in the substring.

Return Value

Result Type: Text The result text substring.

See Also

TextBuilder Data Type
Getting Started with AL
Developing Extensions

Time Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Denotes a time ranging from 00:00:00.000 to 23:59:59.999. An undefined or blank time is specified by 0T.

The displayed text format of the time is determined by your Regional and Language Options in Windows.

The following are examples of valid assignments of times to a Time variable *MyTime*. Time must be set by specifying hours, minutes, and seconds.

```
MyTime := 0T;
MyTime := 115900T;
Message(Format(MyTime));
MyTime := 115934T;
Message(Format(MyTime));
MyTime := 115934.444T;
Message(Format(MyTime));
MyTime := 235900T;
Message(Format(MyTime));
MyTime := 030000T;
Message(Format(MyTime));
```

The following shows what the message windows display accordingly on a computer with the regional format set to English (United States) for the syntax examples above.

11:59:00 AM

11:59:34 AM

11:59:34.444 AM

11:59:00 PM

3:00:00 AM

SQL Server

Microsoft SQL Server stores information about both date and time in columns of the DATETime type. Dynamics 365 uses only the time part and inserts a constant value for the date: 01-01-1754.

The Dynamics 365 undefined time is represented by the same value as an undefined date. The undefined date is represented by the earliest valid DateTime in SQL Server, which is 01-01-1753 00:00:00:000.

See Also

Getting Started with AL
Developing Extensions

Variant Data Type

2/17/2021 • 3 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an AL variable object. The AL variant data type can contain many AL data types.

The following methods are available on instances of the Variant data type.

METHOD NAME	DESCRIPTION
IsAction()	Indicates whether an AL variant contains an Action variable.
IsAutomation()	Indicates whether an AL variant contains an Automation variable.
IsBigInteger()	Indicates whether an AL variant contains a BigInteger variable.
IsBinary()	Indicates whether an AL variant contains a Binary variable.
IsBoolean()	Indicates whether an AL variant contains a Boolean variable.
IsByte()	Indicates whether an AL variant contains a Byte data type variable.
IsChar()	Indicates whether an AL variant contains a Char variable.
IsClientType()	Indicates whether an AL variant contains a ClientType variable.
IsCode()	Indicates whether an AL variant contains a Code variable.
IsCodeunit()	Indicates whether an AL variant contains a Codeunit variable.
IsDataClassificationType()	Indicates whether a AL variant contains a DataClassification variable.
IsDate()	Indicates whether an AL variant contains a Date variable.
IsDateFormula()	Indicates whether an AL variant contains a DateFormula variable.
IsDateTime()	Indicates whether an AL variant contains a DateTime variable.
IsDecimal()	Indicates whether an AL variant contains a Decimal variable.
IsDefaultLayout()	Indicates whether an AL variant contains a DefaultLayout variable.
IsDotNet()	Indicates whether an AL variant contains a DotNet variable.
IsDuration()	Indicates whether an AL variant contains a Duration variable.
IsExecutionMode()	Indicates whether an AL variant contains an ExecutionMode variable.

METHOD NAME	DESCRIPTION
IsFieldRef()	Indicates whether an AL variant contains a FieldRef variable.
IsFile()	Indicates whether an AL variant contains a File variable.
IsFilterPageBuilder()	Indicates whether an AL variant contains a FilterPageBuilder variable.
IsGuid()	Indicates whether an AL variant contains a Guid variable.
IsInStream()	Indicates whether an AL variant contains an InStream variable.
IsInteger()	Indicates whether an AL variant contains an Integer variable.
IsJsonArray()	Indicates whether an AL variant contains a JsonArray variable.
IsJsonObject()	Indicates whether an AL variant contains a JsonObject variable.
IsJsonToken()	Indicates whether an AL variant contains a JsonToken variable.
IsJsonValue()	Indicates whether an AL variant contains a JsonValue variable.
IsNotification()	Indicates whether an AL variant contains a Notification variable.
IsObjectType()	Indicates whether an AL variant contains an ObjectType variable.
IsOption()	Indicates whether an AL variant contains an Option variable.
IsOutStream()	Indicates whether an AL variant contains an OutStream variable.
IsRecord()	Indicates whether an AL variant contains a Record variable.
IsRecordId()	Indicates whether an AL variant contains a RecordId variable.
IsRecordRef()	Indicates whether an AL variant contains a RecordRef variable.
IsReportFormat()	Indicates whether an AL variant contains a ReportFormat variable.
IsSecurityFiltering()	Indicates whether an AL variant contains a SecurityFiltering variable.
IsTableConnectionType()	Indicates whether an AL variant contains a TableConnectionType variable.
IsTestPermissions()	Indicates whether an AL variant contains a TestPermissions variable.
IsText()	Indicates whether an AL variant contains a Text variable.

METHOD NAME	DESCRIPTION
IsTextBuilder()	Indicates whether an AL variant contains a TextBuilder variable.
IsTextConstant()	Indicates whether an AL variant contains a Text constant.
IsTextEncoding()	Indicates whether an AL variant contains a TextEncoding variable.
IsTime()	Indicates whether an AL variant contains a Time variable.
IsTransactionType()	Indicates whether an AL variant contains a TransactionType variable.
IsWideChar()	Indicates whether an AL variant contains a WideChar variable.
IsXmlAttribute()	Indicates whether an AL variant contains an XmlAttribute variable.
IsXmlAttributeCollection()	Indicates whether an AL variant contains an XmlAttributeCollection variable.
IsXmlCData()	Indicates whether an AL variant contains an XmlCData variable.
IsXmlComment()	Indicates whether an AL variant contains an XmlComment variable.
IsXmlDeclaration()	Indicates whether an AL variant contains an XmlDeclaration variable.
IsXmlDocument()	Indicates whether an AL variant contains an XmlDocument variable.
IsXmlDocumentType()	Indicates whether an AL variant contains an XmlDocumentType variable.
IsXmlElement()	Indicates whether an AL variant contains an XmlElement variable.
IsXmlNamespaceManager()	Indicates whether an AL variant contains an XmlNamespaceManager variable.
IsXmlNameTable()	Indicates whether an AL variant contains an XmlNameTable variable.
IsXmlNode()	Indicates whether an AL variant contains an XmlNode variable.
IsXmlNodeList()	Indicates whether an AL variant contains an XmlNodeList variable.
IsXmlProcessingInstruction()	Indicates whether an AL variant contains an XmlProcessingInstruction variable.
IsXmlReadOptions()	Indicates whether an AL variant contains an XmlReadOptions variable.
IsXmlText()	Indicates whether an AL variant contains an XmlText variable.

METHOD NAME	DESCRIPTION
IsXmlWriteOptions()	Indicates whether an AL variant contains an XmlWriteOptions variable.

See Also

Getting Started with AL
Developing Extensions

Variant.IsAction Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an Action variable.

Syntax

```
Ok :=   Variant.IsAction()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an Action variable, otherwise **false**.

Example

The following example determines whether an AL variant contains an Action variable. The MyAction variable is assigned to the variant variable that is named MyVariant. The **IsAction** method determines whether the variant contains an Action variable and stores the return value in the varResult variable. In this case, the variant contains an Action variable so **Yes** is returned and displayed in a message box. The IsCode Method (Variant) determines whether the variant contains a code variable. The return value is **No** because the variant does not contain a code.

```
var
    MyAction: Action;
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant contain an Action variable? %1.';
    Text001: Label 'Does the variant- contain a code variable? %1.';
begin
    MyVariant := MyAction;
    varResult := MyVariant.IsAction;
    Message(Text000,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001, varResult);
end;
```

See Also

Variant Data Type

Variant.IsAutomation Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an Automation variable.

Syntax

```
Ok :=   Variant.IsAutomation()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an Automation variable, otherwise **false**.

Example

The following example determines whether an AL variant contains an Automation variable. The MyAutomation variable is assigned to the variant variable that is named MyVariant. The **IsAutomation** method determines whether the variant contains an Automation variable and stores the return value in the varResult variable. In this case, the variant contains an Automation variable so **Yes** is returned and displayed in a message box. The IsCode Method (Variant) determines whether the variant contains a code variable. The return value is **No** because the variant does not contain a code.

```
var
    MyAutomation: Automation "AFormAut 1.0 Type Library";
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant contain an Automation variable? %1.';
    Text001: Label 'Does the variant- contain a code variable? %1.';
begin
    MyVariant := MyAutomation;
    varResult := MyVariant.IsAutomation;
    Message(Text000,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001, varResult);
end;
```

See Also

Variant Data Type

Variant.IsBigInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a BigInteger variable.

Syntax

```
Ok :=   Variant.IsBigInteger()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a BigInteger variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsBinary Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Binary variable.

Syntax

```
Ok :=   Variant.IsBinary()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Binary variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsBoolean Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Boolean variable.

Syntax

```
Ok :=   Variant.IsBoolean()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Boolean variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a Boolean variable. The code initializes the MyBoolean variable with a Boolean value. The MyBoolean variable is assigned to the variant variable that is named MyVariant. The **IsBoolean** method determines whether the variant contains a Boolean variable and stores the return value in the varResult variable. In this case, the variant contains a Boolean variable so **true** is returned and displayed in a message box. The Boolean value is obtained from the **Critical** field in the **Item** table. The IsCode Method (Variant) determines whether the variant contains a code variable. The return value is **false** because the variant does not contain a code.

```
var
    ItemRec: Record Item;
    MyBoolean: Boolean;
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant >%1< contain a Boolean variable? %2.';
    Text001: Label 'Does the variant >%1< contain a code variable? %2.';
begin
    MyBoolean := ItemRec.Critical;
    MyVariant := MyBoolean;
    varResult := MyVariant.IsBoolean;
    Message(Text000,MyVariant,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001,MyVariant,varResult);
end;
```

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsByte Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Byte data type variable.

Syntax

```
Ok :=   Variant.IsByte()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Byte variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a Byte data type variable. The code initializes the MyByte data type variable with the constant text string **A**. The MyByte variable is assigned to the variant variable that is named MyVariant. The **IsByte** method determines whether the variant contains a Byte variable and stores the return value in the varResult variable. In this case, the variant contains a Byte variable so **true** is returned and displayed in a message box.

```
var
    MyByte: Byte;
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant contain a byte variable? %1.';
begin
    MyByte := 'A';
    MyVariant := MyByte;
    varResult := MyVariant.IsByte;
    Message(Text000,varResult);
end;
```

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsChar Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Char variable.

Syntax

```
Ok :=   Variant.IsChar()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Char variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsClientType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a ClientType variable.

Syntax

```
Ok :=   Variant.IsClientType()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a ClientType variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Code variable.

Syntax

```
Ok :=   Variant.IsCode()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Code variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a code variable. The code initializes the MyCode variable with a string value. The MyCode variable is assigned to the variant variable that is named MyVariant. The **IsCode** method determines whether the variant contains a code variable and stores the return value in the varResult variable. In this case, the variant contains a code variable so **true** is returned and displayed in a message box. The IsText Method (Variant) determines whether the variant contains a text variable. The return value is **false** because the variant does not contain a text.

```
var
    MyCode: Code[100];
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant >%1< contain a code variable? %2.';
    Text001: Label 'Does the variant >%1< contain a text variable? %2.';
begin
    MyCode := 'A1297';
    MyVariant :=  MyCode;
    varResult := MyVariant.IsCode;
    Message(Text000,MyVariant,varResult);
    varResult := MyVariant.IsText;
    Message(Text001,MyVariant,varResult);
end;
```

See Also

Variant Data Type

Variant.IsCodeunit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Codeunit variable.

Syntax

```
Ok :=   Variant.IsCodeunit()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Codeunit variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDataClassificationType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether a AL variant contains a DataClassification variable.

Syntax

```
Ok :=   Variant.IsDataClassificationType()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a DataClassification variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDate Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Date variable.

Syntax

```
Ok :=   Variant.IsDate()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Date variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a Date variable. The code initializes the MyDate variable with a Date value. The MyDate variable is assigned to the variant variable that is named MyVariant. The **IsDate** method determines whether the variant contains a Date variable and stores the return value in the varResult variable. In this case, the variant contains a Date variable so **true** is returned and displayed in a message box. The IsCode Method (Variant) determines whether the variant contains a Code variable. The return value is **false** because the variant does not contain a code.

```
var
    MyDate: Date;
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant >%1< contain a date variable? %2.';
    Text001: Label 'Does the variant >%1< contain a code variable? %2.';
begin
    MyDate := Today;
    MyVariant :=  MyDate;
    varResult := MyVariant.IsDate;
    Message(Text000,MyVariant,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001,MyVariant,varResult);
end;
```

See Also

Variant Data Type

Variant.IsDateFormula Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a DateFormula variable.

Syntax

```
Ok :=   Variant.IsDateFormula()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a DateFormula variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDateTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a DateTime variable.

Syntax

```
Ok :=   Variant.IsDateTime()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a DateTime variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDecimal Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Decimal variable.

Syntax

```
Ok :=   Variant.IsDecimal()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Decimal variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDefaultLayout Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a DefaultLayout variable.

Syntax

```
Ok :=   Variant.IsDefaultLayout()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a DefaultLayout variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDotNet Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a DotNet variable.

Syntax

```
Ok :=   Variant.IsDotNet()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a DotNet variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsDuration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Duration variable.

Syntax

```
Ok :=   Variant.IsDuration()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Duration variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsExecutionMode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an ExecutionMode variable.

Syntax

```
Ok :=   Variant.IsExecutionMode()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an ExecutionMode variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsFieldRef Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a FieldRef variable.

Syntax

```
Ok :=   Variant.IsFieldRef()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a FieldRef variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsFile Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a File variable.

Syntax

```
Ok :=   Variant.IsFile()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a File variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsFilterPageBuilder Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a FilterPageBuilder variable.

Syntax

```
Ok :=   Variant.IsFilterPageBuilder()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a FilterPageBuilder variable, otherwise **false**.

Example

The following example uses the IsFilterPageBuilder method on a filter page object that includes a filter control for the **Date** system table.

```
var
    varDateItem: Text;
    varFilterPageBuilder: FilterPageBuilder;
    myVariant: Variant;
begin
    varDateItem := 'Date record';
    varFilterPageBuilder.AddTable(varDateItem + ‘ 1’,DATABASE::Date);
    myVariant := varFilterPageBuilder;
    if not myVariant.IsFilterPageBuilder then
      ERROR(‘This variant should contain a FilterPageBuilder variable’);
end;
```

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsGuid Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Guid variable.

Syntax

```
Ok :=   Variant.IsGuid()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Guid variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsInStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an InStream variable.

Syntax

```
Ok :=   Variant.IsInStream()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an InStream variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsInteger Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an Integer variable.

Syntax

```
Ok :=   Variant.IsInteger()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an Integer variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsJsonArray Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a JsonArray variable.

Syntax

```
Ok :=   Variant.IsJsonArray()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a JsonArray variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsJsonObject Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a JsonObject variable.

Syntax

```
Ok :=   Variant.IsJsonObject()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a JsonObject variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsJsonToken Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a JsonToken variable.

Syntax

```
Ok :=   Variant.IsJsonToken()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a JsonToken variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsJsonValue Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a JsonValue variable.

Syntax

```
Ok :=   Variant.IsJsonValue()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a JsonValue variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsNotification Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Notification variable.

Syntax

```
Ok :=   Variant.IsNotification()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Notification variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsObjectType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an ObjectType variable.

Syntax

```
Ok :=   Variant.IsObjectType()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a ObjectType variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsOption Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an Option variable.

Syntax

```
Ok :=   Variant.IsOption()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an Option variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsOutStream Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an OutStream variable.

Syntax

```
Ok :=   Variant.IsOutStream()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an OutStream variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsRecord Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Record variable.

Syntax

```
Ok :=   Variant.IsRecord()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Record variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a record variable. The Get method gets customer number 10000 from the **Customer** table. The record is stored in the MyRecord variable. The MyRecord variable is assigned to the variant variable that is named MyVariant. The **ISRecord** method determines whether the variant contains a Record variable and stores the return value in the varResult variable. In this case, the variant contains a Record variable so **true** is returned and displayed in a message box. The IsCode Method (Variant)determines whether the variant contains a code variable. The return value is **false** because the variant does not contain a code.

```
var
    MyRecord: Record Customer;
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant >%1< contain a record variable? %2.';
    Text001: Label 'Does the variant >%1< contain a code variable? %2.';
begin
    MyRecord.Get('10000');
    MyVariant := MyRecord;
    varResult := MyVariant.IsRecord;
    Message(Text000,MyVariant,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001,MyVariant,varResult);
end;
```

See Also

Variant.IsRecordId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a RecordId variable.

Syntax

```
Ok :=   Variant.IsRecordId()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a RecordId variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsRecordRef Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a RecordRef variable.

Syntax

```
Ok :=   Variant.IsRecordRef()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a RecordRef variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsReportFormat Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a ReportFormat variable.

Syntax

```
Ok :=   Variant.IsReportFormat()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a ReportFormat variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsSecurityFiltering Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a SecurityFiltering variable.

Syntax

```
Ok :=   Variant.IsSecurityFiltering()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a SecurityFiltering variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsTableConnectionType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a TableConnectionType variable.

Syntax

```
Ok :=   Variant.IsTableConnectionType()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a TableConnectionType variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsTestPermissions Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a TestPermissions variable.

Syntax

```
Ok :=   Variant.IsTestPermissions()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a TestPermissions variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Text variable.

Syntax

```
Ok :=   Variant.IsText()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Text variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a text variable. The code initializes the MyText variable with a text value. The MyText variable is assigned to the variant variable that is named MyVariant. The **IsText** method determines whether the variant contains a text variable and stores the return value in the varResult variable. In this case, the variant contains a text variable so **Yes** is returned and displayed in a message box. The **IsCode** method determines whether the variant contains a code variable. The return value is **No** because the variant does not contain a code.

```
var
    MyText: Text[50];
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant >%1< contain a text variable? %2.';
    Text001: Label 'Does the variant >%1< contain a code variable? %2.';
begin
    MyText := 'This is some text';
    MyVariant :=  MyText;
    varResult := MyVariant.IsText;
    Message(Text000,MyVariant,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001,MyVariant,varResult);
end;
```

See Also

Variant Data Type

Variant.IsTextBuilder Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a TextBuilder variable.

Syntax

```
Ok :=   Variant.IsTextBuilder()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a TextBuilder variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsTextConstant Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Text constant.

Syntax

```
Ok :=   Variant.IsTextConstant()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Text constant, otherwise **false**.

Example

The following example determines whether an AL variant contains a text constant. The code assigns the Text000 text constant to the variant variable that is named MyVariant. The **IsTextCONSTANT** method determines whether the variant contains a text constant and stores the return value in the varResult variable. In this case, the variant contains a text constant so **Yes** is returned and displayed in a message box.

```
var
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'This is some text.";
    Text001: Label 'DOes the variant contain a text constant? %1.';
begin
    MyVariant := Text000;
    varResult := MyVariant.IsTextConstant;
    Message(Text001,MyVariant,varResult);
    Message(Text001,varResult);
end;
```

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsTextEncoding Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a TextEncoding variable.

Syntax

```
Ok :=   Variant.IsTextEncoding()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a TextEncoding variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsTime Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a Time variable.

Syntax

```
Ok :=   Variant.IsTime()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a Time variable, otherwise **false**.

Example

The following example determines whether an AL variant contains a time variable. The code initializes the MyTime variable with a Time value. The MyTime variable is assigned to the variant variable that is named MyVariant. The **IsTime** method determines whether the variant contains a Time variable and stores the return value in the varResult variable. In this case, the variant contains a Time variable so **Yes** is returned and displayed in a message box. The IsCode Method (Variant) determines whether the variant contains a Code variable. The return value is **No** because the variant does not contain a code.

```
var
    MyTime: Time;
    MyVariant: Variant;
    varResult: Boolean;
    Text000: Label 'Does the variant >%1\< contain a time variable? %2.';
    Text001: Label 'Does the variant >%1\< contain a code variable? %2.';
begin
    MyTime := Time;
    MyVariant := MyTime;
    varResult := MyVariant.IsTime;
    Message(Text000,MyVariant,varResult);
    varResult := MyVariant.IsCode;
    Message(Text001,MyVariant,varResult);
end;
```

See Also

Variant Data Type

Variant.IsTransactionType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a TransactionType variable.

Syntax

```
Ok :=   Variant.IsTransactionType()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a TransactionType variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsWideChar Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains a WideChar variable.

Syntax

```
Ok :=   Variant.IsWideChar()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains a WideChar variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlAttribute variable.

Syntax

```
Ok :=   Variant.IsXmlAttribute()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlAttribute variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlAttributeCollection Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlAttributeCollection variable.

Syntax

```
Ok :=   Variant.IsXmlAttributeCollection()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlAttributeCollection variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlCData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlCData variable.

Syntax

```
Ok :=   Variant.IsXmlCData()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlCData variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlComment Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlComment variable.

Syntax

```
Ok :=   Variant.IsXmlComment()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlComment variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlDeclaration variable.

Syntax

```
Ok :=   Variant.IsXmlDeclaration()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlDeclaration variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlDocument variable.

Syntax

```
Ok :=   Variant.IsXmlDocument()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlDocument variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlDocumentType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlDocumentType variable.

Syntax

```
Ok :=   Variant.IsXmlDocumentType()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlDocumentType variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlElement Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlElement variable.

Syntax

```
Ok :=   Variant.IsXmlElement()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlElement variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlNamespaceManager Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlNamespaceManager variable.

Syntax

```
Ok :=   Variant.IsXmlNamespaceManager()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlNamespaceManager variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlNameTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlNameTable variable.

Syntax

```
Ok :=   Variant.IsXmlNameTable()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlNameTable variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlNode variable.

Syntax

```
Ok :=   Variant.IsXmlNode()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlNode variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlNodeList Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlNodeList variable.

Syntax

```
Ok :=   Variant.IsXmlNodeList()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlNodeList variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlProcessingInstruction Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlProcessingInstruction variable.

Syntax

```
Ok :=   Variant.IsXmlProcessingInstruction()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlProcessingInstruction variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlReadOptions Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlReadOptions variable.

Syntax

```
Ok :=   Variant.IsXmlReadOptions()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlReadOptions variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlText variable.

Syntax

```
Ok :=   Variant.IsXmlText()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlText variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Variant.IsXmlWriteOptions Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Indicates whether an AL variant contains an XmlWriteOptions variable.

Syntax

```
Ok :=   Variant.IsXmlWriteOptions()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Variant Type: Variant An instance of the Variant data type.

Return Value

Ok Type: Boolean **true** if the AL variant contains an XmlWriteOptions variable, otherwise **false**.

See Also

Variant Data Type
Getting Started with AL
Developing Extensions

Version Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a version matching the format: Major.Minor.Build.Revision .

The following methods are available on the Version data type.

METHOD NAME	DESCRIPTION
Create(String)	Creates a version object from the provided string. The string should be in the format W.X.Y.Z, where W, X, Y and Z represent positive integers and where Y and Z are optional. If the input string is not in the expected format, an exception is thrown.
Create(Integer, Integer [, Integer] [, Integer])	Creates a version object from the major, minor, build and revision numbers provided.

The following methods are available on instances of the Version data type.

METHOD NAME	DESCRIPTION
Build()	Gets the build number of the version.
Major()	Gets the major number of the version.
Minor()	Gets the minor number of the version.
Revision()	Gets the revision number from the version.

See Also

Getting Started with AL
Developing Extensions

Version.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a version object from the provided string. The string should be in the format W.X.Y.Z, where W, X, Y and Z represent positive integers and where Y and Z are optional. If the input string is not in the expected format, an exception is thrown.

Syntax

```
Value :=   Version.Create(Version: String)
```

Parameters

Version
Type: String
The string to convert into a version object.

Return Value

Value Type: Version The version created from the provided string.

See Also

Version Data Type
Getting Started with AL
Developing Extensions

Version.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates a version object from the major, minor, build and revision numbers provided.

Syntax

```
Value :=   Version.Create(Major: Integer, Minor: Integer [, Build: Integer] [, Revision: Integer])
```

Parameters

Major

Type: Integer

The major version number.

Minor

Type: Integer

The minor version number.

Build

Type: Integer

The build version number.

Revision

Type: Integer

The revision version number.

Return Value

Value Type: Version The version created from the provided major, minor, build and revision numbers.

See Also

Version Data Type
Getting Started with AL
Developing Extensions

Version.Build Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the build number of the version.

Syntax

```
BuildVersion :=   Version.Build()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Version Type: Version An instance of the Version data type.

Return Value

BuildVersion Type: Integer The build version number of the version.

See Also

Version Data Type
Getting Started with AL
Developing Extensions

Version.Major Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the major number of the version.

Syntax

```
MajorVersion :=   Version.Major()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Version Type: Version An instance of the Version data type.

Return Value

MajorVersion Type: Integer The major version number of the version.

See Also

Version Data Type
Getting Started with AL
Developing Extensions

Version.Minor Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the minor number of the version.

Syntax

```
MinorVersion :=   Version.Minor()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Version Type: Version An instance of the Version data type.

Return Value

MinorVersion Type: Integer The minor version number of the version.

See Also

Version Data Type
Getting Started with AL
Developing Extensions

Version.Revision Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the revision number from the version.

Syntax

```
RevisionVersion :=   Version.Revision()
```

NOTE

This method can be invoked using property access syntax.

Parameters

Version Type: Version An instance of the Version data type.

Return Value

RevisionVersion Type: Integer The revision version number of the version.

See Also

Version Data Type
Getting Started with AL
Developing Extensions

WebServiceActionContext Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Represents an AL WebServiceActionContext.

The following methods are available on instances of the WebServiceActionContext data type.

METHOD NAME	DESCRIPTION
AddEntityKey(Integer, Any)	Add a new <fieldId, value> pair to the collection of entity keys.
GetObjectId()	Gets the object ID.
GetObjectType()	Gets the object type.
GetResultCode()	Gets the web service action result status code.
SetObjectId(Integer)	Sets the object ID.
SetObjectType(ObjectType)	Sets the object type.
SetResultCode(WebServiceActionResultCode)	Sets the web service action result status code.

See Also

Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.AddEntityKey Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Add a new <fieldId, value> pair to the collection of entity keys.

Syntax

```
[Ok := ]  WebServiceActionContext.AddEntityKey(FieldId: Integer, FieldValue: Any)
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

FieldId
Type: Integer
The field ID of the entity key.

FieldValue
Type: Any
The value for the field in the entity key.

Return Value

Ok Type: Boolean **true** if adding the entity key succeeded, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

```
actionContext.AddEntityKey(Rec.FieldNO(Id), ToSalesHeader.Id);
```

For a complete code example, see Creating and Interacting with an OData V4 Bound Action.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.GetObjectId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the object ID.

Syntax

```
ObjectId :=   WebServiceActionContext.GetObjectId()
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

Return Value

ObjectId Type: Integer The object ID.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.GetObjectType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the object type.

Syntax

```
ObjectType :=   WebServiceActionContext.GetObjectType()
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

Return Value

ObjectType Type: ObjectType The object type.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.GetResultCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Gets the web service action result status code.

Syntax

```
ResultCode :=   WebServiceActionContext.GetResultCode()
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

Return Value

ResultCode Type: WebServiceActionResultCode The web service action result status code.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.SetObjectId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Sets the object ID.

Syntax

```
WebServiceActionContext.SetObjectId(ObjectId: Integer)
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

ObjectId
Type: Integer
The new object ID.

Example

```
actionContext.SetObjectId(Page::SalesInvoiceCopy);
```

For a complete code example, see Creating and Interacting with an OData V4 Bound Action.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.SetObjectType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Sets the object type.

Syntax

```
WebServiceActionContext.SetObjectType(ObjectType: ObjectType)
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

ObjectType
Type: ObjectType
The new object type.

Example

```
actionContext.SetObjectType(ObjectType::Page);
```

For a complete code example, see Creating and Interacting with an OData V4 Bound Action.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

WebServiceActionContext.SetResultCode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 2.0.*

Sets the web service action result status code.

Syntax

```
WebServiceActionContext.SetResultCode(ResultCode: WebServiceActionResultCode)
```

Parameters

WebServiceActionContext Type: WebServiceActionContext An instance of the WebServiceActionContext data type.

ResultCode
Type: WebServiceActionResultCode
The new web service action result status code.

Example

```
...
// Set the result code to inform the caller that an item was created.
actionContext.SetResultCode(WebServiceActionResultCode::Created);
...
```

For a complete code example, see Creating and Interacting with an OData V4 Bound Action.

See Also

WebServiceActionContext Data Type
Creating and Interacting with an OData V4 Bound Action
Getting Started with AL
Developing Extensions

XmlAttribute Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an XML attribute.

The following methods are available on the XmlAttribute data type.

METHOD NAME	DESCRIPTION
Create(String, String)	Creates an XmlAttribute node.
Create(String, String, String)	Creates an XmlAttribute node.
CreateNamespaceDeclaration(String, String)	Creates an attribute that represents a namespace declaration.

The following methods are available on instances of the XmlAttribute data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
IsNamespaceDeclaration()	Determines if this attribute is a namespace declaration.
LocalName()	Gets the local name of the attribute.
Name()	The qualified name of the attribute.
NamespacePrefix()	Gets the prefix of the attribute (if any).
NamespaceUri()	Gets the namespace URI of the attribute.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.

METHOD NAME	DESCRIPTION
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
Value([String])	Gets or sets the value of the attribute.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlAttribute.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlAttribute node.

Syntax

```
XmlAttribute :=   XmlAttribute.Create(Name: String, Value: String)
```

Parameters

Name

Type: String

The qualified name of the attribute. If the name is of the form {{namespace}}localName, it will be qualified with the given namespace.

Value

Type: String

The value of the attribute.

Return Value

XmlAttribute Type: XmlAttribute The created XmlAttribute node.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlAttribute node.

Syntax

```
XmlAttribute :=   XmlAttribute.Create(LocalName: String, NamespaceUri: String, Value: String)
```

Parameters

LocalName

Type: String

The local name of the attribute.

NamespaceUri

Type: String

The namespace URI of the attribute.

Value

Type: String

The value of the attribute.

Return Value

XmlAttribute Type: XmlAttribute The created XmlAttribute node.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.CreateNamespaceDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an attribute that represents a namespace declaration.

Syntax

```
XmlAttribute :=   XmlAttribute.CreateNamespaceDeclaration(Prefix: String, NamespaceUri: String)
```

Parameters

Prefix

Type: String

The prefix of the attribute (if any).

NamespaceUri

Type: String

The URI of the attribute. If the prefix is xmlns, then this parameter must be http://www.w3.org/2000/xmlns/; otherwise an exception is thrown.

Return Value

XmlAttribute Type: XmlAttribute The created XmlAttribute node.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlAttribute.AddAfterSelf(Content: Any,...)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlAttribute.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlAttribute.AsXmlNode()
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlAttribute.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlAttribute.GetDocument(var Document: XmlDocument)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlAttribute.GetParent(var Parent: XmlElement)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.IsNamespaceDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Determines if this attribute is a namespace declaration.

Syntax

```
IsNamespaceDeclaration :=   XmlAttribute.IsNamespaceDeclaration()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

IsNamespaceDeclaration Type: Boolean **true** if the attribute represents a namespace declaration, otherwise **false**.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.LocalName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the local name of the attribute.

Syntax

```
LocalName :=   XmlAttribute.LocalName()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

LocalName Type: String The local name of the attribute.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

The qualified name of the attribute.

Syntax

```
Name :=   XmlAttribute.Name()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

Name Type: String The qualified name of the node.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.NamespacePrefix Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the prefix of the attribute (if any).

Syntax

```
NamespacePrefix :=   XmlAttribute.NamespacePrefix()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

NamespacePrefix Type: String The prefix of the attribute (if any).

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.NamespaceUri Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the namespace URI of the attribute.

Syntax

```
NamespaceUri :=   XmlAttribute.NamespaceUri()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

NamespaceUri Type: String The namespace URI of the attribute.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlAttribute.Remove()
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlAttribute.ReplaceWith(Node: Any,...)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlAttribute.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

XPath
Type: String
The XPath expression.
NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlAttribute.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlAttribute.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlAttribute.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node:
XmlNode)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.Value Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the value of the attribute.

Syntax

```
[Value := ]  XmlAttribute.Value([NewValue: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

NewValue

Type: String

The new value of the attribute.

Return Value

Value Type: String The value of the attribute.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlAttribute.WriteTo(OutStream: OutStream)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlAttribute.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlAttribute.WriteTo(var Text: Text)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttribute.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlAttribute.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlAttribute Type: XmlAttribute An instance of the XmlAttribute data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttribute Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a collection of XML attributes.

The following methods are available on instances of the XmlAttributeCollection data type.

METHOD NAME	DESCRIPTION
Count()	Gets the number of attributes in the XmlAttributeCollection.
Get(Integer, var XmlAttribute)	Gets the specified attribute.
Get(String, var XmlAttribute)	Gets the specified attribute.
Get(String, String, var XmlAttribute)	Gets the specified attribute.
Remove(XmlAttribute)	Removes the specified attribute from the collection.
Remove(String)	Removes the specified attribute from the collection.
Remove(String, String)	Removes the specified attribute from the collection.
RemoveAll()	Removes all attributes from the collection.
Set(String, String)	Sets the value of the specified attribute or creates it if is not part of the collection.
Set(String, String, String)	Sets the value of the specified attribute or creates it if is not part of the collection.

See Also

Getting Started with AL
Developing Extensions

XmlAttributeCollection.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of attributes in the XmlAttributeCollection.

Syntax

```
Count :=   XmlAttributeCollection.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

Return Value

Count Type: Integer The number of attributes in the XmlAttributeCollection.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the specified attribute.

Syntax

```
[Ok := ]  XmlAttributeCollection.Get(Index: Integer, var Result: XmlAttribute)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

Index
Type: Integer
The index of the attribute to retrieve.

Result
Type: XmlAttribute
Variable containing the requested XmlAttribute if the operation is successful.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the specified attribute.

Syntax

```
[Ok := ]  XmlAttributeCollection.Get(Name: String, var Result: XmlAttribute)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

Name
Type: String
The qualified name of the attribute to retrieve.

Result
Type: XmlAttribute
Variable containing the requested XmlAttribute if the operation is successful.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the specified attribute.

Syntax

```
[Ok := ]  XmlAttributeCollection.Get(LocalName: String, NamespaceUri: String, var Result: XmlAttribute)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

LocalName
Type: String
The local name of the attribute to retrieve.

NamespaceUri
Type: String
The namespace URI of the attribute to retrieve.

Result
Type: XmlAttribute
Variable containing the requested XmlAttribute if the operation is successful.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified attribute from the collection.

Syntax

```
XmlAttributeCollection.Remove(Attribute: XmlAttribute)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

Attribute
Type: XmlAttribute
The attribute to remove.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified attribute from the collection.

Syntax

```
XmlAttributeCollection.Remove(Name: String)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

Name

Type: String

The qualified name of the attribute to remove.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified attribute from the collection.

Syntax

```
XmlAttributeCollection.Remove(LocalName: String, NamespaceUri: String)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

LocalName

Type: String

The local name of the attribute to remove.

NamespaceUri

Type: String

The namespace URI of the attribute to remove.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.RemoveAll Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes all attributes from the collection.

Syntax

```
XmlAttributeCollection.RemoveAll()
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the value of the specified attribute or creates it if is not part of the collection.

Syntax

```
XmlAttributeCollection.Set(Name: String, Value: String)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

Name

Type: String

The fully qualified name of the attribute to set.

Value

Type: String

The value to set for the attribute.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlAttributeCollection.Set Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the value of the specified attribute or creates it if is not part of the collection.

Syntax

```
XmlAttributeCollection.Set(LocalName: String, NamespaceUri: String, Value: String)
```

Parameters

XmlAttributeCollection Type: XmlAttributeCollection An instance of the XmlAttributeCollection data type.

LocalName
Type: String
The local name of the attribute to set.

NamespaceUri
Type: String
The namespace URI of the attribute to set.

Value
Type: String
The value to set for the attribute.

See Also

XmlAttributeCollection Data Type
Getting Started with AL
Developing Extensions

XmlCData Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a CData section.

The following methods are available on the XmlCData data type.

METHOD NAME	DESCRIPTION
Create(String)	Creates an XmlCData node.

The following methods are available on instances of the XmlCData data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
Value([String])	Gets or sets the value of this node.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.

METHOD NAME	DESCRIPTION
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlCData.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlCData node.

Syntax

```
CDataNode :=   XmlCData.Create(Value: String)
```

Parameters

Value

Type: String

A string that contains the value of the new XmlCData node.

Return Value

CDataNode Type: XmlCData The created XmlCData node.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlCData.AddAfterSelf(Content: Any,...)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlCData.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlCData.AsXmlNode()
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlCData.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlCData.GetDocument(var Document: XmlDocument)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlCData.GetParent(var Parent: XmlElement)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlCData.Remove()
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlCData.ReplaceWith(Node: Any,...)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlCData.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlCData.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlCData.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlCData.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node: XmlNode)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.Value Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the value of this node.

Syntax

```
[Value := ]  XmlCData.Value([NewValue: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

NewValue

Type: String

The new value of this node.

Return Value

Value Type: String The value of this node.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlCData.WriteTo(OutStream: OutStream)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlCData.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlCData.WriteTo(var Text: Text)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlCData.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlCData.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlCData Type: XmlCData An instance of the XmlCData data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlCData Data Type
Getting Started with AL
Developing Extensions

XmlComment Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an XML comment.

The following methods are available on the XmlComment data type.

METHOD NAME	DESCRIPTION
Create(String)	Creates an XmlComment node.

The following methods are available on instances of the XmlComment data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
Value([String])	Gets or sets the string value of this comment.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.

METHOD NAME	DESCRIPTION
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlComment.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlComment node.

Syntax

```
Comment :=   XmlComment.Create(Value: String)
```

Parameters

Value

Type: String

A string that contains the contents of the new XmlComment node.

Return Value

Comment Type: XmlComment The created XmlComment node.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlComment.AddAfterSelf(Content: Any,...)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlComment.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlComment.AsXmlNode()
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlComment.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlComment.GetDocument(var Document: XmlDocument)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlComment.GetParent(var Parent: XmlElement)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlComment.Remove()
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlComment.ReplaceWith(Node: Any,...)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlComment.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlComment.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlComment.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlComment.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node:
XmlNode)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.Value Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the string value of this comment.

Syntax

```
[Value := ]  XmlComment.Value([NewValue: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

NewValue
Type: String
The new value of this comment.

Return Value

Value Type: String The new value of this comment.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlComment.WriteTo(OutStream: OutStream)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlComment.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlComment.WriteTo(var Text: Text)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlComment.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlComment.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlComment Type: XmlComment An instance of the XmlComment data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlComment Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an XML declaration.

The following methods are available on the XmlDeclaration data type.

METHOD NAME	DESCRIPTION
Create(String, String, String)	Creates an XmlDeclaration node.

The following methods are available on instances of the XmlDeclaration data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
Encoding([String])	Gets or sets the encoding of the XML document.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
Standalone([String])	Gets or sets the standalone property for this document.
Version([String])	Gets or sets the version property for this document.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.

METHOD NAME	DESCRIPTION
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlDeclaration.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDeclaration node.

Syntax

```
XmlDeclaration :=   XmlDeclaration.Create(Version: String, Encoding: String, Standalone: String)
```

Parameters

Version

Type: String

The version of the XML, usually "1.0".

Encoding

Type: String

The encoding for the XML document.

Standalone

Type: String

A string containing "yes" or "no" that specifies whether the XML is standalone or requires external entities to be resolved.

Return Value

XmlDeclaration Type: XmlDeclaration The created XmlDeclaration node.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlDeclaration.AddAfterSelf(Content: Any,...)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlDeclaration.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlDeclaration.AsXmlNode()
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlDeclaration.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.Encoding Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the encoding of the XML document.

Syntax

```
[Value := ]  XmlDeclaration.Encoding([NewValue: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

NewValue

Type: String

The new value for the encoding of the XML document.

Return Value

Value Type: String The encoding of the XML document.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlDeclaration.GetDocument(var Document: XmlDocument)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlDeclaration.GetParent(var Parent: XmlElement)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlDeclaration.Remove()
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlDeclaration.ReplaceWith(Node: Any,...)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlDeclaration.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlDeclaration.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlDeclaration.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlDeclaration.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node:
XmlNode)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.Standalone Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the standalone property for this document.

Syntax

```
[Value := ]  XmlDeclaration.Standalone([NewValue: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

NewValue

Type: String

A string containing the standalone property for this document.

Return Value

Value Type: String The standalone property for this document.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.Version Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the version property for this document.

Syntax

```
[Value := ]  XmlDeclaration.Version([NewValue: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

NewValue

Type: String

A string containing the version property for this document.

Return Value

Value Type: String The version property for this document.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDeclaration.WriteTo(OutStream: OutStream)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDeclaration.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDeclaration.WriteTo(var Text: Text)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDeclaration.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDeclaration.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlDeclaration Type: XmlDeclaration An instance of the XmlDeclaration data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDeclaration Data Type
Getting Started with AL
Developing Extensions

XmlDocument Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an XML document.

The following methods are available on the XmlDocument data type.

METHOD NAME	DESCRIPTION
Create()	Creates an XmlDocument.
Create(Any,...)	Creates an XmlDocument.
ReadFrom(String, var XmlDocument)	Reads and parses the XML document from the given data source.
ReadFrom(String, XmlReadOptions, var XmlDocument)	Reads and parses the XML document from the given data source.
ReadFrom(InStream, var XmlDocument)	Reads and parses the XML document from the given data source.
ReadFrom(InStream, XmlReadOptions, var XmlDocument)	Reads and parses the XML document from the given data source.

The following methods are available on instances of the XmlDocument data type.

METHOD NAME	DESCRIPTION
Add(Any,...)	Adds the specified content as a child of this document.
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AddFirst(Any,...)	Adds the specified content at the start of the child list of this document.
AsXmlNode()	Converts the node to an XmlNode.
GetChildElements()	Gets a list containing the child elements for this document, in document order.
GetChildElements(String)	Gets a list containing the child elements for this document, in document order.
GetChildElements(String, String)	Gets a list containing the child elements for this document, in document order.
GetChildNodes()	Gets a list containing the child elements for this document, in document order.

METHOD NAME	DESCRIPTION
GetDeclaration(var XmlDeclaration)	Gets the XML declaration for this document.
GetDescendantElements()	Gets a list containing the descendant elements for this document, in document order.
GetDescendantElements(String)	Gets a list containing the descendant elements for this document, in document order.
GetDescendantElements(String, String)	Gets a list containing the descendant elements for this document, in document order.
GetDescendantNodes()	Gets a list containing the descendant nodes for this document, in document order.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetDocumentType(var XmlDocumentType)	Gets the Document Type Definition (DTD) for this document.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
GetRoot(var XmlElement)	Gets the root element of the XML tree for this document.
NameTable()	Gets the XmlNameTable associated with this document.
Remove()	Removes this node from its parent element.
RemoveNodes()	Removes the child nodes from this document.
ReplaceNodes(Any,...)	Replaces the children nodes of this document with the specified content.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SetDeclaration(XmlDeclaration)	Sets the XML declaration for this document.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlDocument.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDocument.

Syntax

```
XmlDocument :=   XmlDocument.Create()
```

Return Value

XmlDocument Type: XmlDocument The created XmlDocument node.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDocument.

Syntax

```
XmlDocument :=   XmlDocument.Create(Content: Any,...)
```

Parameters

Content

Type: Any

The content to add to this document.

Return Value

XmlDocument Type: XmlDocument The created XmlDocument node.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads and parses the XML document from the given data source.

Syntax

```
[Ok := ]  XmlDocument.ReadFrom(Text: String, var Result: XmlDocument)
```

Parameters

Text

Type: String

A string containing an XML document.

Result

Type: XmlDocument

The XmlDocument parsed from the given data source.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads and parses the XML document from the given data source.

Syntax

```
[Ok := ]  XmlDocument.ReadFrom(Text: String, ReadOptions: XmlReadOptions, var Result: XmlDocument)
```

Parameters

Text

Type: String

A string containing an XML document.

ReadOptions

Type: XmlReadOptions

Specifies options for customizing how the document is parsed.

Result

Type: XmlDocument

The XmlDocument parsed from the given data source.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads and parses the XML document from the given data source.

Syntax

```
[Ok := ]  XmlDocument.ReadFrom(InStream: InStream, var Result: XmlDocument)
```

Parameters

InStream

Type: InStream

A stream containing an XML document.

Result

Type: XmlDocument

The XmlDocument parsed from the given data source.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.ReadFrom Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads and parses the XML document from the given data source.

Syntax

```
[Ok := ]  XmlDocument.ReadFrom(InStream: InStream, ReadOptions: XmlReadOptions, var Result: XmlDocument)
```

Parameters

InStream

Type: InStream

A stream containing an XML document.

ReadOptions

Type: XmlReadOptions

Specifies options for customizing how the document is parsed.

Result

Type: XmlDocument

The XmlDocument parsed from the given data source.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content as a child of this document.

Syntax

```
[Ok := ]  XmlDocument.Add(Content: Any,...)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Content
Type: Any
The content to be added as a child of this document.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlDocument.AddAfterSelf(Content: Any,...)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlDocument.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.AddFirst Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content at the start of the child list of this document.

Syntax

```
[Ok := ]  XmlDocument.AddFirst(Content: Any,...)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Content
Type: Any
The content to be added as a child of this document.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlDocument.AsXmlNode()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlDocument.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetChildElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this document, in document order.

Syntax

```
ChildElements :=   XmlDocument.GetChildElements()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

ChildElements Type: XmlNodeList A list containing the child elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetChildElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this document, in document order.

Syntax

```
ChildElements :=   XmlDocument.GetChildElements(Name: String)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Name
Type: String
The fully qualified name of the elements to retrieve.

Return Value

ChildElements Type: XmlNodeList A list containing the child elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetChildElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this document, in document order.

Syntax

```
ChildElements :=   XmlDocument.GetChildElements(LocalName: String, NamespaceUri: String)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

LocalName
Type: String
The local name of the elements to retrieve.

NamespaceUri
Type: String
The namespace URI of the elements to retrieve.

Return Value

ChildElements Type: XmlNodeList A list containing the child elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetChildNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this document, in document order.

Syntax

```
ChildNodes :=   XmlDocument.GetChildNodes()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

ChildNodes Type: XmlNodeList A list containing the child elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XML declaration for this document.

Syntax

```
[Ok := ]  XmlDocument.GetDeclaration(var Result: XmlDeclaration)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Result
Type: XmlDeclaration
The XML declaration for this document.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDescendantElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant elements for this document, in document order.

Syntax

```
DescendantElements :=   XmlDocument.GetDescendantElements()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

DescendantElements Type: XmlNodeList A list containing the descendant elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDescendantElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant elements for this document, in document order.

Syntax

```
DescendantElements :=   XmlDocument.GetDescendantElements(Name: String)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Name
Type: String
The fully qualified name of the elements to retrieve.

Return Value

DescendantElements Type: XmlNodeList A list containing the descendant elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDescendantElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant elements for this document, in document order.

Syntax

```
DescendantElements :=   XmlDocument.GetDescendantElements(LocalName: String, NamespaceUri: String)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

LocalName
Type: String
The local name of the elements to retrieve.

NamespaceUri
Type: String
The namespace URI of the elements to retrieve.

Return Value

DescendantElements Type: XmlNodeList A list containing the descendant elements for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDescendantNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant nodes for this document, in document order.

Syntax

```
DescendantNodes :=   XmlDocument.GetDescendantNodes()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

DescendantNodes Type: XmlNodeList A list containing the descendant nodes for this document, in document order.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlDocument.GetDocument(var Document: XmlDocument)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetDocumentType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the Document Type Definition (DTD) for this document.

Syntax

```
[Ok := ]  XmlDocument.GetDocumentType(var DocumentType: XmlDocumentType)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

DocumentType
Type: XmlDocumentType
The Document Type Definition (DTD) for this document.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlDocument.GetParent(var Parent: XmlElement)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.GetRoot Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the root element of the XML tree for this document.

Syntax

```
[Ok := ]  XmlDocument.GetRoot(var Result: XmlElement)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Result
Type: XmlElement
The root element of the XML tree for this document.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.NameTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlNameTable associated with this document.

Syntax

```
NameTable :=   XmlDocument.NameTable()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

NameTable Type: XmlNameTable The XmlNameTable associated with this document.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlDocument.Remove()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.RemoveNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the child nodes from this document.

Syntax

```
XmlDocument.RemoveNodes()
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.ReplaceNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the children nodes of this document with the specified content.

Syntax

```
[Ok := ]  XmlDocument.ReplaceNodes(Content: Any,...)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Content
Type: Any
The content that replaces the children nodes.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlDocument.ReplaceWith(Node: Any,...)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlDocument.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlDocument.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlDocument.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlDocument.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node:
XmlNode)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.SetDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the XML declaration for this document.

Syntax

```
[Ok := ]  XmlDocument.SetDeclaration(Declaration: XmlDeclaration)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Declaration
Type: XmlDeclaration
The new value of the XML declaration of this document.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocument.WriteTo(OutStream: OutStream)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

Example

The following example illustrates how to create a Stream from a Blob and write to a Stream from an XML document.

```
pageextension 50100 CustomerListExt extends "Customer List"
{
    trigger OnOpenPage();
    var
        xmlDoc: XmlDocument;
        xmlDec: XmlDeclaration;
        xmlElem: XmlElement;
        xmlElem2: XmlElement;
        TempBlob: Record TempBlob Temporary;
        outStr: OutStream;
        inStr: InStream;
        TempFile: File;
        fileName: Text;
    begin
        xmlDoc := xmlDocument.Create();
        xmlDec := xmlDeclaration.Create('1.0', 'UTF-8', '');
        xmlDoc.SetDeclaration(xmlDec);

        xmlElem := xmlElement.Create('root');
        xmlElem.SetAttribute('release', '2.1');

        xmlElem2 := XmlElement.Create('FirstName');
        xmlElem2.Add(xmlText.Create('Max'));

        xmlElem.Add(xmlElem2);
        xmlDoc.Add(xmlElem);

        // Create an outStream from the Blob, notice the encoding.
        TempBlob.CreateOutStream(outStr, TextEncoding::UTF8);

        // Write the contents of the doc to the stream
        xmlDoc.WriteTo(outStr);

         // From the same Blob, that now contains the XML document, create an inStr
        TempBlob.CreateInStream(inStr, TextEncoding::UTF8);

        // Save the data of the InStream as a file.
        File.DownloadFromStream(inStr, 'Export', '', '', fileName);
    end;
}
```

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocument.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocument.WriteTo(var Text: Text)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocument.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocument.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlDocument Type: XmlDocument An instance of the XmlDocument data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocument Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an XML document type.

The following methods are available on the XmlDocumentType data type.

METHOD NAME	DESCRIPTION
Create(String)	Creates an XmlDocumentType node.
Create(String, String)	Creates an XmlDocumentType node.
Create(String, String, String)	Creates an XmlDocumentType node.
Create(String, String, String, String)	Creates an XmlDocumentType node.

The following methods are available on instances of the XmlDocumentType data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetInternalSubset(var Text)	Gets the internal subset for this Document Type Definition (DTD).
GetName(var Text)	Gets the name for this Document Type Definition (DTD).
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
GetPublicId(var Text)	Gets the public identifier for this Document Type Definition (DTD).
GetSystemId(var Text)	Gets the system identifier for this Document Type Definition (DTD).
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.

METHOD NAME	DESCRIPTION
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SetInternalSubset(String)	Sets the internal subset for this Document Type Definition (DTD).
SetName(String)	Sets the name for this Document Type Definition (DTD).
SetPublicId(String)	Sets the public identifier for this Document Type Definition (DTD).
SetSystemId(String)	Sets the system identifier for this Document Type Definition (DTD).
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlDocumentType.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDocumentType node.

Syntax

```
XmlDocumentType :=   XmlDocumentType.Create(Name: String)
```

Parameters

Name

Type: String

A string that contains the qualified name of the DTD, which is the same as the qualified name of the root element of the XML document.

Return Value

XmlDocumentType Type: XmlDocumentType The created XmlDocumentType node.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDocumentType node.

Syntax

```
XmlDocumentType :=   XmlDocumentType.Create(Name: String, PublicId: String)
```

Parameters

Name

Type: String

A string that contains the qualified name of the DTD, which is the same as the qualified name of the root element of the XML document.

PublicId

Type: String

A string that contains the public identifier of an external public DTD.

Return Value

XmlDocumentType Type: XmlDocumentType The created XmlDocumentType node.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDocumentType node.

Syntax

```
XmlDocumentType :=   XmlDocumentType.Create(Name: String, PublicId: String, SystemId: String)
```

Parameters

Name

Type: String

A string that contains the qualified name of the DTD, which is the same as the qualified name of the root element of the XML document.

PublicId

Type: String

A string that contains the public identifier of an external public DTD.

SystemId

Type: String

A string that contains the system identifier of an external private DTD.

Return Value

XmlDocumentType Type: XmlDocumentType The created XmlDocumentType node.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlDocumentType node.

Syntax

```
XmlDocumentType :=   XmlDocumentType.Create(Name: String, PublicId: String, SystemId: String,
InternalSubSet: String)
```

Parameters

Name

Type: String

A string that contains the qualified name of the DTD, which is the same as the qualified name of the root element of the XML document.

PublicId

Type: String

A string that contains the public identifier of an external public DTD.

SystemId

Type: String

A string that contains the system identifier of an external private DTD.

InternalSubSet

Type: String

A string that contains the internal subset for an internal DTD.

Return Value

XmlDocumentType Type: XmlDocumentType The created XmlDocumentType node.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlDocumentType.AddAfterSelf(Content: Any,...)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlDocumentType.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlDocumentType.AsXmlNode()
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlDocumentType.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlDocumentType.GetDocument(var Document: XmlDocument)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.GetInternalSubset Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the internal subset for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.GetInternalSubset(var Result: Text)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Result
Type: Text
A string that contains the internal subset for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.GetName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the name for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.GetName(var Result: Text)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Result
Type: Text
A string that contains the name for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlDocumentType.GetParent(var Parent: XmlElement)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.GetPublicId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the public identifier for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.GetPublicId(var Result: Text)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Result
Type: Text
A string that contains the public identifier for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.GetSystemId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the system identifier for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.GetSystemId(var Result: Text)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Result
Type: Text
A string that contains the system identifier for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlDocumentType.Remove()
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlDocumentType.ReplaceWith(Node: Any,...)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlDocumentType.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlDocumentType.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlDocumentType.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

XPath
Type: String
The XPath expression.
Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlDocumentType.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node: XmlNode)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SetInternalSubset Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the internal subset for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.SetInternalSubset(Value: String)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Value
Type: String
A string that contains the new internal subset for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SetName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the name for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.SetName(Value: String)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Value
Type: String
A string that contains the new name for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SetPublicId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the public identifier for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.SetPublicId(Value: String)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Value
Type: String
A string that contains the new public identifier for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.SetSystemId Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the system identifier for this Document Type Definition (DTD).

Syntax

```
[Ok := ]  XmlDocumentType.SetSystemId(Value: String)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Value
Type: String
A string that contains the new system identifier for this Document Type Definition (DTD).

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocumentType.WriteTo(OutStream: OutStream)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocumentType.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocumentType.WriteTo(var Text: Text)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlDocumentType.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlDocumentType.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlDocumentType Type: XmlDocumentType An instance of the XmlDocumentType data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlDocumentType Data Type
Getting Started with AL
Developing Extensions

XmlElement Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents an XML element.

The following methods are available on the XmlElement data type.

METHOD NAME	DESCRIPTION
Create(String)	Creates an XmlElement node.
Create(String, String)	Creates an XmlElement node.
Create(String, String, Any,...)	Creates an XmlElement node.
Create(String, Any,...)	Creates an XmlElement node.

The following methods are available on instances of the XmlElement data type.

METHOD NAME	DESCRIPTION
Add(Any,...)	Adds the specified content as a child of this element.
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AddFirst(Any,...)	Adds the specified content at the start of the child list of this element.
AsXmlNode()	Converts the node to an XmlNode.
Attributes()	Gets a collection of the attributes of this element.
GetChildElements()	Gets a list containing the child elements for this element, in document order.
GetChildElements(String)	Gets a list containing the child elements for this element, in document order.
GetChildElements(String, String)	Gets a list containing the child elements for this element, in document order.
GetChildNodes()	Gets a list containing the child elements for this element, in document order.
GetDescendantElements()	Gets a list containing the descendant elements for this element, in document order.
GetDescendantElements(String)	Gets a list containing the descendant elements for this element, in document order.

METHOD NAME	DESCRIPTION
GetDescendantElements(String, String)	Gets a list containing the descendant elements for this element, in document order.
GetDescendantNodes()	Gets a list containing the descendant nodes for this element, in document order.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetNamespaceOfPrefix(String, var Text)	Gets the namespace associated with a particular prefix for this element.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
GetPrefixOfNamespace(String, var Text)	Gets the prefix associated with a namespace URI for this element.
HasAttributes()	Gets a boolean value indicating whether this element has at least one attribute.
HasElements()	Gets a value indicating whether this element has at least one child element.
InnerText()	Gets the concatenated values of the node and all its child nodes.
InnerXml()	Gets the markup representing only the child nodes of this node.
IsEmpty()	Gets a value indicating whether this element contains no content.
LocalName()	Gets the local name of this element.
Name()	Gets the fully qualified name of this element.
NamespaceUri()	Gets the namespace URI of this element.
Remove()	Removes this node from its parent element.
RemoveAllAttributes()	Removes the attributes of this element.
RemoveAttribute(String)	Removes the specified attribute from this element.
RemoveAttribute(String, String)	Removes the specified attribute from this element.
RemoveAttribute(XmlAttribute)	Removes the specified attribute from this element.
RemoveNodes()	Removes the child nodes from this element.
ReplaceNodes(Any,...)	Replaces the children nodes of this element with the specified content.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.

METHOD NAME	DESCRIPTION
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SetAttribute(String, String)	Sets the value of the specified attribute or create it if is not part of the element's attribute collection.
SetAttribute(String, String, String)	Sets the value of the specified attribute or create it if is not part of the element's attribute collection.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlElement.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlElement node.

Syntax

```
XmlElement :=   XmlElement.Create(Name: String)
```

Parameters

Name

Type: String

The fully qualified name of the element to create.

Return Value

XmlElement Type: XmlElement The created XmlElement node.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlElement node.

Syntax

```
XmlElement :=   XmlElement.Create(LocalName: String, NamespaceUri: String)
```

Parameters

LocalName

Type: String

The local name of the element to create.

NamespaceUri

Type: String

The namespace URI of the element to create.

Return Value

XmlElement Type: XmlElement The created XmlElement node.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlElement node.

Syntax

```
XmlElement :=   XmlElement.Create(LocalName: String, NamespaceUri: String, Content: Any,...)
```

Parameters

LocalName

Type: String

The local name of the element to create.

NamespaceUri

Type: String

The namespace URI of the element to create.

Content

Type: Any

The content to add to the element to create.

Return Value

XmlElement Type: XmlElement The created XmlElement node.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlElement node.

Syntax

```
XmlElement :=   XmlElement.Create(Name: String, Content: Any,...)
```

Parameters

Name

Type: String

The fully qualified name of the element to create.

Content

Type: Any

The content to add to the element to create.

Return Value

XmlElement Type: XmlElement The created XmlElement node.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content as a child of this element.

Syntax

```
[Ok := ]  XmlElement.Add(Content: Any,...)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Content
Type: Any
The content to be added as a child of this element.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlElement.AddAfterSelf(Content: Any,...)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlElement.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.AddFirst Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content at the start of the child list of this element.

Syntax

```
[Ok := ]  XmlElement.AddFirst(Content: Any,...)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Content
Type: Any
The content to be added as a child of this element.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlElement.AsXmlNode()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlElement.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Attributes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a collection of the attributes of this element.

Syntax

```
Attributes :=   XmlElement.Attributes()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Attributes Type: XmlAttributeCollection The attributes of this element.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetChildElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this element, in document order.

Syntax

```
ChildElements :=   XmlElement.GetChildElements()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

ChildElements Type: XmlNodeList A list containing the child elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetChildElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this element, in document order.

Syntax

```
ChildElements :=   XmlElement.GetChildElements(Name: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Name
Type: String
The fully qualified name of the elements to retrieve.

Return Value

ChildElements Type: XmlNodeList A list containing the child elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetChildElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this element, in document order.

Syntax

```
ChildElements :=   XmlElement.GetChildElements(LocalName: String, NamespaceUri: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

LocalName
Type: String
The local name of the elements to retrieve.

NamespaceUri
Type: String
The namespace URI of the elements to retrieve.

Return Value

ChildElements Type: XmlNodeList A list containing the child elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetChildNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the child elements for this element, in document order.

Syntax

```
ChildNodes :=   XmlElement.GetChildNodes()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

ChildNodes Type: XmlNodeList A list containing the child elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetDescendantElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant elements for this element, in document order.

Syntax

```
DescendantElements :=   XmlElement.GetDescendantElements()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

DescendantElements Type: XmlNodeList A list containing the descendant elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetDescendantElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant elements for this element, in document order.

Syntax

```
DescendantElements :=   XmlElement.GetDescendantElements(Name: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Name
Type: String
The fully qualified name of the elements to retrieve.

Return Value

DescendantElements Type: XmlNodeList A list containing the descendant elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetDescendantElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant elements for this element, in document order.

Syntax

```
DescendantElements :=   XmlElement.GetDescendantElements(LocalName: String, NamespaceUri: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

LocalName
Type: String
The local name of the elements to retrieve.

NamespaceUri
Type: String
The namespace URI of the elements to retrieve.

Return Value

DescendantElements Type: XmlNodeList A list containing the descendant elements for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetDescendantNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a list containing the descendant nodes for this element, in document order.

Syntax

```
DescendantNodes :=   XmlElement.GetDescendantNodes()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

DescendantNodes Type: XmlNodeList A list containing the descendant nodes for this element, in document order.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlElement.GetDocument(var Document: XmlDocument)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetNamespaceOfPrefix Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the namespace associated with a particular prefix for this element.

Syntax

```
[Ok := ]  XmlElement.GetNamespaceOfPrefix(Prefix: String, var Result: Text)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Prefix
Type: String
A string that contains the namespace prefix to look up.

Result
Type: Text
The namespace URI associated with the given prefix for this element.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlElement.GetParent(var Parent: XmlElement)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.GetPrefixOfNamespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the prefix associated with a namespace URI for this element.

Syntax

```
[Ok := ]  XmlElement.GetPrefixOfNamespace(Namespace: String, var Result: Text)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Namespace
Type: String
A namespace URI to look up.

Result
Type: Text
A string that contains the namespace prefix.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.HasAttributes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a boolean value indicating whether this element has at least one attribute.

Syntax

```
Value :=   XmlElement.HasAttributes()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: Boolean **true** if the current node has at least one attribute, otherwise **false**.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.HasElements Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this element has at least one child element.

Syntax

```
Value :=   XmlElement.HasElements()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: Boolean **true** if this element has at least one child element, otherwise **false**.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.InnerText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the concatenated values of the node and all its child nodes.

Syntax

```
Value :=   XmlElement.InnerText()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: String The concatenated values of the node and all its child nodes.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.InnerXml Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the markup representing only the child nodes of this node.

Syntax

```
Value :=   XmlElement.InnerXml()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: String The markup representing only the child nodes of this node.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.IsEmpty Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this element contains no content.

Syntax

```
Value :=   XmlElement.IsEmpty()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: Boolean **true** if this element does not have content, otherwise **false**.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.LocalName Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the local name of this element.

Syntax

```
Value :=   XmlElement.LocalName()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: String The local name of this element.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Name Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the fully qualified name of this element.

Syntax

```
Value :=   XmlElement.Name()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: String The fully qualified name of this element.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.NamespaceUri Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the namespace URI of this element.

Syntax

```
Value :=   XmlElement.NamespaceUri()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Value Type: String The namespace URI of this element.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlElement.Remove()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.RemoveAllAttributes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the attributes of this element.

Syntax

```
XmlElement.RemoveAllAttributes()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.RemoveAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified attribute from this element.

Syntax

```
XmlElement.RemoveAttribute(Name: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Name

Type: String

The fully qualified name of the attribute to remove.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.RemoveAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified attribute from this element.

Syntax

```
XmlElement.RemoveAttribute(LocalName: String, NamespaceUri: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

LocalName
Type: String
The local name of the attribute to remove.

NamespaceUri
Type: String
The namespace URI of the attribute to remove.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.RemoveAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the specified attribute from this element.

Syntax

```
XmlElement.RemoveAttribute(Attribute: XmlAttribute)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Attribute
Type: XmlAttribute
The XmlAttribute to remove.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.RemoveNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the child nodes from this element.

Syntax

```
XmlElement.RemoveNodes()
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.ReplaceNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces the children nodes of this element with the specified content.

Syntax

```
[Ok := ]  XmlElement.ReplaceNodes(Content: Any,...)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Content
Type: Any
The content that replaces the children nodes.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlElement.ReplaceWith(Node: Any,...)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlElement.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlElement.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlElement.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlElement.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node:
XmlNode)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.SetAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the value of the specified attribute or create it if is not part of the element's attribute collection.

Syntax

```
XmlElement.SetAttribute(Name: String, Value: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Name

Type: String

The fully qualified name of the attribute to set.

Value

Type: String

The value to set for the attribute.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.SetAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the value of the specified attribute or create it if is not part of the element's attribute collection.

Syntax

```
XmlElement.SetAttribute(LocalName: String, NamespaceUri: String, Value: String)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

LocalName
Type: String
The local name of the attribute to set.

NamespaceUri
Type: String
The namespace URI of the attribute to set.

Value
Type: String
The value to set for the attribute.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlElement.WriteTo(OutStream: OutStream)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlElement.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlElement.WriteTo(var Text: Text)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlElement.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlElement.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlElement Type: XmlElement An instance of the XmlElement data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlElement Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a namespace manager that can be used to resolve, add and remove namespaces to a collection. It also provides scope management for these namespaces.

The following methods are available on instances of the XmlNamespaceManager data type.

METHOD NAME	DESCRIPTION
AddNamespace(String, String)	Adds the given namespace to the collection.
HasNamespace(String)	Gets a value indicating whether the supplied prefix has a namespace defined for the current scope.
LookupNamespace(String, var Text)	Gets the namespace URI for the specified prefix.
LookupPrefix(String, var Text)	Finds the prefix declared for the given namespace URI.
NameTable([XmlNameTable])	Gets or sets the XmlNameTable associated with this object.
PopScope()	Pops a namespace scope off the stack.
PushScope()	Pushes a namespace scope onto the stack.
RemoveNamespace(String, String)	Removes the given namespace for the given prefix.

See Also

Getting Started with AL
Developing Extensions

XmlNamespaceManager.AddNamespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the given namespace to the collection.

Syntax

```
XmlNamespaceManager.AddNamespace(Prefix: String, Uri: String)
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

Prefix

Type: String

The prefix to associate with the namespace being added. Use an empty string to add a default namespace.

Uri

Type: String

The namespace to add.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager.HasNamespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether the supplied prefix has a namespace defined for the current scope.

Syntax

```
HasNamespace :=   XmlNamespaceManager.HasNamespace(Prefix: String)
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

Prefix
Type: String
The prefix of the namespace you want to find.

Return Value

HasNamespace Type: Boolean **true** if the supplied prefix has a namespace defined for the current scope, otherwise **false**.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager.LookupNamespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the namespace URI for the specified prefix.

Syntax

```
[Ok := ]  XmlNamespaceManager.LookupNamespace(Prefix: String, var Result: Text)
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

Prefix
Type: String
The prefix whose namespace URI you want to resolve. To match the default namespace, pass an empty string.

Result
Type: Text
The namespace URI for prefix.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager.LookupPrefix Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Finds the prefix declared for the given namespace URI.

Syntax

```
[Ok := ]  XmlNamespaceManager.LookupPrefix(Uri: String, var Result: Text)
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

Uri

Type: String

The namespace to resolve for the prefix.

Result

Type: Text

The matching prefix. If there is no mapped prefix, the method returns an empty string.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager.NameTable Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the XmlNameTable associated with this object.

Syntax

```
[Value := ]  XmlNamespaceManager.NameTable([NewValue: XmlNameTable])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

NewValue

Type: XmlNameTable

The new XmlNameTable to associate with this object. Setting the NameTable will reset the state of the XmlNamespaceManager.

Return Value

Value Type: XmlNameTable The XmlNameTable associated with this object.

See Also

XmlNamespaceManager Data Type

Getting Started with AL

Developing Extensions

XmlNamespaceManager.PopScope Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Pops a namespace scope off the stack.

Syntax

```
XmlNamespaceManager.PopScope()
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager.PushScope Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Pushes a namespace scope onto the stack.

Syntax

```
XmlNamespaceManager.PushScope()
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNamespaceManager.RemoveNamespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes the given namespace for the given prefix.

Syntax

```
XmlNamespaceManager.RemoveNamespace(Prefix: String, Uri: String)
```

Parameters

XmlNamespaceManager Type: XmlNamespaceManager An instance of the XmlNamespaceManager data type.

Prefix

Type: String

The prefix for the namespace.

Uri

Type: String

The namespace to remove for the given prefix. The namespace removed is from the current namespace scope. Namespaces outside the current scope are ignored.

See Also

XmlNamespaceManager Data Type
Getting Started with AL
Developing Extensions

XmlNameTable Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a table of atomized string objects.

The following methods are available on instances of the XmlNameTable data type.

METHOD NAME	DESCRIPTION
Add(String)	Atomizes the specified string and adds it to the XmlNameTable.
Get(String, var Text)	Gets the atomized string with the specified value.

See Also

Getting Started with AL
Developing Extensions

XmlNameTable.Add Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Atomizes the specified string and adds it to the XmlNameTable.

Syntax

```
[AddedKey := ]  XmlNameTable.Add(Key: String)
```

Parameters

XmlNameTable Type: XmlNameTable An instance of the XmlNameTable data type.

Key
Type: String
The string to add.

Return Value

AddedKey Type: String The new atomized string or the existing one if it already exists

See Also

XmlNameTable Data Type
Getting Started with AL
Developing Extensions

XmlNameTable.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the atomized string with the specified value.

Syntax

```
[Ok := ]  XmlNameTable.Get(Key: String, var Result: Text)
```

Parameters

XmlNameTable Type: XmlNameTable An instance of the XmlNameTable data type.

Key
Type: String
The string to find.

Result
Type: Text
The atomized string object if the string has been atomized.

Return Value

Ok Type: Boolean **true** if the key exists, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNameTable Data Type
Getting Started with AL
Developing Extensions

XmlNode Data Type

2/17/2021 • 2 minutes to read • Edit Online

> **Version**: *Available from runtime version 1.0.*

Represents a XML node which can either be for instance an XML attribute, an XML element or a XML document.

The following methods are available on instances of the XmlNode data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlAttribute()	Converts the node to an XmlAttribute node. The operation will fail if the node is not an XmlAttribute.
AsXmlCData()	Converts the node to an XmlCData node. The operation will fail if the node is not an XmlCData.
AsXmlComment()	Converts the node to an XmlComment node. The operation will fail if the node is not an XmlComment.
AsXmlDeclaration()	Converts the node to an XmlDeclaration node. The operation will fail if the node is not an XmlDeclaration.
AsXmlDocument()	Converts the node to an XmlDocument node. The operation will fail if the node is not an XmlDocument.
AsXmlDocumentType()	Converts the node to an XmlDocumentType node. The operation will fail if the node is not an XmlDocumentType.
AsXmlElement()	Converts the node to an XmlElement node. The operation will fail if the node is not an XmlElement.
AsXmlProcessingInstruction()	Converts the node to an XmlProcessingInstruction node. The operation will fail if the node is not an XmlProcessingInstruction.
AsXmlText()	Converts the node to an XmlText node. The operation will fail if the node is not an XmlText.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
IsXmlAttribute()	Gets a value indicating whether this node is an XmlAttribute.
IsXmlCData()	Gets a value indicating whether this node is an XmlCData.
IsXmlComment()	Gets a value indicating whether this node is an XmlComment.

METHOD NAME	DESCRIPTION
IsXmlDeclaration()	Gets a value indicating whether this node is an XmlDeclaration.
IsXmlDocument()	Gets a value indicating whether this node is an XmlDocument.
IsXmlDocumentType()	Gets a value indicating whether this node is an XmlDocumentType.
IsXmlElement()	Gets a value indicating whether this node is an XmlElement.
IsXmlProcessingInstruction()	Gets a value indicating whether this node is an XmlProcessingInstruction.
IsXmlText()	Gets a value indicating whether this node is an XmlText.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlNode.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlNode.AddAfterSelf(Content: Any,...)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlNode.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlAttribute node. The operation will fail if the node is not an XmlAttribute.

Syntax

```
XmlAttribute :=   XmlNode.AsXmlAttribute()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlAttribute Type: XmlAttribute An XmlAttribute value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlCData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlCData node. The operation will fail if the node is not an XmlCData.

Syntax

```
XmlCData :=   XmlNode.AsXmlCData()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlCData Type: XmlCData An XmlCData value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlComment Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlComment node. The operation will fail if the node is not an XmlComment.

Syntax

```
XmlComment :=   XmlNode.AsXmlComment()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlComment Type: XmlComment An XmlComment value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlDeclaration node. The operation will fail if the node is not an XmlDeclaration.

Syntax

```
XmlDeclaration :=   XmlNode.AsXmlDeclaration()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlDeclaration Type: XmlDeclaration An XmlDeclaration value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlDocument node. The operation will fail if the node is not an XmlDocument.

Syntax

```
XmlDocument :=   XmlNode.AsXmlDocument()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlDocument Type: XmlDocument An XmlDocument value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlDocumentType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlDocumentType node. The operation will fail if the node is not an XmlDocumentType.

Syntax

```
XmlDocumentType :=   XmlNode.AsXmlDocumentType()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlDocumentType Type: XmlDocumentType An XmlDocumentType value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlElement Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlElement node. The operation will fail if the node is not an XmlElement.

Syntax

```
XmlElement :=   XmlNode.AsXmlElement()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlElement Type: XmlElement An XmlElement value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlProcessingInstruction Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlProcessingInstruction node. The operation will fail if the node is not an XmlProcessingInstruction.

Syntax

```
XmlProcessingInstruction :=   XmlNode.AsXmlProcessingInstruction()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlProcessingInstruction Type: XmlProcessingInstruction An XmlProcessingInstruction value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.AsXmlText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlText node. The operation will fail if the node is not an XmlText.

Syntax

```
XmlText :=   XmlNode.AsXmlText()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

XmlText Type: XmlText An XmlText value that references the current XmlNode.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlNode.GetDocument(var Document: XmlDocument)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlNode.GetParent(var Parent: XmlElement)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlAttribute Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlAttribute.

Syntax

```
Value :=   XmlNode.IsXmlAttribute()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlAttribute node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlCData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlCData.

Syntax

```
Value :=   XmlNode.IsXmlCData()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlCData node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlComment Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlComment.

Syntax

```
Value :=   XmlNode.IsXmlComment()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlComment node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlDeclaration Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlDeclaration.

Syntax

```
Value :=   XmlNode.IsXmlDeclaration()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlDeclaration node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlDocument.

Syntax

```
Value :=   XmlNode.IsXmlDocument()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlDocument node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlDocumentType Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlDocumentType.

Syntax

```
Value :=   XmlNode.IsXmlDocumentType()
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlDocumentType node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlElement Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlElement.

Syntax

```
Value :=   XmlNode.IsXmlElement()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlElement node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlProcessingInstruction Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlProcessingInstruction.

Syntax

```
Value :=   XmlNode.IsXmlProcessingInstruction()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlProcessingInstruction node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.IsXmlText Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a value indicating whether this node is an XmlText.

Syntax

```
Value :=   XmlNode.IsXmlText()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Value Type: Boolean **true** if this node is an XmlText node, otherwise **false**.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlNode.Remove()
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlNode.ReplaceWith(Node: Any,...)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlNode.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

XPath
Type: String
The XPath expression.
NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlNode.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlNode.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlNode.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node: XmlNode)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlNode.WriteTo(OutStream: OutStream)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlNode.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlNode.WriteTo(var Text: Text)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNode.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlNode.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlNode Type: XmlNode An instance of the XmlNode data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNode Data Type
Getting Started with AL
Developing Extensions

XmlNodeList Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a collection of XML nodes.

The following methods are available on instances of the XmlNodeList data type.

METHOD NAME	DESCRIPTION
Count()	Gets the number of nodes in the XmlNodeList.
Get(Integer, var XmlNode)	Gets a node at the given index.

See Also

Getting Started with AL
Developing Extensions

XmlNodeList.Count Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the number of nodes in the XmlNodeList.

Syntax

```
Count :=   XmlNodeList.Count()
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlNodeList Type: XmlNodeList An instance of the XmlNodeList data type.

Return Value

Count Type: Integer The number of nodes in the XmlNodeList.

See Also

XmlNodeList Data Type
Getting Started with AL
Developing Extensions

XmlNodeList.Get Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets a node at the given index.

Syntax

```
[Ok := ]  XmlNodeList.Get(Index: Integer, var Node: XmlNode)
```

Parameters

XmlNodeList Type: XmlNodeList An instance of the XmlNodeList data type.

Index
Type: Integer
The one-based index into the list of nodes.

Node
Type: XmlNode
The XmlNode with the specified index in the list.

Return Value

Ok Type: Boolean **true** if a node is found at the given index, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlNodeList Data Type
Getting Started with AL
Developing Extensions

Xmlport Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

XmlPorts are used to export or import data between an external source and a Microsoft Dynamics Business Central database.

The following methods are available on the Xmlport data type.

METHOD NAME	DESCRIPTION
Export(Integer, var OutStream [, var Record])	Creates an XML data stream (XML document) and sends it to a chosen destination.
Import(Integer, var InStream [, var Record])	Reads and parses an incoming XML data stream (XML document).
Run(Integer [, Boolean] [, Boolean] [, var Record])	Loads and executes the XmlPort that you specify.

The following methods are available on instances of the Xmlport data type.

METHOD NAME	DESCRIPTION
Break()	Exits from a loop or a trigger in a data item trigger of a report or XmlPort.
BreakUnbound()	Exits from a loop on records in an XmlPort trigger.
CurrentPath()	Returns the CurrentPath for a given node, used when exporting an XmlPort.
Export()	Creates an XML data stream (XML document) and sends it to a chosen destination.
FieldDelimiter([String])	Gets and sets the FiledDelimiter used when running, importing or exporting the XmlPort.
FieldSeparator([String])	Gets and sets the FieldSeparator used when running, importing or exporting the XmlPort.
Filename([String])	Gets the current value of the FileName Property of an XmlPort and sets this property to a new value.
Import()	Reads and parses an incoming XML data stream (XML document).
ImportFile([Boolean])	Gets or sets the ImportFile property.
Quit()	Aborts the processing of a report or XmlPort.

METHOD NAME	DESCRIPTION
RecordSeparator([String])	Gets and sets the RecordSeparator used when running, importing or exporting the XmlPort.
Run()	Loads and executes the XmlPort that you specify.
SetDestination(var OutStream)	Sets the destination OutStream of the XmlPort.
SetSource(var InStream)	Sets the source InStream of the XmlPort.
SetTableView(var Record)	Applies the table view on the current record as the table view for the page, report, or XmlPort.
Skip()	Skips the current iteration of the current report or XmlPort.
TableSeparator([String])	Gets and sets the TableSeparator used when running, importing or exporting the XmlPort.
TextEncoding([TextEncoding])	Gets and sets the TextEncoding used when running, importing or exporting the XmlPort.

See Also

Getting Started with AL
Developing Extensions
XMLport Overview

Xmlport.Export Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XML data stream (XML document) and sends it to a chosen destination.

Syntax

```
[Ok := ]  Xmlport.Export(Number: Integer, var OutStream: OutStream [, var Record: Record])
```

Parameters

Number

Type: Integer

The ID of the XmlPort that you want to run.

OutStream

Type: OutStream

Where the XmlPort object will write the XML data stream.

Record

Type: Record

The record to use in the XmlPort. Any filters attached to the record will be used. This parameter is optional. If this parameter is omitted, all records in the table are exported.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Import Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads and parses an incoming XML data stream (XML document).

Syntax

```
[Ok := ]  Xmlport.Import(Number: Integer, var InStream: InStream [, var Record: Record])
```

Parameters

Number

Type: Integer

The ID of the XmlPort that you want to run. If the XmlPort that you specify does not exist, a run-time error occurs.

InStream

Type: InStream

The source from which the import XmlPort object will read the XML data stream.

Record

Type: Record

The record to use in the XmlPort. Any filters that are attached to the record will be used. This parameter is optional. If this parameter is omitted, all records in the table are imported. For example, you can use this parameter to change the values of properties such as FieldSeparator or TextEncoding, depending on each record that is imported.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the XmlPort that you specify.

Syntax

```
Xmlport.Run(Number: Integer [, RequestWindow: Boolean] [, Import: Boolean] [, var Record: Record])
```

Parameters

Number

Type: Integer

Specifies the XmlPort to run.

RequestWindow

Type: Boolean

Specify true to show request page; specify false to run the report and skip the request page. This parameter overrides the setting of the UseRequestPage Property of the XMLPort. If you do not provide a value for the RequestWindow parameter, then the setting of the UseRequestPage property is used. > Web client does not support request pages with XmlPorts. If the XMLPort will appear in the web client, you should set the value to false; otherwise, you will get an error at runtime.

Import

Type: Boolean

Specifies whether the XMLPort imports or exports data. Specify true to run the XmlPort and import data; specify false to export data.This parameter is most relevant when the XmlPort does not use a request page and the Direction Property of the XmlPort is set to Both. In this instance, you use the parameter to specify the direction of the data. If the XmlPort uses a request page, then a direction option appears on the request page that enables the user can choose to import or export data. The Import parameter specifies the default value in the direction on option on the request page. If the Direction property is set to Import or Export, then you must set this parameter to match the direction that is set by the Direction property; otherwise, you will get an error at runtime. The default is true.

Record

Type: Record

The record to use in the XmlPort. The system will use any filters that are attached to the specified record.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Break Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exits from a loop or a trigger in a data item trigger of a report or XmlPort.

Syntax

```
Xmlport.Break()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.BreakUnbound Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Exits from a loop on records in an XmlPort trigger.

Syntax

```
Xmlport.BreakUnbound()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.CurrentPath Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Returns the CurrentPath for a given node, used when exporting an XmlPort.

Syntax

```
Path :=   Xmlport.CurrentPath()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Return Value

Path Type: String The current path for a given node.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Export Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XML data stream (XML document) and sends it to a chosen destination.

Syntax

```
[Ok := ]  Xmlport.Export()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Return Value

Ok Type: Boolean **true** if the serialization of the XML document was successful, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.FieldDelimiter Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the FiledDelimiter used when running, importing or exporting the XmlPort.

Syntax

```
[Delimiter := ]  Xmlport.FieldDelimiter([Delimiter: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Delimiter

Type: String

The new value of the FieldDelimiter.

Return Value

Delimiter Type: String The FieldDelimiter used when running, importing or exporting the XmlPort.

See Also

Xmlport Data Type
FieldDelimiter Property
Getting Started with AL
Developing Extensions

Xmlport.FieldSeparator Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the FieldSeparator used when running, importing or exporting the XmlPort.

Syntax

```
[Separator := ]  Xmlport.FieldSeparator([Separator: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Separator
Type: String
The new value of the FieldSeparator.

Return Value

Separator Type: String The FieldSeparator used when running, importing or exporting the XmlPort.

See Also

FieldSeparator Property
Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Filename Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the current value of the FileName Property of an XmlPort and sets this property to a new value.

Syntax

```
[FileName := ]  Xmlport.Filename([FileName: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

FileName

Type: String

The new file name.

Return Value

FileName Type: String The current value of the FileName Property of an XmlPort.

See Also

Xmlport Data Type
FileName Property
Getting Started with AL
Developing Extensions

Xmlport.Import Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Reads and parses an incoming XML data stream (XML document).

Syntax

```
[Ok := ]  Xmlport.Import()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Return Value

Ok Type: Boolean **true** if the import of the XML document was successful, otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.ImportFile Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the ImportFile property.

Syntax

```
[ImportFile := ]  Xmlport.ImportFile([ImportFile: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

ImportFile
Type: Boolean
The new value of the ImportFile property. If the XmlPort is designed for export only and the new value of this property is **true**, an XmlPortExportDirectionException is thrown. If the XmlPort is designed for import only and the new value of this property **false**, an XmlPortImportDirectionException is thrown.

Return Value

ImportFile Type: Boolean The current value of the ImportFile property.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Quit Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Aborts the processing of a report or XmlPort.

Syntax

```
Xmlport.Quit()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.RecordSeparator Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the RecordSeparator used when running, importing or exporting the XmlPort.

Syntax

```
[Separator := ]  Xmlport.RecordSeparator([Separator: String])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Separator
Type: String
The new value of the RecordSeparator.

Return Value

Separator Type: String The RecordSeparator used when running, importing or exporting the XmlPort.

See Also

Xmlport Data Type
RecordSeparator Property
Getting Started with AL
Developing Extensions

Xmlport.Run Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Loads and executes the XmlPort that you specify.

Syntax

```
Xmlport.Run()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.SetDestination Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the destination OutStream of the XmlPort.

Syntax

```
Xmlport.SetDestination(var OutStream: OutStream)
```

NOTE

This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

OutStream

Type: OutStream

The new destination OutStream of the XmlPort.

See Also

Xmlport Data Type

Getting Started with AL

Developing Extensions

Xmlport.SetSource Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the source InStream of the XmlPort.

Syntax

```
Xmlport.SetSource(var InStream: InStream)
```

NOTE

This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

InStream

Type: InStream

The new source InStream of the XmlPort.

See Also

Xmlport Data Type

Getting Started with AL

Developing Extensions

Xmlport.SetTableView Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Applies the table view on the current record as the table view for the page, report, or XmlPort.

Syntax

```
Xmlport.SetTableView(var Record: Record)
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Record

Type: Record

The record that has a table view that you want to apply to the page or data item.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.Skip Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Skips the current iteration of the current report or XmlPort.

Syntax

```
Xmlport.Skip()
```

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

See Also

Xmlport Data Type
Getting Started with AL
Developing Extensions

Xmlport.TableSeparator Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the TableSeparator used when running, importing or exporting the XmlPort.

Syntax

```
[Separator := ]  Xmlport.TableSeparator([Separator: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Separator

Type: String

The new value of the TableSeparator.

Return Value

Separator Type: String The TableSeparator used when running, importing or exporting the XmlPort.

See Also

Xmlport Data Type

TableSeparator Property (XMLports)

Getting Started with AL

Developing Extensions

Xmlport.TextEncoding Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets and sets the TextEncoding used when running, importing or exporting the XmlPort.

Syntax

```
[Encoding := ]  Xmlport.TextEncoding([Encoding: TextEncoding])
```

NOTE

This method can be invoked using property access syntax.

Parameters

Xmlport Type: Xmlport An instance of the Xmlport data type.

Encoding

Type: TextEncoding

The new value of the TextEncoding.

Return Value

Encoding Type: TextEncoding The TextEncoding used when running, importing or exporting the XmlPort.

See Also

Xmlport Data Type

TextEncoding Property (XMLports)

Getting Started with AL

Developing Extensions

XmlProcessingInstruction Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents a processing instruction, which XML defines to keep processor-specific information in the text of the document.

The following methods are available on the XmlProcessingInstruction data type.

METHOD NAME	DESCRIPTION
Create(String, String)	Creates an XmlProcessingInstruction node.

The following methods are available on instances of the XmlProcessingInstruction data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
GetData(var Text)	Gets the content of the processing instruction, excluding the target.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
GetTarget(var Text)	Gets the target of the processing instruction.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SetData(String)	Sets the content of the processing instruction, excluding the target.

METHOD NAME	DESCRIPTION
SetTarget(String)	Sets the target of the processing instruction.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlProcessingInstruction.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlProcessingInstruction node.

Syntax

```
XmlProcessingInstruction :=   XmlProcessingInstruction.Create(Target: String, Data: String)
```

Parameters

Target

Type: String

The target of the processing instruction.

Data

Type: String

The content of the processing instruction, excluding the target.

Return Value

XmlProcessingInstruction Type: XmlProcessingInstruction The created XmlProcessingInstruction node.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlProcessingInstruction.AddAfterSelf(Content: Any,...)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlProcessingInstruction.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlProcessingInstruction.AsXmlNode()
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlProcessingInstruction.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.GetData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the content of the processing instruction, excluding the target.

Syntax

```
[Ok := ]  XmlProcessingInstruction.GetData(var Result: Text)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Result
Type: Text
The content of the processing instruction, excluding the target.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlProcessingInstruction.GetDocument(var Document: XmlDocument)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlProcessingInstruction.GetParent(var Parent: XmlElement)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.GetTarget Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the target of the processing instruction.

Syntax

```
[Ok := ]  XmlProcessingInstruction.GetTarget(var Result: Text)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Result
Type: Text
The target of the processing instruction.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlProcessingInstruction.Remove()
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlProcessingInstruction.ReplaceWith(Node: Any,...)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlProcessingInstruction.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlProcessingInstruction.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var
NodeList: XmlNodeList)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlProcessingInstruction.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlProcessingInstruction.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager,
var Node: XmlNode)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.SetData Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the content of the processing instruction, excluding the target.

Syntax

```
[Ok := ]  XmlProcessingInstruction.SetData(Value: String)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Value
Type: String
The new content of the processing instruction, excluding the target.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.SetTarget Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Sets the target of the processing instruction.

Syntax

```
[Ok := ]  XmlProcessingInstruction.SetTarget(Value: String)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Value
Type: String
The new target of the processing instruction.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlProcessingInstruction.WriteTo(OutStream: OutStream)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlProcessingInstruction.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlProcessingInstruction.WriteTo(var Text: Text)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

Text

Type: Text

The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlProcessingInstruction.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlProcessingInstruction.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlProcessingInstruction Type: XmlProcessingInstruction An instance of the XmlProcessingInstruction data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlProcessingInstruction Data Type
Getting Started with AL
Developing Extensions

XmlReadOptions Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the options configuring how XML is loaded from a data source.

The following methods are available on instances of the XmlReadOptions data type.

METHOD NAME	DESCRIPTION
PreserveWhitespace([Boolean])	Gets or sets a value that indicates whether insignificant white space should be preserved during parsing.

See Also

Getting Started with AL
Developing Extensions

XmlReadOptions.PreserveWhitespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets a value that indicates whether insignificant white space should be preserved during parsing.

Syntax

```
[Value := ]  XmlReadOptions.PreserveWhitespace([NewValue: Boolean])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlReadOptions Type: XmlReadOptions An instance of the XmlReadOptions data type.

NewValue

Type: Boolean

The new value of the flag.

Return Value

Value Type: Boolean **true** if insignificant white spaces are preserved during parsing, otherwise **false**.

See Also

XmlReadOptions Data Type
Getting Started with AL
Developing Extensions

XmlText Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the text content of an element or attribute.

The following methods are available on the XmlText data type.

METHOD NAME	DESCRIPTION
Create(String)	Creates an XmlText node.

The following methods are available on instances of the XmlText data type.

METHOD NAME	DESCRIPTION
AddAfterSelf(Any,...)	Adds the specified content immediately after this node.
AddBeforeSelf(Any,...)	Adds the specified content immediately before this node.
AsXmlNode()	Converts the node to an XmlNode.
GetDocument(var XmlDocument)	Gets the XmlDocument for this node.
GetParent(var XmlElement)	Gets the parent XmlElement of this node.
Remove()	Removes this node from its parent element.
ReplaceWith(Any,...)	Replaces this node with the specified content.
SelectNodes(String, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectNodes(String, XmlNamespaceManager, var XmlNodeList)	Selects a list of nodes matching the XPath expression.
SelectSingleNode(String, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
SelectSingleNode(String, XmlNamespaceManager, var XmlNode)	Selects the first XmlNode that matches the XPath expression.
Value([String])	Gets or sets the value of this node.
WriteTo(OutStream)	Serializes and saves the current node to the given variable.
WriteTo(XmlWriteOptions, OutStream)	Serializes and saves the current node to the given variable.
WriteTo(var Text)	Serializes and saves the current node to the given variable.

METHOD NAME	DESCRIPTION
WriteTo(XmlWriteOptions, var Text)	Serializes and saves the current node to the given variable.

See Also

Getting Started with AL
Developing Extensions

XmlText.Create Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Creates an XmlText node.

Syntax

```
Value :=   XmlText.Create(Content: String)
```

Parameters

Content

Type: String

A string that contains the value of the new XmlText node.

Return Value

Value Type: XmlText The created XmlText node.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.AddAfterSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately after this node.

Syntax

```
[Ok := ]  XmlText.AddAfterSelf(Content: Any,...)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Content
Type: Any
The content to add after this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.AddBeforeSelf Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Adds the specified content immediately before this node.

Syntax

```
[Ok := ]  XmlText.AddBeforeSelf(Content: Any,...)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Content
Type: Any
The content to add before this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.AsXmlNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Converts the node to an XmlNode.

Syntax

```
XmlNode :=   XmlText.AsXmlNode()
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Return Value

XmlNode Type: XmlNode An XmlNode value that references the current XmlText.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.GetDocument Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the XmlDocument for this node.

Syntax

```
[Ok := ]  XmlText.GetDocument(var Document: XmlDocument)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Document
Type: XmlDocument
The owning XmlDocument of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.GetParent Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets the parent XmlElement of this node.

Syntax

```
[Ok := ]  XmlText.GetParent(var Parent: XmlElement)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Parent
Type: XmlElement
The parent XmlElement of this node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.Remove Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Removes this node from its parent element.

Syntax

```
[Ok := ]  XmlText.Remove()
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.ReplaceWith Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Replaces this node with the specified content.

Syntax

```
[Ok := ]  XmlText.ReplaceWith(Node: Any,...)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Node
Type: Any
The content with which to replace the current node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlText.SelectNodes(XPath: String, var NodeList: XmlNodeList)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

XPath
Type: String
The XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.SelectNodes Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects a list of nodes matching the XPath expression.

Syntax

```
[Ok := ]  XmlText.SelectNodes(XPath: String, NamespaceManager: XmlNamespaceManager, var NodeList:
XmlNodeList)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

NodeList
Type: XmlNodeList
An XmlNodeList containing a collection of nodes matching the XPath expression.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlText.SelectSingleNode(XPath: String, var Node: XmlNode)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

XPath
Type: String
The XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.SelectSingleNode Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Selects the first XmlNode that matches the XPath expression.

Syntax

```
[Ok := ]  XmlText.SelectSingleNode(XPath: String, NamespaceManager: XmlNamespaceManager, var Node: XmlNode)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

XPath
Type: String
The XPath expression.

NamespaceManager
Type: XmlNamespaceManager
An XmlNamespaceManager to use for resolving namespaces for prefixes in the XPath expression.

Node
Type: XmlNode
The first XmlNode that matches the XPath query.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.Value Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets the value of this node.

Syntax

```
[Value := ]  XmlText.Value([NewValue: String])
```

NOTE

This method can be invoked using property access syntax.

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

NewValue

Type: String

The new value of this node.

Return Value

Value Type: String The value of this node.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlText.WriteTo(OutStream: OutStream)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlText.WriteTo(WriteOptions: XmlWriteOptions, OutStream: OutStream)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

OutStream
Type: OutStream
The OutStream to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlText.WriteTo(var Text: Text)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlText.WriteTo Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Serializes and saves the current node to the given variable.

Syntax

```
[Ok := ]  XmlText.WriteTo(WriteOptions: XmlWriteOptions, var Text: Text)
```

Parameters

XmlText Type: XmlText An instance of the XmlText data type.

WriteOptions
Type: XmlWriteOptions
Specifies options for customizing how the node is serialized.

Text
Type: Text
The Text variable to which you want to save the serialized representation of the node.

Return Value

Ok Type: Boolean **true** if the operation was successful; otherwise **false**. If you omit this optional return value and the operation does not execute successfully, a runtime error will occur.

See Also

XmlText Data Type
Getting Started with AL
Developing Extensions

XmlWriteOptions Data Type

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Represents the options configuring how XML is saved.

The following methods are available on instances of the XmlWriteOptions data type.

METHOD NAME	DESCRIPTION
PreserveWhitespace([Boolean])	Gets or sets a value that indicates whether insignificant white space should be preserved during serialization.

See Also

Getting Started with AL
Developing Extensions

XmlWriteOptions.PreserveWhitespace Method

2/17/2021 • 2 minutes to read • Edit Online

Version: *Available from runtime version 1.0.*

Gets or sets a value that indicates whether insignificant white space should be preserved during serialization.

Syntax

```
[Value := ]  XmlWriteOptions.PreserveWhitespace([NewValue: Boolean])
```

> **NOTE**
> This method can be invoked using property access syntax.

Parameters

XmlWriteOptions Type: XmlWriteOptions An instance of the XmlWriteOptions data type.

NewValue
Type: Boolean
The new value of the flag.

Return Value

Value Type: Boolean **true** if insignificant white spaces should be preserved during serialization, otherwise **false**.

See Also

XmlWriteOptions Data Type
Getting Started with AL
Developing Extensions

Properties Overview

2/17/2021 • 2 minutes to read • Edit Online

This section describes the properties that are available to developers in Dynamics 365 Business Central. Properties can be set explicitly in AL code using syntax such as:

```
Promoted = true;
PromotedCategory = Process;
ApplicationArea = All;
```

In the sections below, properties are sorted according to the object(s) they apply to.

- Table and Table Extension Properties
- Page and Page Extension Properties
- Codeunit Properties
- Query Properties
- Report Properties
- XMLPort Properties
- Control Add-In Properties
- Enum Properties
- View Properties
- Profile Properties
- Integrating with Dynamics 365 for Sales

See Also

Methods
Triggers

Table, Table Fields, and Table Extension Properties

2/17/2021 • 4 minutes to read • Edit Online

The following topic lists properties that apply to the table object and table fields and, in some cases, to the table extension object as specified below.

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
Access Property		• Table • Table fields
AccessByPermission Property		• BLOB field • BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • TableFilter field • Text field • Time field
AutoFormatExpression Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
AutoFormatType Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
AutoIncrement Property		• BigInteger field
BlankNumbers Property		• BigInteger field • Boolean field • Date field • DateTime field • Decimal field • Duration field • Integer field • Option field • Time field
BlankZero Property		• BigInteger field • Boolean field • Decimal field • Duration field • Integer field • Option

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
CalcFormula Property		• BigInteger field • Boolean field • Code field • Date field • DateTime field • Decimal field • Duration field • GUID field • Integer field • Media field • MediaSet field • Option field • RecordID field • Text field • Time field
Caption Property	X	• Table object • BLOB Field • BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • TableFilter field • Text field • Time field
CaptionClass Property	X	• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
CaptionML Property	X	• Table object • BLOB Field • BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • TableFilter field • Text field • Time field
CharAllowed Property		• Code field • OemCode field • OemText field • Text field
ClosingDates Property	X	• Date field
Compressed Property		• BLOB field
Data Type		• Table fields
DataClassification		• Table object
DataCaptionFields Property	X	• Table object
DataPerCompany Property		• Table object
DateFormula Property		• Code • OemCode field • OemText field • Text field

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
Description Property	X	• Table object • BLOB field • BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • TableFilter field • Text field • Time field
DrillDownPageID Property		• Table object
Editable Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
Enabled Property		• BLOB field • BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • TableFilter field • Text field • Time field
ExtendedDataType Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
Extensible Property		• Table object
ExternalName Property		• Table object
ExternalSchema Property		• Table object

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
FieldClass Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
InitValue Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
LinkedInTransaction Property		• Table object
LinkedObject Property		• Table object
LookupPageID Property		• Table object
MaxValue Property		• BigInteger field • Boolean field • Date field • DateTime field • Decimal field • Duration field • Integer field • Option field • Time field

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
MinValue Property		• BigInteger field • Boolean field • Date field • DateTime field • Decimal field • Duration field • Integer field • Option field • Time field
NotBlank Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
Numeric Property		• Code field • OemCode field • OemText field • Text field
ObsoleteReason		• Table object • Table keys • Text field
ObsoleteState		• Table object • Table keys • Text field
OptionCaption Property	X	• Option field
OptionMembers Property		• Option field
PasteIsValid Property		• Table object

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
Permissions Property		• Table object
ReplicateData Property		• Table object
Scope (Table) Property		• Table object
SignDisplacement Property		• BigInteger field • Boolean field • Date field • DateTime field • Decimal field • Duration field • Integer field • Option field • Time field
SQLDataType Property		• Code field • OemCode field
SqlTimeStamp Property		• BigInteger
SubType Property (BLOB)		• BLOB field
TableRelation Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
TableType Property		• Table object

PROPERTY NAME	AVAILABLE FOR TABLE EXTENSION OBJECT	APPLIES TO
ValidateTableRelation Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
ValuesAllowed Property		• BigInteger field • Boolean field • Code field • Date field • DateFormula field • DateTime field • Decimal field • Duration field • GUID field • Integer field • OemCode field • OemText field • Option field • RecordID field • Text field • Time field
Width Property	X	• BigInteger field • Code field • Decimal field • Duration field • Integer field • OemCode field • OemText field • Text field

See Also

Properties
Page and Page Extension Properties

Page, Page Fields, and Page Extension Properties

2/17/2021 • 3 minutes to read • Edit Online

This topic lists properties that apply to the page object, page fields, and page extension object.

Page object properties

The following properties all apply to the page object, only some of these properties can be set for a page extension object as specified below. This list is sorted alphabetically by property name. For page properties sorted by method on a page, see Page Properties.

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
Access Property		• Codeunit • Query • Table • Table Field • Enum Type • Interface
AccessByPermission Property		• Page object • Field control • Part control • Action
AdditionalSearchTerms Property		• Page object
AdditionalSearchTermsML Property		• Page object
ApplicationArea Property	X (see next column)	• Page object (cannot be specified for a page extension object) • Field control • Part control • Action
AssistEdit Property		• Field control
AutoFormatExpression Property		• Field control
AutoFormatType Property		• Field control
AutoSplitKey Property		• Page object

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
BlankNumbers Property		• Field control
BlankZero Property		• Field control
CaptionML Property	X	• Page object • Container control • Group control • Field control • Part control • ActionGroup • Action • Separator
CaptionClass Property	X	• Field control
CharAllowed Property		• Field control
ClosingDates Property	X	• Field control
ColumnSpan Property		• Field control
ContextSensitiveHelpPage Property	X	• Page object
ContainerType Property	X	• Container control
DataCaptionExpression Property	X	• Page object
DateFormula Property		• Field control
DecimalPlaces Property		• Field control
DelayedInsert Property		• Page object
DeleteAllowed Property		• Page object

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
Description Property	X	• Page object • Container control • Group control • Field control • Part control • ActionGroup • Action
DrillDown Property		• Field control
DrillDownPageId Property		• Field control
Editable Property		• Page object • Group control • Field control • Part control
Ellipsis Property		• Action
Enabled Property	X	• Group control • Field control • Part control • ActionGroup • Action
EntityCaption Property		• Page object
EntityCaptionML Property		• Page object
EntityName Property		• Page object • Part control
EntitySetCaption Property		• Page object
EntitySetCaptionML Property		• Page object
EntitySetName Property		• Page object • Part control
ExtendedDataType Property		• Field control
Extensible Property		• Page object

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
FreezeColumn Property	X	• Group control
GridLayout Property		• Group control
Gesture Property		• Action
HideValue Property	X	• Field control
IndentationControls Property		• Group control
InFooterBar Property	X	• Action
InsertAllowed Property		• Page object
InstructionalTextML Property	X	• Page object • Group control
Image Property		• Field control • ActionGroup • Action
Importance Property	X	• Field control
LinksAllowed Property		• Page object
LookupPageId Property		• Field control
Lookup Property		• Field control
ModifyAllowed Property		• Page object
MultipleNewLines Property		• Page object
Multiplicity Property		• Page part
MinValue Property		• Field control
MaxValue Property		• Field control

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
MultiLine Property		• Field control
NavigationPageId Property		• Field control
NotBlank Property		• Field control
Numeric Property		• Field control
ObsoleteState Property		• Page object • Action • Action area • Action group • Action separator • Area • Chart part • Field control • Group • Label • Part • System part
ObsoleteReason Property		• Page object • Action • Action area • Action group • Action separator • Area • Chart part • Field control • Group • Label • Part • System part
ODataEDMType Property	X	• Page object • Field control
ODataKeyFields Property		• Page object
OptionCaptionML Property		• Field control
PageType Property		• Page object
Permissions Property		• Page object

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
PopulateAllFields Property		• Page object
Promoted Property	X	• Action
PromotedActionCategories Property	X	• Page object
PromotedActionCategoriesML Property	X	• Page object
PromotedCategory Property	X	• Action
PromotedIsBig Property	X	• Action
PromotedOnly Property	X	• Action
Provider Property		• Part control
QuickEntry Property	X	• Field control
RefreshOnActivate Property		• Page object
RowSpan property		• Field control
RunObject Property		• Action
RunPageLink Property		• Action
RunPageMode Property		• Action
RunPageOnRec Property		• Action
RunPageView Property		• Action
SaveValues Property		• Page object
Scope Property		• Action
ShortcutKey Property		• Action

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
ShowAsTree Property		• Group control
ShowCaption Property	X	• Group control • Field control
ShowFilter Property		• Page object • Part control
ShowMandatory Property		• Field control
SignDisplacement Property		• Field control
SourceTable Property		• Page object
SourceTableTemporary Property		• Page object
SourceTableView Property		• Page object
Style Property	X	• Field control
StyleExpr Property	X	• Field control
SubPageView Property		• Part control
SubPageLink Property		• Part control
TableRelation Property		• Field control
ToolTipML Property	X	• Field control • Part control • ActionGroup • Action
UpdatePropagation Property		• Part control
UsageCategory Property		• Page object
ValuesAllowed Property		• Field control

PROPERTY NAME	AVAILABLE FOR PAGE EXTENSION OBJECT	APPLIES TO
Visible Property	X	• Group control • Field control • Part control • ActionGroup • Action • View
Width Property	X	• Field control

See Also

Properties
Page Object
Page Extension Object
Report Object
Table and Table Extension Properties

Profile Properties

2/17/2021 • 2 minutes to read • Edit Online

The following topic lists properties that apply to Profiles specifically.

PROPERTY NAME	OBJECT
ProfileDescription	Profiles
Enabled	Profiles
Promoted	Profiles
RoleCenter	Profiles
Customizations	Profiles
Caption	Profiles

See also

Codeunit Properties
Page Properties
Query Properties
Report Properties
Table Properties
XMLPort Properties

Codeunit Properties

2/17/2021 • 2 minutes to read • Edit Online

This topic lists properties that apply to the Codeunit object.

PROPERTY NAME	APPLIES TO
Access Property	• Codeunit objects
ConstValue Property	• Global Text Constants • AL Locals Text Constants
ConstValueML Property	• Global Text Constants • AL Locals Text Constants
EventSubscriberInstance Property	• Codeunit Object
OptionString Property	• Global Variables • AL Locals Variables
Permissions Property	• Codeunit Object
RunOnClient Property	• AL Locals variables
SingleInstance Property	• Codeunit Object
SubType Property (Codeunit)	• Codeunit Object
SuppressDispose Property	• AL Locals variables
TableNo Property	• Codeunit Object
TestIsolation Property	• Codeunit Object

See Also

Developing Extensions
AL Development Environment
Table and Table Extension Properties
Page and Page Extension Properties Overview

Query Properties

2/17/2021 • 2 minutes to read • Edit Online

This topic lists properties that apply to the query object.

PROPERTY NAME	APPLIES TO
Access Property	• Query object
APIPublisher Property	• Query Object
APIVersion Property (Query)	• Query Object
APIGroup Property	• Query Object
Caption Property	• Query Object • Column control • Filter control
CaptionML Property	• Query Object • Column control • Filter control
ColumnFilter Property	• Column control • Filter control
DataAccessIntent Property	• Query Object
DataItemLink Property (Query)	• DataItem control
Description Property	• Query Object • DataItem control • Column control • Filter control
EntityCaption Property	• Query Object
EntityCaptionML Property	• Query Object
EntityName Property	• Query Object

PROPERTY NAME	APPLIES TO
EntitySetCaption Property	• Query Object
EntitySetCaptionML Property	• Query Object
EntitySetName Property	• Query Object
Method Property	• Column control
ObsoleteState Property	• Query Object • Column control • DataItem control • Filter control
ObsoleteReason Property	• Query Object • Column control • DataItem control • Filter control
OrderBy Property	• Query Object
Permissions Property	• Query Object
QueryCategory Property	• Query Object
QueryType Property	• Query Object
ReadState Property	• Query Object
ReverseSign Property	• Column control
TopNumberOfRows Property	• Query Object

See Also

Properties
Query Object
Table and Table Extension Properties
Page and Page Extension Properties Overview

Report Properties

2/17/2021 • 2 minutes to read • Edit Online

This topic lists properties of the report object.

PROPERTY NAME	APPLIES TO
AccessByPermission Property	• Report Object
AdditionalSearchTerms Property	• Report object
AdditionalSearchTermsML Property	• Report object
AllowScheduling Property	• Report object
ApplicationArea Property	• Report Object
AutoCalcField Property	• Column controls
AutoFormatExpression Property	• Column controls
AutoFormatType Property	• Column controls
Caption Property	• Report Object • Report Labels • Column controls
CaptionML Property	• Report Object • Report Labels • Column controls
CalcFields Property	• DataItem control
ContextSensitiveHelpPage Property	• Report Object
DataAccessIntent Property	• Report Object
DataItemLink Property (Reports)	• DataItem controls

PROPERTY NAME	APPLIES TO
DataItemTableView Property	• DataItem controls
DecimalPlaces Property	• Column controls
DefaultLayout Property	• Report Object
Description Property	• Report Object • Column controls • Report Labels • DataItem controls
EnableExternalAssemblies Property	• Report Object
EnableExternalImages Property	• Report Object
EnableHyperlinks Property	• Report Object
ExecutionTimeout Property	• Report Object
IncludeCaption Property	• Column controls
MaximumDataSetSize Property	• Report Object
MaximumDocumentCount Property	• Report Object
MaxIteration Property	• DataItem controls
ObsoleteReason Property	• Report Object • DataItem controls • Column controls
ObsoleteState Property	• Report Object • DataItem controls • Column controls
OptionCaption Property	• Column controls
OptionCaptionML Property	• Column controls
OptionMembers Property	• Column controls

PROPERTY NAME	APPLIES TO
OptionString Property	• Column controls
PaperSourceDefaultPage Property	• Report Object
PaperSourceFirstPage Property	• Report Object
PaperSourceLastPage Property	• Report Object
PDFFontEmbedding Property	• Report Object
Permissions Property	• Report Object
PreviewMode Property	• Report Object
PrintOnlyIfDetail Property	• DataItem controls
ProcessingOnly Property	• Report Object
RDLCLayout Property	• Report Object
RequestFilterFields Property	• DataItem controls
RequestFilterHeading Property	• DataItem controls
RequestFilterHeadingML Property	• DataItem controls
ShowPrintStatus Property	• Report Object
SourceExpr Property	• Column controls
TransactionType Property	• Report Object
UsageCategory Property	• Report Object
UseRequestPage Property	• Report Object
UseTemporary Property (Reports)	• DataItem controls

PROPERTY NAME	APPLIES TO
UseSystemPrinter Property	• Report Object
WordLayout Property	• Report Object
WordMergeDataItem Property	• Report Object

See Also

Properties
Table and Table Extension Properties
Page and Page Extenstion Properties Overview

XMLport Properties

2/17/2021 • 2 minutes to read • Edit Online

This topic lists properties of the XMLport object, element, and attribute.

PROPERTY NAME	XMLPORT OBJECT
AutoCalcField Property	• Field elements • Field attributes
AutoReplace Property	• Table elements
AutoSave Property	• Table elements
AutoUpdate Property	• Table elements
CalcFields Property	• Table elements
Caption Property	• XMLport object
CaptionML Property	• XMLport object
ContextSensitiveHelpPage Property	• XMLport Object
DefaultFieldsValidation Property	• XMLport object
DefaultNamespace Property	• XMLport object
Direction Property	• XMLport object
Encoding Property	• XMLport object
FieldDelimiter Property	• XMLport object
FieldSeparator Property	• XMLport object
FieldValidate Property	• Field elements • Field attributes

PROPERTY NAME	XMLPORT OBJECT
FileName Property	• XMLport object
Format Property	• XMLport object
ID Property	• XMLport object
InlineSchema Property	• XMLport object
LinkFields Property	• Table elements
LinkTable Property	• Table elements
LinkTableForceInsert Property	• Table elements
MaxOccurs Property	• Text elements • Table elements • Field elements
MinOccurs Property	• Text elements • Table elements • Field elements
Name Property	• XMLport object
NamespacePrefix Property	• XMLport object
Namespaces Property	• XMLport object
NodeName Property	• Text elements • Table elements • Field elements • Text attributes • Field attributes
ObsoleteState Property	• XMLport object
ObsoleteReason Property	• XMLport object
Occurrence Property	• Text attributes • Field attributes

PROPERTY NAME	XMLPORT OBJECT
Permissions Property	• XMLport object
PreserveWhiteSpace Property	• XMLport object
RecordSeparator Property	• XMLport object
ReqFilterFields Property	• Table elements
ReqFilterHeading Property	• Table elements
ReqFilterHeadingML Property	• Table elements
SourceField Property	• Field elements • Field attributes
SourceTable Property (XMLports)	• Table elements
SourceTableView Property (XMLports)	• Table elements
TableSeparator Property (XMLports)	• XMLport object
TextEncoding Property (XMLports)	• XMLport object
TextType Property	• Text elements • Text attributes
TransactionType Property	• XMLport object
Unbound Property	• Text elements • Field elements
UseDefaultNamespace Property	• XMLport object
UseLax Property	• XMLport object
UseRequestPage Property	• XMLport object
UseTemporary Property (XMLports)	• Table elements

PROPERTY NAME	XMLPORT OBJECT
Width Property (XMLport)	• Text elements • Table elements • Field elements • Text attributes • Field attributes
XmlName Property	• Text elements • Table elements • Field elements • Text attributes • Field attributes

See Also

Properties
Table and Table Extension Properties
Page and Page Extension Properties Overview
Codeunit Properties
Query Properties
Report Properties
Table Properties
XMLPort Properties
Enum Properties

Control Add-In Properties

2/17/2021 • 2 minutes to read • Edit Online

The following topic lists properties that apply to the Control Add-In Object.

PROPERTY NAME	OBJECT
VerticalShrink	Control add-in
HorizontalShrink	Control add-in
MinimumHeight	Control add-in
MinimumWidth	Control add-in
MaximumHeight	Control add-in
MaximumWidth	Control add-in
VerticalStretch	Control add-in
HorizontalStretch	Control add-in
RequestedHeight	Control add-in
RequestedWidth	Control add-in

See also

Properties
Table and Table Extension Properties
Page and Page Extension Properties Overview
Codeunit Properties
Query Properties
Report Properties
Table Properties
XMLPort Properties
Enum Properties

Enum Properties

2/17/2021 • 2 minutes to read • Edit Online

This topic lists properties that apply to the Extensible Enums.

PROPERTY NAME	APPLIES TO
Access Property	• Enum objects
AssignmentCompatibility Property	• Enum objects
AssignmentCompatibilityReason Property	• Enum objects
Caption Property	• Enum objects • Enum value
CaptionML Property	• Enum objects • Enum value
DefaultImplementation Property	• Enum objects
Extensible Property	• Enum objects
Implementation Property	• Enum value
ObsoleteReason Property	• Enum objects • Enum value
ObsoleteState Property	• Enum objects • Enum value
ObsoleteTag Property	• Enum objects • Enum value

See Also

Developing Extensions
AL Development Environment
Properties

View Properties

2/17/2021 • 2 minutes to read • Edit Online

The following topic lists properties that apply to Views specifically.

PROPERTY NAME	OBJECT
Filters	Views
OrderBy	Views
SharedLayout	Views

See also

Codeunit Properties
Page Properties
Query Properties
Report Properties
Table Properties
XMLPort Properties

Integrating Microsoft Dataverse for Extension Development

2/17/2021 • 2 minutes to read • Edit Online

NOTE

Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

Develop extensions and streamline the workflow by synchronizing Microsoft Dataverse data with Dynamics 365 Business Central.

For developing extensions to integrate with sales data, you simply enable the tables used in Microsoft Dataverse. The extension development process includes the following set of properties in Dynamics 365 Business Central to enable field mapping. You can enable the field mapping by using the following properties. The tables are extensible, so that you can update Microsoft Dataverse with data as well.

Associated table and field properties

The following properties are used for integrating with Microsoft Dataverse:

PROPERTIES	APPLIES TO	DESCRIPTION
TableType Property	Tables	Specifies the table type. This enables the table to integrate with the external database. For example, `CDS`.
ExternalName Property	Tables, Fields	Specifies the name of the original table in the external database when used as a table property. Specifies the field name of the corresponding field specified in the external table when used as a field property.
ExternalAccess Property	Fields	Specifies the access to the underlying Microsoft Dataverse table when Microsoft Dataverse tables are generated using the **AL Table Proxy Generator** tool, see AL Proxy Table Generator
ExternalType Property	Fields	Specifies the data type of the corresponding column in the Microsoft Dataverse table.

PROPERTIES	APPLIES TO	DESCRIPTION
OptionMembers Property	Fields	Sets the option values for a field, text box, or variable.
OptionOrdinalValues Property	Fields	Specifies the list of option values. You can set this property, if the **ExternalType** is set to **Picklist**.

Enabling the table

Typically in Microsoft Dataverse, tables handle the internal processes. In order to access to the underlying Microsoft Dataverse table, you use the `TableType` property and select the value called **CDS**. This enables the table as an integration table for integrating Dynamics 365 Business Central with Microsoft Dataverse. The table is mainly based on a table in Microsoft Dataverse, such as the Accounts table.

Snippet support

Typing the shortcut `ttable` will create the basic layout for a table object when using the AL Language extension in Visual Studio Code.

Example

In the following example, the `SalesIntegration` table uses the `TableType` and `ExternalName` properties to link the underlying **Microsoft Dataverse** table for mapping the columns from the `Sales` table with the specified fields.

```
table 50100 SalesIntegration
{
    TableType = CDS;
    ExternalName = 'Sales';

    fields
    {
        field(1; ActualSales; Integer)
        {
            ExternalName = 'ActualSale';
            ExternalAccess = Full;
            ExternalType = 'String';
        }

        field(2; SalesCategories; Option)
        {
            ExternalName='SalesCategory';
            ExternalAccess = Read;
            ExternalType = 'Picklist';
            OptionMembers = Manufacturing, Marketing, Support;
            OptionOrdinalValues = -1, 1, 2;
        }
    }
}
```

See Also

Table Properties
TableType Property
AL Proxy Table Generator

Triggers Overview

2/17/2021 • 2 minutes to read • Edit Online

The following sections describe the triggers that are available for the different AL objects:

- Table and Field Triggers
- Page and Action Triggers
- Codeunit Triggers
- Report and Data Item Triggers
- XMLport Triggers
- Query Triggers

See Also

AL Method Reference
Properties

Table and Field Triggers

2/17/2021 • 2 minutes to read • Edit Online

Dynamics 365 Business Central recognizes certain actions that happen to a table when you use it, for example, when you insert or modify data. In response, you specify to execute AL code defined in a trigger. Triggers are predefined methods that are executed when certain actions happen. The bodies of these methods are initially empty and must be defined by the developer. Defining AL code in triggers allows you to change the default behavior of Dynamics 365 Business Central.

The triggers in a table can be divided into two categories:

- Table triggers
- Field triggers

Tables have the following triggers.

TABLE TRIGGER	RUNS WHEN
OnInsert Trigger	A new record is inserted into the table.
OnBeforeInsert Trigger	A new record is inserted into the table.
OnAfterInsert Trigger	A new record is inserted into the table.
OnModify Trigger	A record in the table is modified.
OnBeforeModify Trigger	A record in the table is modified.
OnAfterModify Trigger	A record in the table is modified.
OnDelete Trigger	A record in the table is deleted.
OnBeforeDelete Trigger	A record in the table is deleted.
OnAfterDelete Trigger	A record in the table is deleted.
OnRename Trigger	A record is modified in a primary key field.
OnBeforeRename Trigger	A record is modified in a primary key field.
OnAfterRename Trigger	A record is modified in a primary key field.

Fields have the following triggers.

FIELD TRIGGER	RUNS WHEN
OnValidate (Fields) Trigger	Data is entered in a field or when the VALIDATE (Record) is executed.
OnLookup (Fields) Trigger	Lookup is activated.

FIELD TRIGGER	RUNS WHEN
OnBeforeValidate (Fields) Trigger	Before data is entered in a field.
OnAfterValidate (Fields) Trigger	After data is entered in a field.

See Also

Table Object
Table Extension Object
Triggers
Table and Table Extension Properties

Page and Action Triggers

2/17/2021 • 2 minutes to read • Edit Online

Page triggers allow you to use AL code to control the behavior of the system as a result of an event on the page, such as a page opening or a field changing its value. You typically use page triggers for advanced validation and logic.

Page triggers can be divided into three categories:

- General page triggers that apply to the entire page
- Field page triggers that apply to a field control on a page
- Action triggers that apply to an action on a page.

> **IMPORTANT**
> If you define two methods that have the same name, one defined in a page and the other in a table that is referenced by the page, you cannot invoke the method defined in the page directly. By default, a call to the method invokes the method that is defined in the table. This behavior occurs when the method is called from a source expression or a trigger.

General Triggers

The following table lists triggers that apply to the entire page.

PAGE TRIGGER NAME	RUNS
OnInit Trigger	When the page is loaded, but before the controls are available.
OnOpenPage Trigger	When the page is initialized and the controls are available.
OnClosePage Trigger	When the page about to close and after OnQueryClosePage Trigger trigger.
OnFindRecord Trigger	When the page is opened and a record is retrieved from a table.
OnNextRecord Trigger	When the page changes from displaying one record to another record in a table. For example, on a Customer card page, this happens when a user selects **Next** (Ctrl+Page Down) or **Previous** (Ctrl+Page Up).
OnAfterGetCurrRecord Trigger	After the current record is retrieved from the table.
OnAfterGetRecord Trigger	When a record has been retrieved but not yet displayed.
OnNewRecord Trigger	When a new record has been initialized but not yet displayed.
OnInsertRecord Trigger	When a new record is about to be inserted in the table.

PAGE TRIGGER NAME	RUNS
OnModifyRecord Trigger	When a record is about to be modified in the table.
OnDeleteRecord Trigger	When a record is about to be deleted from the table.
OnQueryClosePage Trigger	When the page is about to close, but before the OnClosePage Trigger.

Field Triggers

The following table describes the triggers that are available on field controls.

CONTROL TRIGGER	RUNS
OnValidate (Page fields) Trigger	When the user changes the value in a field and then selects away from the field so that the field loses focus.
OnLookup (Page fields) Trigger	When the user requests a lookup by clicking a field's lookup button or pressing F4.
OnDrillDown Trigger	When the user requests a drill-down by choosing the field's drill-down button or pressing Shift+F8.
OnAssistEdit Trigger	When the user requests assist-edit by choosing an AssistEdit button or by pressing Shift+F4.

Action Triggers

The following table lists triggers that apply to actions on a page.

TRIGGERS	RUNS
OnAction Trigger	When an action is initiated on a page.

Page Background Triggers

The following table lists triggers that apply to page background tasks. For more information, see Page Background Tasks.

TRIGGERS	RUNS
OnPageBackgroundTaskCompleted	Runs after a page background task has successfully completed.
OnPageBackgroundTaskError	Runs when an error occurs in a page background task.

See Also

Triggers

Codeunit Triggers

2/17/2021 • 2 minutes to read • Edit Online

The following triggers apply to codeunits.

CODEUNIT TRIGGER NAME	RUNS
OnRun Trigger	When the codeunit is executed.
OnBeforeTestRun Trigger	Before a test method of a test codeunit is run.
OnAfterTestRun Trigger	After a test method of a test codeunit is run.
OnCheckPreconditionsPerCompany Trigger	Before an extension upgrade of an upgrade codeunit is run.
OnCheckPreconditionsPerDatabase Trigger	Before an extension upgrade of an upgrade codeunit is run.
OnUpgradePerCompany Trigger	When an extension upgrade of an upgrade codeunit is run.
OnUpgradePerDatabase Trigger	When an extension upgrade of an upgrade codeunit is run.
OnValidateUpgradePerCompany Trigger	After an extension upgrade of an upgrade codeunit is run.
OnValidateUpgradePerDatabase Trigger	After an extension upgrade of an upgrade codeunit is run.
OnInstallAppPerCompany Trigger	When an extension installation or reinstallation in an install codeunit is run.
OnInstallAppPerDatabase Trigger	When an extension installation or reinstallation in an install codeunit is run.

See Also

Triggers

Report and Data Item Triggers

2/17/2021 • 2 minutes to read • Edit Online

In reports, triggers are typically used to perform calculations and verification. Triggers let you control how data is selected and retrieved in a more complex and effective way than you can achieve by using properties.

Report Triggers

The following table lists triggers that apply to the report itself.

TRIGGER	RUNS
OnInitReport Trigger	When the report is loaded.
OnPreReport Trigger	Before the report is run, but after the **RequestPage** has been run.
OnPostReport Trigger	After the report has run, but not if the report was stopped manually or by the QUIT Method (Report, XMLport).

Data Item Triggers

The following table lists triggers that apply to each data item on the report.

TRIGGER	RUNS
OnPreDataItem Trigger	Before the data item is processed, but after the associated variable has been initialized.
OnAfterGetRecord (Data Items) Trigger	When a record has been retrieved from the table.
OnPostDataItem Trigger	When the data item has been iterated for the last time.

See Also

Report Triggers
Triggers

XMLport Triggers

2/17/2021 • 2 minutes to read • Edit Online

The following triggers apply to XMLports.

XMLport triggers

XMLPORT TRIGGER	RUNS
OnAfterAssignField Trigger	Runs after a field has been assigned a value and before it is validated and imported. This trigger is only used to import data.
OnAfterAssignVariable Trigger	Runs after the value defined in the XML document is assigned to the text variable. This trigger is only used to import data.
OnAfterGetField Trigger	Runs after a field is passed to the XML document. This trigger is only used to export data.
OnAfterGetRecord (XMLports) Trigger	Runs after a record is retrieved from a table and before it is exported to the XML document. This trigger is only used to export data.
OnAfterInitRecord Trigger	Runs after a record is loaded. This trigger is only used to import data.
OnAfterInsertRecord Trigger	Runs after a record has been inserted into a database table. This trigger is only used to import data.
OnAfterModifyRecord Trigger	Runs after a record has been modified. The trigger is used to import data.
OnBeforeInsertRecord Trigger	Runs after a record has been loaded and before it is inserted into a database table. This trigger is only used to import data.
OnBeforeModifyRecord Trigger	Runs before a record is modified. This trigger is used to import data.
OnBeforePassField Trigger	Runs before a field is passed to the XML document. This trigger is only used to export data.

XMLPORT TRIGGER	RUNS
OnBeforePassVariable Trigger	Runs after the source expression has been formatted into a text variable and before the text variable is passed to the XML document. This trigger is only used to export data.
OnInitXMLport Trigger	Executes when the XMLport is loaded and before any table views and filters are set.
OnPreXMLport Trigger	Runs after the table views and filters are set and before the XMLport is run.
OnPostXMLport Trigger	Runs after the XMLport is run.
OnPreXMLItem Trigger	Runs after the table is initialized and before you start exporting data to an XML object. This trigger only applies to XMLport elements that have a source type of Table. This trigger is only used to export data.

See Also

XMLPort Object
Triggers
XMLPort Properties

Query Triggers

2/17/2021 • 2 minutes to read • Edit Online

This topic describes the AL triggers that are available for queries. Triggers are typically used to perform calculations and verification. Triggers let you control how data is selected and retrieved in a more complex and effective way than you can achieve by using properties.

Query object triggers

The following table lists the triggers that apply to the query object.

TRIGGER	RUNS
OnBeforeOpen	Before the query object is run and the dataset is generated. For example, you can use the **OnBeforeOpen** trigger to apply filters using the **SETFILTER** method.

See Also

Query Object
Triggers
SETFILTER Method (Query)
Report Triggers

Security and Protection in Business Central

2/17/2021 • 2 minutes to read • Edit Online

An enterprise business solution must have a built-in security system that helps protect your database and the information that it contains from unauthorized access. It must also allow you to specify what authorized users are allowed to do in the database, such as what data they can read and modify. The following sections help you understand and improve the security of Business Central.

Application Security

Security Tips for Business Users

Online Security

On-Premises Security

IMPORTANT

Starting March, 2020, Business Central only supports Transport Layer Security (TLS) version 1.2 or later. This applies to Business Central 2019 release wave 2 online and on-premises (version 15.x). Upgrade to Business Central 2019 release wave 2 or later to remove support for earlier versions of TLS. If your solution or an add-on uses TLS 1.0 or 1.1, you must update that configuration or add-on to TLS 1.2 or later as soon as possible. For more information, see Blog post: Update to TLS 1.2 or later in Dynamics 365 Business Central in 2019 release wave 2.

Security Tips for Business Users

2/17/2021 • 2 minutes to read • Edit Online

This article describes how you can work with end-users and configure their devices to improve the security.

> [IMPORTANT] Use this article together with industry standard security practices for securing users and their access to company data. This article describes additional considerations for how you can work with end-users and configure their devices to improve security.

Kiosks and shared devices

Customers using kiosks, where multiple users sign into Business Central with their own identity from that terminal, should apply additional security practices for an appropriate level of isolation between users.

- Before ending a session at the kiosk, business users should remember to sign out of Business Central, then close all browser windows. Closing the Business Central browser tab or closing the browser without signing out may not fully complete the signout process.
- Each user should use private or guest browsing modes so that any data cached by Business Central is discarded when the browser is closed. Private or guest browsing may degrade some Business Central features and performance optimizations that are only available when the browser provides access to its storage mechanisms.

See Also

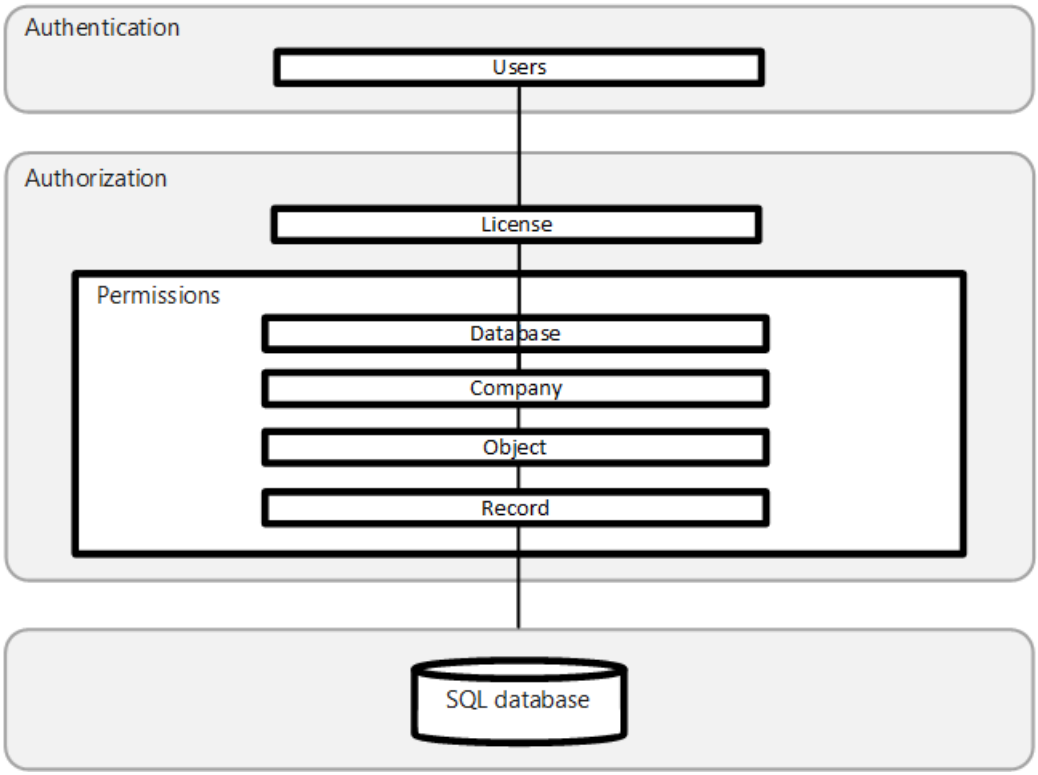
Security and Protection
Application Security
Online Security
On-Premises Security

Application Security in Business Central

2/17/2021 • 2 minutes to read • Edit Online

This section helps you understand and improve the security of your Business Central application regardless of where it is hosted. In the articles listed below, you will find guidance and recommended practices related to authentication, authorization, and auditing, as well as data encryption and secure development practices that can be applied to any Business Central environment.

Business Central uses a layered approach to application security, as outlined in the following diagram.



Authentication

Before users can sign-in to the Business Central application, they must be authenticated as valid user in the system. Business Central On-Premises supports several authentication methods, such as Windows and Azure Active Directory. Business Central Online uses strictly Azure Active Directory (Azure AD). For more information, see the following articles:

Managing Users and Permissions

Authentication and Credential Types

Multi-factor Authentication

The authentication method configured for Business Central Server is also used to access web services. For more information, see Web Services Authentication.

Authorization

Once authenticated, authorization determines which areas a user can access, such as the pages and reports that they can open, and the permissions that they have on associated data. For more information, see the following articles:

User Permissions in the Application

Analyzing Permission Changes Trace Telemetry

Data Security

Removing Elements from the User Interface According to Permissions

Analyzing Authorization Telemetry

Using OAuth to Authorize Business Central Web Services

Auditing

Business Central includes several auditing features that help you track information about who is signing-in, what their permissions are, what data have they changed, and more. For more information, see the following articles:

Authorization Assessment

Data Auditing

Security Auditing

Data Classification

Data Encryption

You can encrypt data on the Business Central server by generating new or importing existing encryption keys that you enable on the Business Central server instance that connects to the database. For more information, see Encrypting Data in Dynamics 365 Business Central.

Security Development Lifecycle

Microsoft's Security Development Lifecycle (SDL) is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost. For more information, see Security Development Lifecycle.

See Also

Security and Protection
Security Tips for Business Users
Online Security
On-Premises Security

Business Central Online Security

2/17/2021 • 2 minutes to read • Edit Online

This section helps you understand and improve the security of your Dynamics 365 Business Central tenant. In the links below you will find information, guidance and recommended practices related to authentication, data encryption and safely integrating with other services. You will also find information on Business Central's certifications and regulatory compliance.

Authentication

Business Central Online uses Azure Active Directory (Azure AD) as the authentication method, which is automatically set up and managed for you.

Data isolation and encryption

Data belonging to a single tenant is stored in an isolated database and is never mixed with data from other tenants. This ensures complete isolation of data in day-to-day use as well as in backup-restore scenarios. Furthermore, Business Central Online uses encryption to help protect tenant data:

- Data is encrypted at-rest by using Transparent Data Encryption (TDE) and backup encryption.
- Data backups are always encrypted.
- All network traffic inside the service is encrypted by using industry standard encryption protocols.

Service integration

We recommend that you use encrypted network protocols to connect to the PowerBI server and Business Central web services. For more information, see the following articles:

Connect to Business Central with Power BI

Using Security Certificates with Business Central On-Premises

See Also

Microsoft Trust Center (what we do to make the service secure)
Microsoft Dynamics 365 Cloud Services Compliance
Security and Protection
Security Tips for Business Users

Business Central On-Premises Security

2/17/2021 • 2 minutes to read • Edit Online

This section helps you understand and improve the security of Business Central hosted on-premises. In the links below you will find information, hardening guidance and recommended best practices addressing client, database, server and network security.

Authentication

Before users can sign-in to the Business Central application, they must be authenticated as valid user in the system. Business Central supports several authentication methods. You configure the authentication method on the server-tiers of Business Central.

For more information, see Authentication and Credential Types.

Server Security

Business Central Server handles communication between clients and databases, controlling authentication, event logging, scheduled tasks, reporting and more. The following articles explain how to improve the security of Business Central Server instances.

Hardening Business Central Server Security

Locking Down Server Communication settings

Client Security

The following articles explain how to improve the security of connections from the clients to the Business Central Server.

Configuring SSL to Secure the Client Connections

Using Security Certificates with Business Central On-Premises

Database Security

The articles in this section explain how to improve database security in Business Central.

The following articles discuss configurations that you can perform on the Business Central Server:

Configuring the Database

Encrypt Traffic

The following are general articles about SQL Server security that can also help secure the database:

Upgrade to TLS 1.2

IMPORTANT

Starting March, 2020, Business Central only supports Transport Layer Security (TLS) version 1.2 or later. This applies to Business Central 2019 release wave 2 online and on-premises (version 15.x). Upgrade to Business Central 2019 release wave 2 or later to remove support for earlier versions of TLS. If your solution or an add-on uses TLS 1.0 or 1.1, you must update that configuration or add-on to TLS 1.2 or later as soon as possible. For more information, see Blog post: Update to TLS 1.2 or later in Dynamics 365 Business Central in 2019 release wave 2.

Data Encryption at Rest

SQL Server Hardening

SQL Server Auditing

Backup Encryption

Azure Database Security Best Practices

Network Security

The following articles explain to secure client, web service, and PowerBI connections over a wide area network using HTTPS and security certificates.

Configuring SSL to Secure the Client Connections

Using Security Certificates with Business Central On-Premises

Connect to Business Central with Power BI

See Also

Security and Protection
Data Security
Security Tips for Business Users

Privacy FAQ for Business Central Customers

2/17/2021 • 2 minutes to read • Edit Online

Here you find links that can help you find answers to questions concerning privacy when you use Business Central.

- Privacy and personal data for Microsoft Dynamics 365: Privacy and personal data for Microsoft Dynamics 365
- International availablity for Business Central: Country/regional availability and supported languages
- Customer definitions on Microsoft Trust Center: How Microsoft categorizes data for online services
- Site for legal terms for all Microsoft products: Product Terms
- Azure Geo to Azure regional mapping: Azure geographies
- Dynamics 365 and Power Platform availability: Dynamics 365 and Power Platform availability
- Get the PDF file that shows the international availability of Dynamics 365:International availability of Dynamics 365

See also

Security and Protection in Business Central

Service Overview for Business Central Online

2/17/2021 • 5 minutes to read • Edit Online

Dynamics 365 Business Central is a complete enterprise resource planning (ERP) software solution for mid-sized organizations that is fast to implement, easy to configure, and simple to use, both on-premises and online. Sign up for a trial before you decide to move to the cloud, and read this article to learn about the systems that make Business Central online run as a service that you can bet your business on.

Lifecycle policy

Business Central online is governed by Microsoft's Modern Lifecycle Policy, which means continuous service updates and a major update every six months. For more information, see Dynamics 365 Business Central Service Compliance and Dynamics 365 release schedule and early access.

Get an overview of new and upcoming changes in the Dynamics 365 release plans.

Global availability

Business Central online is available in a number of markets, and new countries go live on a quarterly basis. For more information, see Countries and Translations Supported.

Built upon a foundation of trust, security, and compliance

On the Service Trust site, review the available independent audit reports for Microsoft's Cloud services. Find information about compliance with data protection standards and regulatory requirements, such as International Organization for Standardization (ISO), Service Organization Controls (SOC), National Institute of Standards and Technology (NIST), Federal Risk and Authorization Management Program (FedRAMP), and the General Data Protection Regulation (GDPR).

For example, you can find audit reports and certifications for ISO27001, ISO27018, ISO27017, ISO 27701, or the SOC 2 Type II reports at https://servicetrust.microsoft.com/.

Databases and backups

Business Central online runs on Azure and uses Azure SQL Database as the database that stores your data. A tenant's data is stored at rest in the Azure region that is closest to their geographical location. Administrators can always find the exact Azure region that hosts their environments in the Business Central administration center. For an overview of Azure geographies and regions, see the Azure global infrastructure site.

Databases are protected by automatic backups that are kept for 30 days. Administrators of a Business Central tenant can't directly access or manage these backups because they're managed automatically by Microsoft. But admins can restore their environments to a specific point in time in the past using the Business Central admin center. For more information, see Restoring an environment and Automated backups - Azure SQL Database.

Your data is safe with us. Should anything go wrong, different resources within Microsoft take action. If an Azure region is temporarily unavailable, for example, the customers' data is preserved using automatic geo-redundant backups, so your data becomes available again once the Azure region is back online. All these steps are triggered automatically. Read a general description of how Azure maintains their service level agreements at High availability for Azure SQL Database and SQL Managed Instance. In extreme cases, such as if the region would be expected to be offline longer, we would start the process of restoring the data from the various Business Central environments geo-redundant backups into another region within the same geography. Although such cases happen rarely, recovering data into another Azure region is a standard, well-described

internal procedure, which we practice regularly as a part of our audits and internal drills.

Similarly, we apply automated tasks for performance tuning, high availability, disaster recovery, and backups. For more information about these types of task, see the Azure SQL docs.

Service level agreements (SLA)

Business Central online is governed under the Modern Lifecycle Policy. The service level agreement terms are described in the document that you can download from the **Service Level Agreements for Microsoft Online Services** section on the Licensing terms page.

Furthermore, administrators can monitor a tenant's health and specify upgrade windows in the Business Central Administration Center.

Databases are protected by automatic backups that are kept for 30 days. Administrators of a Business Central tenant can't directly access or manage these backups because they're managed automatically by Microsoft. But admins can restore their environments to a specific point in time in the past using the Business Central admin center. For more information, see Restoring an environment and Automated backups - Azure SQL Database.

For more information about Azure SQL, see High availability for Azure SQL Database and SQL Managed Instance.

Service updates

Business Central online is a service that consists of a Microsoft-maintained platform and business functionality. Microsoft partners can provide more business functionality, such as to address specific industry or localization needs. Both business functionality and service components are monitored continuously and updated as appropriate. Generally, new capabilities are made available as part of major updates. Critical fixes are rolled out as soon as possible after they passed tests and have been verified in Microsoft's protected staging environment. For more information, see Major Updates of Business Central Online. You can always refer to the release plan for an overview of new and upcoming functionality, and you can keep an eye out for the minor updates at aka.ms/bclastminorupdate.

For each environment, administrators can set a maintenance window for when Microsoft is allowed to update that environment. For more information, see Managing Updates in the Business Central Admin Center. Microsoft then schedules updates of the business functionality to be applied during these maintenance windows.

An exception to this type of schedule is the continuous updates to the underlying service. The service components apply to multiple environments, such as all tenants in a region. So Microsoft schedules those updates to a time when traffic is lower in each region, typically during the evening or night. Typically, these service updates are transparent to any users, as the service is designed to manage traffic in a way that any users still working in Business Central are not affected by these service updates.

> **TIP**
> All updates that Microsoft applies to Business Central online are also shipped with the subsequent cumulative update for Business Central. For an example, see Update 17.1 for Microsoft Dynamics 365 Business Central online 2020 release wave 2.

See also

FAQ for Dynamics 365 Business Central
Administration of Business Central Online
Technical Support for Business Central

Escalating support issues to Microsoft
Data and access when a trial or subscription ends
Azure global infrastructure Azure reliability
Microsoft Service Trust

Performance Overview

2/17/2021 • 2 minutes to read • Edit Online

The following sections help you understand and improve the performance of Business Central. The content is centered around the different ways a functional consultant, a developer, or an administrator can make changes with a performance impact.

- Visit Performance Tips for Business Users to learn more about tips on how you can work with end-users to improve the performance.
- Visit How Application Configurations Affect Performance to learn about how in-product configurations affect performance.
- Visit Performance Articles for Developers to learn about how to code for performance in AL, and how to identify performance issues using the Performance Toolkit extension. The extension is available for free on AppSource.
- Visit Performance in Business Central Online to learn more about performance in the Business Central service.
- Visit Performance of Business Central On-Premises Installations to learn more about the various ways the different components in Business Central can be configured to achieve better performance for an on-premises installation.
- Visit How to Work with a Performance Problem to learn more on how to tackle a performance problem.

The video series *Performance Considerations when Building an App* is also a good starting point to learn more about performance of Business Central:

- Performance Considerations when Building an App, Part I

  https://www.youtube.com/embed/MooYL05V11Y

  This first part of performance considerations videos focuses on basic principles. It makes you aware of why it is essential to consider performance in every line of code you write, and it introduces the performance implications of the general architectural document and posting design patterns of Business Central.

- Performance Considerations when Building an App, Part II

  https://www.youtube.com/embed/VN7V4GyULtY

  Learn about best practices on implementing the existing data retrieval methods, learn how to greatly improve performance by minimizing server roundtrips, and get introduced to important dos and don'ts when coding for performance.

How Application Configurations Affect Performance

2/17/2021 • 4 minutes to read • Edit Online

The sections in this article are tips and tricks on how to set up Business Central for performance and also describe how in-product configurations affect the performance of Business Central.

Uninstall extensions that you don't use

Any extensions that you install can affect the overall system performance. So if you've installed an app from AppSource, but later discover it's not needed, then uninstall it. The same advice applies to the extension that comes preinstalled in an environment. For example, uninstall all migration extensions after you've migrated data, or if you don't intend to migrate data.

Run things in the background

It's often desirable to offload work from the user session to happen in the background. Examples are:

- Schedule long running reports to run in background
- Schedule jobs (for example posting) to run in background
- Enable background posting in areas where your business is using reservations and item tracking using serial and lot numbers
- Adjust item costs as a periodic background job. Don't adjust automatically.

> **TIP**
> don't run job queues too frequently.

Avoid locking

When the Business Central database needs to have exclusive access to a table or a data row, it will issue a lock. If another session needs to access a locked resource, it needs to wait until the session holding the lock is finished with its work. There are a few places in the Business Central application where you can reduce the risk of locking.

Use number series that allow gaps

Number series in Business Central are shared resources that sometimes cause locking issues. Not all records that you create in Business Central are financial transactions that must use sequential numbering. Customer cards, sales quotes, and warehouse activities are examples of records that are assigned a number from a number series. They aren't subject to financial auditing and can be deleted. For all such number series, consider using number series that allow gaps to avoid locking issues. For more information, see Gaps in Number Series.

Be cautious with the Rename/Copy company operations

The **Rename company** and **Copy company** operations aren't intended to run while business transactions are being applied to Business Central. First, the operations are likely to induce locks on the tables that data is copied from. These locks will block users from transacting in the company. Second, the operations use resources on the database, which can in turn cause resource starvation for users working in other companies.

If you must do a **Rename/Copy company** operation, it's highly recommended to do it outside working hours. Turn off scheduled jobs to avoid locking issues.

The **Copy Company** operation also has a number of long term effects including the following:

- Increased database size
- Upgrade operations take longer
- Larger .bacpac files when requesting backups from the Business Central administration center

  This also means that export/import operations involving .bacpac files take longer

> **NOTE**
> For Business Central online, the **Rename company** operation is no longer supported. Instead, you can change a company's display name.

Periodic activities that maintain performance

Block inactive customers, vendors, or items to improve filtering and searching on document data entry

- Block Customers
- Block Vendors
- Block Items from Sales or Purchasing

Performance effect of enabling integration on a table

There's a performance overhead involved in enabling integration on an entity such as **Customer** or **Contact**. Only enable integration if you intend to integrate with Dynamics 365 Sales. Enable it only on the required entities.

For more information, see Synchronizing Data in Business Central and Dynamics 365 Sales.

Functionality with known performance impact

These areas of the application are known to cause a performance impact and require extra testing with realistic data setup before they're rolled out.

- Security filtering mode
- Inventory Posting
- Dimensions
- Dynamic Order tracking
- Automatic reservation
- Item tracking and Lot/SN Expiration dates
- Change log

Manage the database access intent on reports, API pages, and queries

Business Central supports the **Read Scale-Out** feature in Azure SQL Database and SQL Server to load-balance analytical workloads. **Read Scale-Out** is built in to Business Central online, but it can also be enabled for on-premises.

Read Scale-Out applies to queries, reports, or API pages. With these objects, instead of sharing the primary, they can be set up to run against a read-only replica. This setup essentially isolates them from the main read-write workload. This way, they won't affect the performance of business processes.

A drawback of reading from a replica is that it introduces a slight delay compared to reading from the primary database. **Read Scale-Out** is controlled by the DataAccessControl property on objects. This property determines whether to use a replica if one is available. If this delay isn't acceptable for an object, you can overwrite the default database access intent from the UI. For more information, see Managing Database Access Intent

Don't do these things

Finally, make sure that you don't repeat these performance mistakes that we have seen cause massive performance issues for customers:

- Don't adjust cost item entries with a high frequency.
- Don't set up a change log for everything. For more information, see Auditing Changes in Business Central.
- Don't run job queues too frequently.
- Don't adjust item costs automatically if you have many item entries. Run in the background instead.
- Don't postpone setting up global dimensions, because it can be a heavy operation when you have much data. Set up correct global dimensions to avoid changing them later on.
- Don't run the **Copy company** operation during business hours.

See Also

Performance Overview
Performance Topics For Developers
Performance tips for business users
Performance Online
Performance of On-Premises Installations
How to Work with a Performance Problem

Performance Tips for Business Users

2/17/2021 • 4 minutes to read • Edit Online

This section describes how you can work with end-users to improve the performance that each individual experiences with Business Central.

Choosing a desktop browser

Business Central supports multiple browsers. Each browser offers a variety of features and capabilities. The browser plays a significant role in the responsiveness and fluidity of the user interface.

- Where possible, avoid older browsers such as Internet Explorer and Edge Legacy. Modern browsers generally offer better performance. See the list of supported and recommended browsers for Business Central online and Business Central on-premises.

  > **IMPORTANT**
  > Business Central will end support for Internet Explorer and Edge Legacy in April 2021. Consider switching to a modern browser, such as the new Microsoft Edge, before support ends.

- Always keep your browser updated to the latest version, which may include the latest performance enhancements.
- Some Business Central performance features and optimizations, such as page caching, are only available for modern browsers.

Browsers and page caching

The overall structure of a page (but not business data) is cached on the client device when a page is accessed the first time. The next time that same page is accessed, the layout of the page will be immediately displayed, while the latest data is fetched from the Business Central service. To take full advantage of page caching, Business Central requires a modern browser with access to the browser's storage mechanisms.

BROWSER	PAGE CACHING AVAILABILITY
Microsoft Edge	Available
Chrome	Available
Safari	Partly available. In some cases, users won't benefit from page caching across sessions.
FireFox	Partly available. In some cases, users won't benefit from page caching across sessions.
Edge Legacy	Unavailable
Internet Explorer	Unavailable

IMPORTANT

Browsers that run in private or guest mode typically discard browser storage when the browser window is closed. This discards cached data and users won't be able to take advantage of page caching the next time they sign into Business Central. Users relying on private or guest browsing to work across multiple identities and organizations are advised to use browser profiles instead.

Any browser policies or settings that restrict access to local storage or the browser's IndexedDB may prevent Business Central from applying performance optimizations.

Choosing a network connection

If a choice of networks is available, consider connecting to a secure network that has lower latency. In general, the user interface performs better when latency is low.

Keep powered up

Newer browsers and operating systems are generally better at handling resources such as memory, network, and storage. Some devices will automatically limit resources available to the browser when running on battery power. Consider keeping laptops and similar devices plugged in to a power source for best performance.

Making pages load faster

Business Central has numerous mechanisms that make the user interface more responsive and help pages load faster. For example:

- List pages load records in small batches while the user scrolls through the list, allowing it to scale to very large tables.
- The overall structure of a page (but not business data) is cached on the client device after a page is accessed the first time.
- The time taken to load any page is also affected by the number of controls shown on the page. Users can improve performance on complex pages using these methods:
  - By *collapsing* secondary content that may be needed only occasionally. For example, when the FactBox pane on a page is collapsed, Business Central saves time from attempting to process and display all the related facts.
  - By *hiding* secondary content entirely from the page. For example, hiding Role Center parts or columns that are not used by the user, department, or organization will also improve the time needed to load the page. Learn more about Personalizing Your Workspace.

Agility of navigating and entering data

Raw speed is not the only factor that determines whether users have a pleasant and performant experience. Business Central provides numerous features that increase efficiency when exploring, navigating and entering data:

- Keyboard Shortcuts
- Focusing on lists
- Quick Entry
- Finding Pages and information with Tell Me
- Finding Pages and Reports with the Role Explorer
- Optimize your workspace for data entry
- Entering or editing using Microsoft Excel

Attaching pictures and other files

High quality images or large documents can take time to process or download from Business Central. Consider the following that may improve performance when working with files:

- Reduce the size or quality of any images or photos that you upload. Some addon apps for Business Central that capture images from your device may also include settings for image quality.
- To avoid having to download files repeatedly simply to preview them, consider storing files on external storage that allows previewing of pictures and documents, such as SharePoint Online. You can then create a link from a record in Business Central to that file. Generally reducing the number of images and documents stored in Business Central also lightens the load of routine database maintenance tasks.
  Learn how to Manage Attachments, Links, and Notes on Cards and Documents.

Searching and filtering records

Search in lists searches all columns in a table. To avoid resource starvation on broad data searches, a search might be subject to a timeout in which case the user will see a *"Searching for rows is taking too long. Try to search or filter using different criteria."* message.

Users experiencing slow search in lists should consider using a column filter instead. Learn about Searching and Filtering.

See Also

Performance Overview
Configuring the application for performance
Performance Topics For Developers
Performance Online
Performance of On-Premises Installations
How to Work with a Performance Problem

Performance Articles For Developers

2/17/2021 • 16 minutes to read • Edit Online

In this article, you can read about ways to tune performance when developing for Business Central.

- Writing efficient pages
- Writing efficient Web Services
- Writing efficient reports
- AL performance patterns
- Efficient Data access
- Testing and validating performance
- Tuning the Development Environment

Writing efficient pages

There are many patterns that a developer can use to get a page to load faster. Consider the following patterns:

- Avoid unnecessary recalculation
- Do less
- Offloading the UI thread

Pattern - Avoid unnecessary recalculation

To avoid unnecessary recalculation of expensive results, consider caching the data and refresh the cache regularly. Let's say you want to show the top five open sales orders or a VIP customers list on the role center. The content of such a list probably doesn't change significantly every hour. There's no need to calculate that from raw data every time the page is loaded. Instead, create a table that can contain the calculated data and refresh every hour/day using a background job.

Another example of unexpected recalculation is when using query objects. In contrast to using the record API, query results aren't cached in the primary key cache in the Business Central server. Any use of a query object will always go to the database. So, sometimes it's faster to not use a query object.

Pattern - Do less

One way to speed up things is to reduce the work that the system must do. For example, to reduce slowness of role centers, consider how many page parts are needed for the user. Another benefit of a simple page with few UI elements can also be ease of use and navigation.

Remove calculated fields from lists if they aren't needed, especially on larger tables. Also, if indexing is inadequate, calculated fields can significantly slow down a list page.

Consider creating dedicated lookup pages instead of the normal pages when adding a lookup (the one that looks like a dropdown) from a field. Default list pages will run all triggers and FactBoxes even if they aren't shown in the lookup. For example, Business Central 2019 release wave 1 added dedicated lookup pages for Customer, Vendor, and Item to the Base Application.

Pattern - Offloading the UI thread

To get to a responsive UI fast, consider using Page Background Tasks for calculated values, for example, the values shown in cues.

For more information about Page Background Tasks, see Page Background Tasks.

Writing efficient Web Services

Business Central supports for Web services to make it easier to integrate with external systems. As a developer, you need to think about performance of web services both seen from the Business Central server (the endpoint) and as seen from the consumer (the client).

Endpoint performance

Anti-patterns (don't do this)

Avoid using standard UI pages to expose as web service endpoints. Many things, like FactBoxes, aren't exposed in OData, but will use resources to compute.

Things that have historically caused performance issues on pages that are exposed as endpoints are:

- Heavy logic in `OnAfterGetCurrRecord`
- Many SIFT fields
- FactBoxes

Avoid exposing calculated fields, because calculated fields are expensive. Try to move them to a separate page or to refactor the code so the value is stored on the physical table (if applicable). Complex types are also a performance hit because they take a lot of time to calculate.

Don't use temp tables as a source if you have many records. Temp tables that are based on APIs are a performance hit. The server has to fetch and insert every record, and there's no caching on data in temp tables. Paging becomes difficult to do in a performant manner. A rule of thumb is if you have more than 100 records, don't use temp tables.

Don't insert child records belonging to same parent in parallel. This condition causes locks on parent and Integration Record tables because parallel calls try to update the same parent record. The solution is to wait for the first call to finish or use $batch, which will make sure calls get executed one after another.

Performance patterns (do this)

- Instead of exposing UI pages as web service endpoints, use the built-in API pages because they've been optimized for this scenario. Select thehighest API version available. Don't use the beta version of the API pages. To read more about API pages, see API Page Type.
- The choice of protocol for the endpoint can have a significant impact on performance. Favor OData version 4 for the fastest performance. It's possible to expose procedures in a codeunit as an OData end point using unbound actions. To read more about OData unbound actions, see Creating and Interacting with an OData V4 Unbound Action.
- For OData, limit the set ($filter or $top) if you're using an expensive $expand statement. If you've moved calculated fields to a separate page, then it's good practice to limit the set to get better performance.
- If you want OData endpoints that work as data readers (like for consumption in Power BI), consider using API queries and set DataAccessIntent = ReadOnly. For more information, see API Query Type and DataAccessIntent Property.
- Use OData transaction $batch requests where relevant. They can reduce the number of requests the client needs to do when errors occur. For more information, see Tips for working with the APIs - OData transactional $batch requests.

Web service client performance

The online version of Business Central server has set up throttling limits on web service endpoints to ensure that excessive traffic can't cause stability and performance issues.

Make sure that your client respects the two HTTP status codes *429 (Too Many Requests)* and *504 (Gateway Timeout)*.

- Handling status code 429 requires the client to adopt a retry logic while providing a cool off period. You can apply different strategies, like:
  - Regular interval retry
  - Incremental intervals retry
  - Exponential back-off
  - Randomization
- Handling status code 504 - Gateway Timeout requires the client to refactor the long running request to execute within time limit by splitting the request into multiple requests. Then, deal with potential 429 codes by applying a back off strategy.

Read more about web service limits, see Working with API limits in Dynamics 365 Business Central.

The same advice applies for outgoing web service calls using the AL module HttpClient. Make sure your AL code can handle slow response times, throttling, and failures in external services that you integrate with.

Writing efficient reports

Reports generally fall into two categories. They can be specific to a single instance of an entity, like an invoice. Or, they can be of a more analytical nature that joins data from multiple instances of multiple entities. Typically, performance issues in reports lie in the latter category. The following articles contain advice about implementing faster reports:

- To use queries to implement fast reports, see Queries in Business Central.
- Compared to Word layouts, RDL layouts can result in slower performance with document reports, especially for actions related to the user interface (like sending emails). For more information, see Creating an RDL Layout Report.

Read more about how to tune RDLC reports here:

- RDLC Performance Optimization Tips

Efficient extracts to data warehouses

When establishing a data warehouse, you typically need to do two types of data extraction:

1. A historical load (all data from a given point-in-time)
2. Delta loads (what's changed since the historical load)

The fastest way to get a historical load from Business Central online is to get a database export as a BACPAC file (using the Business Central admin center) and restore it in Azure SQL Database or on a SQL Server. For on-premises installations, you can just take a backup of the tenant database.

The fastest (and the less disruptive) way to get delta loads from Business Central online is to set up API queries configured with read-scaleout and use the data audit field **LastModifiedOn** (introduced in version 17.0) on filters.

AL performance patterns

Knowledge about different AL performance patterns can greatly improve the performance of the code you write. In this section, we'll describe the following patterns and their impact on performance.

- Use built-in data structures
- Run async (and parallelize)
- Use set-based methods instead of looping

- Other AL performance tips and tricks

Pattern - Use built-in data structures

AL comes with built-in data structures that have been optimized for performance and server resource consumption. Make sure that you're familiar with them to make your AL code as efficient as possible.

When concatenating strings, make sure to use the `TextBuilder` data type and not repeated use of the `+=` operator on a `Text` variable. For more information, see TextBuilder Data Type.

If you need a key-value data structure that is optimized for fast lookups, use a `Dictionary` data type. For more information, see Dictionary Data Type.

Use a `List` data type if you need an unbound "array" (where you would previously create a temporary table object). For more information, see List Data Type.

Use the `Media` or `Mediaset` data types instead of the `Blob` data type. The `Media` and `MediaSet` data types have a couple advantages over the `Blob` data type when working with images. First of all, a thumbnail version of the image is generated when you save the data. You can use the thumbnail when loading a page and then load the larger image asynchronously using a page background task. Second, data for `Media` and `MediaSet` data types is cached on the client. Data for the `Blob` data type is never cached on the server. It's always fetched from the database.

Pattern - Run async (and parallelize)

It's often desirable to offload AL execution from the UI thread to a background session.

Here are some examples of this pattern:

- Don't let the user wait for batches
- Split large tasks into smaller tasks and run them in parallel

There are many different ways to spin up a new task:

- Job Queue
- TaskScheduler.CreateTask
- StartSession
- Page Background Task

They come with different characteristics as described in this table:

METHOD TO START A NEW TASK	PROPERTIES
Page Background Task	Can (will) be canceled Read-only Call back to parent session Lightweight
StartSession	Created immediately Runs on same server Not as controlled as a Page Background Task
Task	Queued up Any server in a cluster can start it Survives server restarts No logging

METHOD TO START A NEW TASK	PROPERTIES
Job queue	Scheduled Recurrence Any server in a cluster can start it Survives server restarts Logging of results

Pattern - Use set-based methods instead of looping

The AL methods such as `FindSet`, `CalcFields`, `CalcSums`, and `SetAutoCalcFields` are examples of set-based operations that are much faster than looping over a result set and do the calculation for each row.

- CalcFields, CalcSums, and Count
- FindSet Method

One common use of the `CalcSums` method is to efficiently calculate totals.

Try to minimize work done in the `OnAfterGetRecord` trigger code. Common performance coding patterns in this trigger are:

- Avoiding `CalcFields` calls. Defer them until the end.
- Avoiding repeated calculations. Move them outside the loop, if possible.
- Avoid changing filters. This pattern requires the server to throw away the result set.

Consider using a query object if you want to use a set-based coding paradigm. These pros and cons for using query objects:

PROS FOR USING A QUERY OBJECT	CONS FOR USING A QUERY OBJECT
- Will bypass the AL record API where server reads all fields. - With a covering index, you can get fast read performance for tables with many fields. - Can join multiple tables.	- Query object result sets aren't cached in the servers primary key (data) cache. - No writes are allowed. - You can't add a page on a query object.

Read more about query objects here:

- Using Queries Instead of Record Variables
- Query object
- Query overview
- TopNumberOfRows Property
- Query Objects and Performance

Pattern - Use partial records when looping over data or when table extension fields aren't needed

When writing AL code for which the fields needed on a record are known, you can use the partial records capability to only load out these fields initially. The remaining fields are still accessible, but they'll be loaded as needed.

Partial records improve performance in two major ways. First, they limit the fields that need to be loaded from the database. Loading more fields leads to more data being read, sent over the connection, and created on the record. Second, partial records limit the number of table extensions that need to be joined.

The performance gains compound when looping over many records, because both effects scale with the number of rows loaded.

For more information, see Using Partial Records.

Other AL performance tips and tricks

If you need a fast, non-blocking number sequence that can be used from AL, refer to the number sequence object type. Use a number sequence object if you:

- Don'twant to use a number series.
- Can accept holes in thenumberrange.

For more information, see NumberSequence Data Type.

Table extension impact on performance

Table extensions are eager-joined in the data stack when accessing the base table. It's currently not possible to define indexes that span base and extension fields. So avoid splitting your code into too many table extensions. Also, be careful about extending central tables, such as General Ledger entry, because it can severely hurt performance.

An alternative when doing data modeling for extending a table with new fields is to use a related table and define a FlowField on the base table.

Here are the pros and cons of the two data models:

DATA MODEL FOR EXTENDING A TABLE	PROPERTIES
Table extension	Fields can be added to lists and are searchable. Always loaded with the base table. Expensive at runtime but easy to use. Use only for critical fields.
Related tables	Need to set up table relations. Dedicated page for editing. Requires flow field to be shown in lists. Doesn't affect performance of base table. Excellent for FactBoxes.

Limit your event subscriptions

The following are best practices for getting performant events:

- There's no significant cost of having a publisher defined.
- Static automatic has a cost over manually binding (there's an overhead of creating and disposing objects).
- Codeunit size of the subscriber matters. Try to have smaller codeunits.
- Use single instance codeunits for subscribers, if possible.

Table events change the behavior of SQL optimizations on the Business Central Server in the following ways:

- The Business Central Server will issue SQL update/delete statements row in a for loop rather than one SQL statement.
- They impact `ModifyAll` and `DeleteAll` methods that normally do bulk SQL operations to be forced to do single row operations.

Efficient data access

Many performance issues are related to how data is defined, accessed, and modified. It's important to know how concepts in AL metadata and the AL language translate to their counterparts in SQL.

Tables and keys

Many performance issues can be traced back to missing indexes (also called keys in Business Central), but index design is often not a key skill for AL developers. For best performance, even with large amounts of data, it's imperative to design appropriate indexes according to the way your code will access data.

These articles on indexing are worth knowing as an AL developer:

- Table Keys and Performance in Business Central
- Key Property
- About SQL Server indexes

Indexes have a cost to update, so it's recommended to not use them too frequently.

SumIndexField Technology (SIFT)

SumIndexField Technology (SIFT) lets you quickly calculate the sums of numeric data type columns in tables, even in tables with thousands of records. The data type includes Decimal, Integer, BigInteger, and Duration. SIFT optimizes the performance of FlowFields and query results in a Business Central application.

Ensure appropriate SIFT indices for allFlowFields of type sum or count.

Read more about SIFT here:

- SumIndexField Technology (SIFT)
- SIFT and Performance
- Tuning and Tracing
- SIFT and SQL Server

The following article can help you find missing SIFT indexes on FlowFields:

Troubleshooting: Long Running SQL Queries Involving FlowFields by Disabling SmartSQL.

How AL relates to SQL

The AL programming language, to some degree, hides how data is read and written to the database. To effectively code for performance, you need to know how AL statements translate to the equivalent SQL statements.

The following articles cover how AL relates to SQL:

- AL Database Methods and Performance on SQL Server
- Data Access
- Data read/write performance
- Bulk Inserts

How to get insights into how AL translates to SQL

If you want to track how Business Central Server translates AL statements to SQL statements, use either database statistics in the AL debugger or telemetry on long running queries.

Read more here:

- About database statistics in the AL debugger
- Telemetry on Long Running SQL Queries

Using Read-Scale Out

Business Central supports the **Read Scale-Out** feature in Azure SQL Database and SQL Server. **Read Scale-Out** is used to load-balance analytical workloads in the database that only read data. **Read Scale-Out** is built in as part of Business Central online, but it can also be enabled for on-premises.

Read Scale-Out applies to queries, reports, or API pages. With these objects, instead of sharing the primary, they can be set up to run against a read-only replica. This setup essentially isolates them from the main read-write workload so that they won't affect the performance of business processes.

As a developer, you control **Read Scale-Out** on report, API page, and query objects by using the DataAccessControl property. For more information, see Using Read Scale-Out for Better Performance.

Testing and validating performance

It's imperative to test and validate a Business Central project before deploying it to production. In this section, you find resources on how to analyze and troubleshoot performance issues and guidance on how to validate performance of a system.

Performance Unit Testing

You can use the `SessionInformation` data type in unit tests that track the number of SQL statements or rows read. Use it before and after the code to be tested. Then, have assert statements that check for normal behavior.

For more information, see SessionInformation Data Type.

Performance Scenario and Regression Testing

Use the Performance Toolkit to simulate the amount of resources that customers use in realistic scenarios to compare performance between builds of their solutions.

The Performance Toolkit helps answer questions such as, "Does my solution for Business Central support X number of users doing this, that, and the other thing at the same time?"

For more information, see The Performance Toolkit Extension.

Performance Throughput Analysis

The Performance Toolkit doesn't answer questions such as, "How many orders can Business Central process per hour?" For this kind of analysis, test the time to execute key scenarios using the Performance Toolkit, and then use the guidance on Operational Limits for Business Central Online. For advanced analysis, consider using a queueing model such as a M/M/1 queue to answer whether the system can process the workload you intend.

Performance telemetry

The following performance telemetry is available in Azure Application Insights (if that has been configured for the environment):

- Database locks
- Long Running AL operations
- Long Running SQL Queries
- Page views
- Reports
- Sessions started
- Web Service Requests

Read more in this section: How to use telemetry to analyze performance

Troubleshooting

The following articles can be of help in troubleshooting performance issues:

- Find missing SIFT indexes for FlowFields by Disabling SmartSQL
- Use Page Inspection to find extensions participating on a page
- Viewing Table Sizes

Tuning the Development Environment

The following articles explain what you can do as a developer to tune your development environment for better performance:

- Optimizing Visual Studio Code for AL Development
- Code Analysis on large projects

See Also

Performance Overview
How Application Configurations Affect Performance
Performance Online
Performance of On-Premises Installations
How to Work with a Performance Problem
Performance tips for business users

Performance in Business Central Online

2/17/2021 • 2 minutes to read • Edit Online

These sections describe how settings in Business Central online impact the performance experience of users.

Performance on sandbox environments

The Business Central service offers the ability to test code in a sandbox environment before to deploying to a production environment.

Users often can't get the same performance and reliability in Business Central online as they get in their production environments. This discrepancy is caused by a couple factors.

Firstly, it's the nature of the operations that our users do in the sandbox environments. Some typical examples of this are:

- Frequent publishing and installation of per-tenant extensions (PTEs), which aren't yet of production quality
- Creating multiple companies for different users to try their scenarios
- Initializing test environments with data via RapidStart

Secondly, it is because the sandbox services configuration. Sandbox configurations tend to be more dense and have different thresholds than production. Read more about sandboxes in Managing Environments.

Telemetry

For monitoring and analyzing performance issues in the Business Central service, we recommend connecting Azure AppInsights to the environments that you want to get signals from. For more information, see Sending telemetry to Microsoft Azure Application Insights.

Here are some ways where telemetry can help troubleshoot performance issues:

AREA	TELEMETRY	WHY
Page Background Task	Authorization signal	Each page background task will open a new session. Any expensive action in the OnCompanyOpen trigger will slow down opening new sessions.
Sign-in	Authorization signal	Any expensive action in the OnCompanyOpen trigger will slow down opening new sessions.
Something was slow during this period of time	Company lifecycle signal	Check whether a copy-company operation was running while the performance issue occurred.
Something was slow during this period of time	Database locks signal	Maybe the performance issue was because of locking in the database.
Suddenly the XYZ page is slow	Extension lifecycle signal	Maybe an extension was installed that interferes with the page in question.

AREA	TELEMETRY	WHY
Some pages or reports are slow to load	Long running SQL queries	Investigate whether the data operations on the page or report are taking a long time to complete.
A report is slow	Report signal	Check whether the report is reading more data than you expected.
System UI feels slow	Web service requests signal	Calling your environment too aggressively with web service requests can affect performance of the system.

This page shows an overview of all currently available signals: Monitoring and Analyzing Telemetry.

Operational Limits

Large-scale cloud services use shared resources to achieve the best possible use of resources – like IO, CPU, and memory. To ensure that tenants run smoothly, limits are applied to various operations. These limits control things like: how long an operation can run before being canceled, or how many operations can run at the same time. Some of the operations include:

- Web service requests
- Report and query generation
- Client connections
- Background task execution

Without limits, one tenant could use many more resources than other tenants. Other tenants running on the same resources might experience slower performance. Although you can't change these limits, it's useful to be aware of them. For more information about the limits, see Operational Limits for Business Central Online.

See Also

Performance Overview
Performance Articles For Developers
Performance of On-Premises Installations
How to Work with a Performance Problem
Performance tips for business users

Performance of Business Central On-Premises Installations

2/17/2021 • 6 minutes to read • Edit Online

In this section, we highlight a number of resources that might be useful when doing performance investigations and tuning of on-premises installations. On-premises, in this context, means deployment to any environment that isn't the Business Central service. Running Business Central on Azure resources is also considered on-premises.

Content is ordered into the following groups:

- Tuning the technology stack
- Scaling Business Central
- Measure and monitor performance

Performance tuning the technology stack

A Business Central installation typically consists of the following components, which can be tuned to improve performance:

- Client
- Web Server
- Server (service-tier)
- Database

Client

There are three things of importance when dealing with client performance:

- Hardware of the computer running the client
- Choice of browser
- Network bandwidth and latency between the client and the data center running Business Central

These subjects are described in Performance Tips for Business Users.

Web Server

You can improve web server performance by configuring Kernel-mode authentication:

- Configuring Kernel Mode Authentication on the Business Central Web Server

Business Central Server (service-tier)

You can adjust the following Business Central Server settings related to database performance.

SERVER SETTING	DESCRIPTION	READ MORE
BufferedInsertEnabled	Disabling bulk inserts can be helpful when you're troubleshooting failures that occur when inserting records.	Disabling Bulk Inserts
DisableQueryHintForceOrder	Check if this setting is set to **true**.	Configuring Query Hints for Optimizing SQL Server Performance with Business Central

SERVER SETTING	DESCRIPTION	READ MORE
DisableQueryHintLoopJoin	Check if this setting is set to **true**.	Configuring Query Hints for Optimizing SQL Server Performance with Business Central
DisableSmartSql	If the performance of loading a page that contains FlowFields in Business Central is bad, you might want to try isolating and testing FlowField queries separately.	Troubleshooting: Long Running SQL Queries Involving FlowFields by Disabling SmartSQL
EnableCloudReplicationMaintenance	Specifies whether to keep the cloud replication status in the database up to date. When enabled, synchronize operations on tenants and extensions will update records in the **Intelligent Cloud Status** table, and set change tracking on tables that are configured to replicate data. Enabling this setting isn't required for migrating most on-premises solutions to the cloud, and you'll improve synchronization and upgrade performance by disabling it.	Configuring Business Central Server - EnableCloudReplicationMaintenance
EnableProfileCacheSynchronization	Specifies whether profile cache synchronization across multiple server instances is enabled. However, enabling this setting may lower the tenant performance.	Configuring Business Central Server - EnableProfileCacheSynchronization
SqlBulkImportBatchSize	Specifies how many SQL memory chunks that a data import must be distributed across. Lowering the value increases the number of network transfers and decreases performance. But it also lowers the amount of memory that the server instance consumes.	Database Settings

SERVER SETTING	DESCRIPTION	READ MORE
SessionEventTableRetainInterval NonInteractiveSessionsLogRetainInterval SessionEventTablePurgeLookupPeriod	The system table **2000000111 Session Event** stores information about sessions between Business Central Server instances and clients. Entries are recorded for various events, like when a user signs in or out of the client, or when a web service request starts or stops. It's a good idea to limit the number of entries in this table because performance can be adversely affected as the table size grows. Under normal conditions, you shouldn't experience any problems. But there might be situations or periods, typically dealing with web service calls, during which a high number of session events occur. These three settings enable you to control the table size by specifying how long to keep entries in the table before they are automatically deleted.	Configuring Business Central Server - Session Event Table

Web service limits

You can adjust server instance settings related to web service calls to implement resource governance (and avoid resource starvation on the server instances) here:

- SOAP Services Settings
- OData Services Settings

Task Scheduler

You can adjust server instance settings related to the task scheduler to implement resource governance (and avoid resource starvation on the server instances) here:

- Maximum Concurrent Running Tasks

Database (SQL Server or Azure SQL database)

First of all, make sure that you avoid common pitfalls in your SQL Server setup:

- Installation Considerations for Microsoft SQL Server and Business Central
- Setting SQL Compatibility Level to Optimize Database Performance

Tune data access

These articles describe how you can use SQL Server table partitioning and data compression to get faster data retrieval:

- Using SQL Server table partitioning
- Using SQL Server data compression

SQL Server vs. Azure SQL database

Read more about the difference between SQL Server and Azure SQL database performance:

- Differences between Azure SQL database and SQL Server

Using Read-Scale Out

If you run the Business Central database in a High Availability architecture, you can use the built-in **Read Scale-Out** feature in Azure SQL Database or SQL Server to load-balance read-only workloads. **Read Scale-Out** uses

read-only replicas instead of sharing the primary database. This way, read-only workloads (reports, queries, and API pages) will be isolated from the main read-write workload (codeunits). They won't affect the performance of business processes.

For more information, see Using Read Scale-Out for Better Performance.

Troubleshooting database performance issues

These articles can be useful when troubleshooting database performance issues:

- Troubleshooting: Long Running SQL Queries Involving FlowFields by Disabling SmartSQL
- Configuring Query Hints for Optimizing SQL Server Performance with Business Central
- Troubleshooting: Using Query Store to Monitor Query Performance in Business Central

Performance of BACPAC generation

The sqlpackage is the command-line tool used to generate BACPAC/DACPAC files. The February 2019 update of sqlpackage solved a significant schema-compare performance issue that occurred when generating scripts. Make sure you use **version 18.1 or later** if you experience issues in BACPAC generation performance.

You can also limit the amount of SQL schema to restore from a BACPAC. On tables used only with temporary record variables and pages, set the TableType property to **Temporary**.

Performance impact on setting up CDC on SQL Server

There's a performance impact if you set up CDC on the database. SQL Server will be slower (depending on the retention period used). You'll also need storage for the extra data.

Performance impact of enabling Transparent Data Encryption (TDE)

Enabling Transparent Data Encryption (TDE) has a slight performance degradation on SQL Server because it needs CPU resources for the encryption/decryption of data.

Scaling Business Central

On compute (Business Central Server and Web servers), it's possible to scale horizontally by separating Business Central Server instances and Web servers on different nodes. For more information, see Configuring Business Central Server.

It's also common on larger installations to separate traffic based on client type (direct UI and OData traffic to different Business Central Server instances); possibly cohosting Business Central Server instances and Web servers on the same nodes.

The Business Central Server has a built-in thread dispatcher for AL execution. If you have more cores available, then the dispatcher allows for more parallel execution. But keep in mind that AL execution is single-threaded until the Business Central 2019 release wave 2 (when we introduced async processing). For long running operations, such as heavy reports, using faster CPUs will give better performance.

On the database side, make sure SQL Server has enough resources for sessions (both CPU and memory) and try to optimize the setup of SQL Server to Business Central. For more information, see Installation Considerations for Microsoft SQL Server and Business Central.

Measure and Monitor performance

The following resources describe ways that you measure and monitor performance in your Business Central on-premises installation:

- Monitoring Business Central Server Events
- Tools for Monitoring Performance Counters and Events
- Troubleshooting: Using the Event Viewer to Monitor Long Running SQL Queries in Business Central
- Troubleshooting: Using Query Store to Monitor Query Performance in Business Central

See Also

Performance Overview
Performance Topics For Developers
Performance tips for business users
How Application Configurations Affect Performance
Performance Online
How to Work with a Performance Problem

How to work with a performance problem

2/17/2021 • 2 minutes to read • Edit Online

What do you do if users complain that "it is slow"? In this section, we describe a troubleshooting process that can help to guide you to find the root cause of the problem.

Before getting started on solving a performance tuning problem, it often helps to define and quantify "slow" and also negotiate acceptable values for execution time of "slow" operations with users. This is sometimes called "establishing a baseline."

To define baselines for performance, and to test whether new code or extensions introduce a performance regression, you can use the Performance Toolkit extension. The extension makes it easier to simulate and compare user experiences to your baseline. The following are examples of when the extension can help:

- When you want to ensure that new code does not introduce a regression.
- In a sandbox environment when, for example, the number of users running the same process increases significantly.
- When you want to roll out a new process, or install a new extension.

To solve a performance problem, a common pattern is to do iterations of the following:

1. Measure system performance and collecting performance data
2. Locate a bottleneck
3. Eliminate the bottleneck

and continue until the "slow" operations are comparable to the established baseline.

See Also

Performance Overview
Performance Topics For Developers
Performance tips for business users
How Application Configurations Affect Performance
Performance Online
Performance of On-Premises Installations

Business Central Web Services

2/17/2021 • 5 minutes to read • Edit Online

Business Central supports two types of web services: SOAP and OData. Web services are a lightweight, industry-standard way to make application functionality available to a variety of external systems and users. Developers can create and publish functionality as web services, where they expose pages, codeunits, or queries, and even enhance a page web service by using an extension codeunit. When Business Central objects are published as web services, they are immediately available on the network.

Business Central web services are stateless and do not preserve the values of global variables or single-instance codeunits between calls.

Comparing SOAP and OData Web Services

Developers planning to create Microsoft Dynamics NAV web services may need to decide which type of web service is better suited to their needs. The following table shows the types of web service applications that you can create for the web service protocols.

OBJECT	SOAP WEB SERVICES	ODATA WEB SERVICES
Pages	Yes: Create, Read, Update, and Delete operations (CRUD)	Yes: Create, Read, Update, and Delete operations (CRUD)
Codeunits	Yes	Yes (through OData unbound actions)
Queries	No	Yes: Read-only

Business Central supports OData web services in addition to the SOAP web services that have been available since Microsoft Dynamics NAV 2009.

SOAP Web Services

SOAP web services allow full flexibility for building operation-centric services. They provide industry standard interoperability. Windows Communication Framework (WCF) has supported SOAP services since its initial release in .NET Framework 3.0, and later releases of the .NET Framework have added additional support and default bindings to make it easier to build SOAP services using WCF.

The most common type of messaging pattern in SOAP is the Remote Procedure Call (RPC), where one network node (the client) sends a request message to another node (the server), and the server sends a response message to the client.

OData Web Services

The OData standard is well suited for web service applications that require a uniform, flexible, general purpose interface for exposing create retrieve update delete (CRUD) operations on a tabular data model to clients. OData is less suited for applications that are primarily RPC-oriented or in which data operations are constrained to certain prescribed patterns. OData supports Representational State Transfer (REST)-based data services, which enable resources, identified using Uniform Resource Identifiers (URIs), and defined in an abstract data model (EDM), to be published and edited by web clients within corporate networks and across the Internet using simple Hypertext Transfer Protocol (HTTP) messages. OData services are lightweight, with functionality often referenced directly in the URI.

Whereas SOAP web services expose a WSDL document, OData web services expose an EDMX document

containing metadata for all published web services.

OData is supported in PowerPivot, a data-analysis add-in to Microsoft Excel that provides enhanced Business Intelligence capabilities. PowerPivot supports sharing and collaboration on user-generated business intelligence solutions in a Microsoft SharePoint Server environment. For more information about PowerPivot, see https://www.powerpivot.com/.

The extensions to the Atom Publishing Protocol defined in the AtomPub extensions to the OData protocol documentation (which you can download here) describe how REST-based data services can enable resources, identified using URIs and defined in an abstract data model (EDM), to be published and edited by web clients within corporate networks and across the Internet using simple HTTP messages.

In addition to the AtomPub format, the OData implementation in Business Central also supports the JSON format, a somewhat less verbose format that may perform better in low-bandwidth environments.

Page Web Services

When you expose a page as an OData web service, you can query that data to return a service metadata (EDMX) document, an AtomPub document, or a JavaScript Object Notation (JSON) document. You can also write back to the database if the exposed page is writable. For more information, see OData Web Services.

When you expose a page as a SOAP web service, you expose a default set of operations that you can use to manage common operations such as Create, Read, Update, and Delete. Page-based web services offer built-in optimistic concurrency management. Each operation call in a page-based web service is managed as a single transaction.

For SOAP services, you can also use extension codeunits to extend the default set of operations that are available on a page. Adding an extension codeunit to a page is useful if you want to perform operations other than the standard Create, Read, Update, and Delete operations. The benefit of adding an extension codeunit to a page is that you can make the web service complete by adding operations that are logical to that service. Those operations can use the same object identification principle as the basic page operations.

Codeunit Web Services

For SOAP services only, codeunit web services provide you with the most control and flexibility. When a codeunit is exposed as a web service, all functions defined in the codeunit are exposed as operations.

Query Web Services

When you expose a Business Central query as an OData web service, you can query that data to return a service metadata (EDMX) document or an AtomPub document. For more information about how to create and use Business Central queries, see Query Object.

Web Services and Regional Settings

Data is formatted according to the value of the **Services Language** setting for the relevant Business Central Server instance. The default value is **en-us**. This means that Business Central Server interprets all incoming data as the specified culture, such as dates and amounts.

If you know that the **Services Language** setting is always en-us, for example, your code can be based on that assumption. In a multilanguage environment, you will see more predictable transformations of data if data that is transmitted through web services is in a consistent culture.

Similarly, you can use the **ServicesOptionFormat** setting to specify how Business Central Server must understand option values. If you set the **ServicesOptionFormat** setting to *OptionString*, Business Central Server understand option values as the *name* of the option value, which is always en-us. If you set the setting to *OptionCaption*, web service data will be interpreted in the language specified by the **Services Language** setting.

Web Services in Multitenant Deployments

If your Business Central solution is used in a multitenant deployment architecture, you must make sure that any code that generates or consumes a web service specifies the relevant tenant. Web services are set up in the application, but typically you want to consume company-specific and tenant-specific data.

If you use the GETURL method, the generated URL will automatically apply to the user's tenant ID. For more information, see GETURL Method.

The URL for accessing a web service in a multitenant deployment must specify the tenant ID in one of two ways: As a query parameter, or as a host name. If you use host names for tenants, the host name must be specified as an alternative ID.

For example, the following URL consumes the **Customer** ODATA web service for a specific tenant:

```
https://localhost:7048/BC130/OData/Company('CRONUS-International-Ltd.')/Customer?Tenant=Tenant1
```

For more information, see Multitenant Deployment Architecture.

See Also

Publish a Web Service
Web Services Overview
SOAP Web Service URIs
Using SystemService to Find Companies
Basic Page Operations
Web Services Best Practices
Configuring Business Central Server

Terms of Use

2/17/2021 • 2 minutes to read • Edit Online

By accessing or using any API or Web service exposed in Business Central, you are agreeing to the Microsoft APIs Terms of Use.

For more information, see Microsoft APIs Terms of Use.

Publishing a Web Service

2/17/2021 • 3 minutes to read • Edit Online

You can set up a web service in the Business Central Web client or Dynamics NAV Client connected to Business Central. You must then publish the web service so that it is available to service requests over the network. Users can discover web services by pointing a browser at the computer that is running Business Central Server and requesting a list of available services. When you publish a web service, it is immediately available over the network for authenticated users. All authorized users can access metadata for Business Central web services, but only users who have sufficient Business Central permissions can access actual data.

Creating and Publishing a Web Service

The following steps explain how to create and publish a web service.

1. Open the client.
2. Choose the icon, enter **Web Services**, and then choose the related link.
3. In the **Web Services** page, choose **New**.
4. In the **Object Type** column, select **Codeunit**, **Page**, or **Query**.

   > **NOTE**
   > **Codeunit** and **Page** are valid types for SOAP web services. **Page** and **Query** are valid types for OData web services.

5. In the **Object ID** column, select the object ID of the object that you want to expose. For example, to expose the customer card as a web service, enter **21**.

   If the database contains multiple companies, you can choose an object ID that is specific to one of the companies.

6. In the **Service Name** field, assign a name to the web service. For example, if you expose the customer card as a web service, enter **Customers**.

   - **Codeunit** and **Page** are valid types for SOAP web services. **Page** and **Query** are valid types for OData web services.
   - If the database contains multiple companies, you can choose an object ID that is specific to one of the companies.
   - The service name is visible to consumers of your web service and is the basis for identifying and distinguishing web services, so you should make the name meaningful.
   - If you are setting up integration with Microsoft Outlook using codeunit 5313, then you must use **DynamicsNAVsynchOutlook** as the service name.

7. Select the check box in the **Published** column.

   When you publish the web service, in the **OData URL** and **SOAP URL** fields, you can see the URLs that are generated for the web service. You can test the web service immediately by choosing the links in the **OData URL** and **SOAP URL** fields. Optionally, you can copy the value of the field and save it for later use.

After you publish a web service, it is available on the Business Central Server computer that you were connected

to when you published. The web service is available across all Business Central Server instances running on the server computer.

You can verify the availability of that web service by using a browser, or you can choose the link in the **OData URL** and **SOAP URL** fields in the **Web Services** window. The following procedure illustrates how you can verify the availability of the web service for later use.

Verify the availability of a web service

1. In your browser, enter the relevant URL. The following table illustrates the types of URLs that you can enter. For SOAP web services, use the following format for your URI.

WEB SERVICE TYPE	SYNTAX	EXAMPLE
SOAP	https://*Server.SOAPWebServicePort*/*ServerInstance*/WS/*CompanyName*/services/	https://localhost:7047/BC160/WS/CRONUS International Ltd./services/
OData	https://*Server.ODataWebServicePort*/*ServerInstance*/OData/Company('*CompanyName*')	https://localhost:7048/BC160/OData/Company('CRONUS International Ltd.')

   The company name is case-sensitive.

   > **TIP**
   > If you're using the Business Central API, you can see a description of the endpoints at Endpoints for the APIs for Dynamics 365 Business Central On-Premises and Online.

2. Review the information that is displayed in the browser. Verify that you can see the name of the web service that you have created.

When you access a web service, and you want to write data back to Business Central , you must specify the company name. You can specify the company as part of the URI as shown in the examples, or you can specify the company as part of the query parameters. For example, the following URIs point to the same OData web service and are both valid URIs.

```
https://localhost:7048/<serverinstance>/OData/Company('CRONUS International Ltd.')/Customer
```

```
https://localhost:7048/<serverinstance>/OData/Customer?company='CRONUS International Ltd.'
```

Unpublishing a web service

To unpublish a webs service, clear the **Published** check box. This will make the web service inaccessible.

> **IMPORTANT**
> When you unpublish a web service that is marked for **All Tenants**, other web services that use the same object will be automatically unpublished, even though the **Published** check box is still selected for these web services.

See Also

Handling UI Interaction When Working with Web Services

2/17/2021 • 2 minutes to read • Edit Online

Whether you are publishing or consuming web services, exceptions and dialog boxes that may be displayed while code runs must be handled correctly. Exceptions must be handled to prevent the system from ending the web service client execution. You can handle exceptions in the following ways:

- Writing conditional code inside Business Central.
- Writing the code in the web service client application.

The most robust solution is to use both methods.

Publishing Web Services

When publishing a web service, you must make sure that the code that you are publishing does not assume the ability to interact with a user through the UI. You can use the GUIALLOWED Method to suppress the UI. For example, you can use this method to determine whether a codeunit is being called from the client or from a web service client. You must make sure to suppress errors when a codeunit is called from a web service client.

When implementing a conditional code check in Business Central, you should implement the check only around code that could cause an error. You should not encapsulate the whole business logic.

> **NOTE**
> The server returns the following exception when trying to invoke a dialog UI through a web service: **Microsoft.Dynamics.Nav.Types.Exceptions.NavNCLCallbackNotAllowedException: Callback functions are not allowed.**

AL Keywords That Can Cause Faults or Exceptions

Variables of the Dialog Data Type or any of the methods listed as dialog methods can cause callback not allowed exceptions when they are called from a web service application. The Message Method (Dialog) is the only method in this category that does not cause an exception.

Other keywords that you should not use are:

- PAGE.RUN
- PAGE.RUNMODAL
- ACTIVATE
- REPORT.RUN
- REPORT.RUNMODAL
- HYPERLINK
- FILE.UPLOAD
- FILE.DOWNLOAD

You should also avoid operations on client-side Automation and .NET Framework interoperability objects.

Consuming Web Services

You must handle exceptions in client code that calls a Business Central web service. Appropriate exception capturing code should be included around any call to a Business Central web service.

See Also

Web Services Overview
Publish a Web Service

Managing Time Zones with Web Services

2/17/2021 • 2 minutes to read • Edit Online

Business Central Server provides a **Services Default Time Zone** setting for defining the time zone in which web service calls run. This setting affects both SOAP and OData web services, in addition to NAS Services.

Time Zone Configuration

You can configure the Services Default Time Zone using the Server Administration Tool, Business Central Windows PowerShell Cmdlets, or by directly editing CustomSettings.config, the configuration file for the relevant Business Central Server instance. The following table describes the possible values for the **Services Default Time Zone** setting.

VALUE	DESCRIPTION
UTC	Specifies that all business logic for services on the server instance runs in Coordinated Universal Time (UTC). This is the default value. This is how web services business logic was handled in Microsoft Dynamics NAV 2009 SP1 and Microsoft Dynamics NAV 2009.
Server Time Zone	Specifies that services use the time zone of the computer that is running Business Central Server.
ID of any Windows time zone	Specifies that services use a Windows time zone as defined in the system registry under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones. For example, Romance Standard Time is a valid Windows time zone value.

When a web service writes data back to Business Central , dates and times are based on the setting of the Services Default Time Zone setting. However, the web service consumer can override the setting and specify a different time zone.

> **NOTE**
> Business Central Server stores dates and times as UTC. When a web service receives data from Business Central Server, the time zone is UTC even if the Services Default Time Zone setting is set to a different time zone.

For example, if the Services Default Time Zone setting is set to **UTC+3**, the following table describes two scenarios where a web service consumer modifies Business Central data and sends this back to Business Central Server.

WEB SERVICE CHANGES THE DATETIME FIELD TO	BUSINESS CENTRAL SERVER INTERPRETS THE DATETIME VALUE AS	BUSINESS CENTRAL SERVER SAVES THE DATETIME VALUE AS
01/01/2014 17:00 UTC+1	01/01/2014 17:00 UTC+1	01/01/2014 16:00 UTC
01/01/2014 17:00	01/01/2014 17:00 UTC+3	01/01/2014 14:00 UTC

See Also

SessionSettings.TimeZone Method
Server Administration Tool
Business Central Windows PowerShell Cmdlets

Preserving Data When Working with a Statically Generated Proxy

2/17/2021 • 2 minutes to read • Edit Online

You can lose data if you develop a web service client that interacts with a statically generated proxy. Similarly, a client web service cannot detect if a field has been added to or removed from a page since a proxy was last generated. This topic describes an approach for avoiding this pitfall.

Avoiding Data Loss by Building the Proxy with Your Client Application

To avoid data loss due to a statically generated proxy, automatically generate your proxy whenever you build your client.

Assume you have published page 21, Customer Card, as a SOAP web service with **Customer** as the **Service Name** value. Add a service reference to this web service from a C# console application and insert the following code in the `Main` method:

```
static void Main(string[] args)
{
  CustomerService.Customer_Service svc
      = new CustomerService.Customer_Service();
  svc.UseDefaultCredentials = true;
  Customer c = svc.Read("01121212");
  Console.WriteLine(c.Name);
  Console.ReadKey();
}
```

When you run the application, you will see the following output (assuming the CRONUS International Ltd. demonstration database):

```
Spotsmeyer's Furnishings
```

Then go back to page 21 and set the **Name** property for the control that is bound to the **Name** field to **CustomerName**, and save the page.

Finally, switch back to the console application without updating the web reference and run the code. Instead of getting an error message that indicates that the web request does not match the web service description, you do not get an error message, and `Console.WriteLine` shows an empty line.

See Also

Web Services Overview

Web Services Authentication

2/17/2021 • 4 minutes to read • Edit Online

When users send a request for a web service, they're authenticated according to the credential type that is configured for Business Central Server. To access a web service, users must provide valid credentials for the credential type being used. If Business Central is configured for Windows credential type, then users are automatically authenticated against the Windows account that their computer is running under. In this case, they aren't prompted for their credentials. For other credential types, users are prompted to enter a user name and password.

> **IMPORTANT**
> With Business Central online, the use of access keys (Basic Auth) for web service authentication is deprecated. We recommend that you use OAuth2 instead. For more information, see Using OAuth to Authorize Business Central Web Services.

About NavUserPassword and AccessControlService credential types

If your solution uses NavUserPassword or AccessControlService as the credential type, users can access data through SOAP and OData web services by specifying a password or access key. You set up the user accounts in the Business Central client, based on how they'll access Business Central data. For example, if you set up a user account that will allow an external application to read Business Central data through a web service, you can generate a web service access key and specify that key for the relevant user accounts. Then, you add the access key to the configuration of the application that consumes the web service. In contrast, when users access Business Central data through a web service in Microsoft Excel, for example, they specify a password instead of a web service access key.

Business Central also supports OAuth authentication on OData and SOAP endpoints. OAuth is an open standard for authorization that provides client applications with secure delegated access to server resources. OAuth enables you to extend single sign-on with Microsoft 365 to Business Central web services. For more information, see Using OAuth to Authorize Business Central Web Services (OData and SOAP).

> **IMPORTANT**
> If the Business Central Server is configured to use NavUserPassword or AccessControlService authentication, then the username, password, and access key can be exposed if the SOAP or OData data traffic is intercepted and the connection string is decoded. To avoid this condition, configure SOAP and OData web services to use Secure Socket Layer (SSL).

Unicode characters in user name or password

When Business Central data is consumed by a web service, users can't be authenticated if their user name or password contains Unicode characters. This condition is a limitation in the basic authentication mechanism that is defined in the HTTP/1.1 specification.

The same limitation applies to exposing Business Central data in external products such as a browser or a Microsoft .NET Framework assembly.

How to use an Access Key for SOAP and OData Web Service

Authentication

If your solution is configured for NavUserPassword or AccessControlService authentication, then you can configure Business Central user accounts to include an access key that can be used instead of a password to authenticate SOAP and OData web service requests. A web service access key is a random 44 character string that is associated with the user account. Because it can only be used for SOAP and OData web services, it doesn't require the same level of protection as a password.

Generate a Web Service Access Key

Follow these instructions to generate a web service access key for a user. You perform these steps from the user setup in Business Central client.

1. Choose the icon, enter **Users**, choose the related link, and then open the user account that you want to edit.
2. In the **Web Service Access** section, select the **Web Service Access Key** field.
3. In the **Set the Web Service Access Key** window, if you do not want the key to expire, select the **Key Never Expires** check box. If you want the key to expire, set the **Key Expiration Date** field to the date.
4. Choose the **OK** button.

   The access key is automatically generated. If you're signed in as the user that you modified, the key appears in the **Web Service Access Key** field. Otherwise, the key is masked so only asterisks (*) are shown.

Implement the Web Service Access Key

Typically, you would create a user account strictly for web services, and then use the account's credentials, which include the user name and access key, in a web service application. For example, if you develop your own web service application, then you can design your application to programmatically pass the credentials to the web service. Some applications let you provide the connection credentials through a user interface. The steps for implementing the web service access key are done in Business Central client.

1. Create a user specifically for web services.

   For more information, see Manage Users and Permissions.
2. Sign in as the new user, and generate a web service access key in the user account.
3. Use the access key in the web service application.

TO	SEE
Learn how to use code to pass the user name and web access key to a web service	Passing Credentials for Authentication to Web Services

See Also

Web Services Overview
SOAP Web Services
OData Web Services
Authentication and Credential Types

Using Security Certificates with Business Central On-Premises

2/17/2021 • 7 minutes to read • Edit Online

> **NOTE**
> Dynamics NAV Client connected to Business Central is **DISCONTINUED AFTER:** Business Central Spring 2019.

You use certificates to help secure connections over a wide area network (WAN), such as connections from the Business Central Web Server, Dynamics NAV Client connected to Business Central, and web services to the Business Central Server. Implementing security certificates on your deployment environment requires modifications to various components, like the Business Central Server, Business Central Web Server, and clients.

About Security Certificates

A certificate is a file that Business Central Server uses to prove its identity and establish a trusted connection with the client that is trying to connect. Business Central can support the following configurations:

- *Chain trust*, which specifies that each certificate must belong to a hierarchy of certificates that ends in a root authority at the top of the chain.
- *Peer trust*, which specifies that both self-issued certificates and certificates in a trusted chain are accepted.

The implementation in this section describes the chain trust configuration, which is the more secure option.

> **NOTE**
> An instance of Business Central Server that has been configured for secure WAN communication always prompts users for authentication when they start the client, even when the client computer is in the same domain as Business Central Server.

Certificates for Production

In a production environment, you should obtain a certificate from a certification authority or trusted provider. Some large organizations may have their own certification authorities, and other organizations can request a certificate from a third-party organization.

Obtaining Certificates

You implement chain trust by obtaining X.509 service certificates from a trusted provider. These certificates and their root certification authority (CA) certificates must be installed in the certificates store on the computer that is running Business Central Server. The CA certificate must also be installed in the certificate store on computers that are running the Business Central Web Server and Dynamics NAV Client connected to Business Central so that clients can validate the server.

Most enterprises and hosting providers have their own infrastructure for issuing and managing certificates. You can also use these certificate infrastructures. The only requirement is that the service certificates must be set up for key exchange and therefore must contain both private and public keys. Additionally, the service certificates that are installed on Business Central Server instances must have the Service Authentication and Client Authentication certificate purposes enabled.

IMPORTANT

Microsoft recommends against using wildcard SSL certificates in Business Central installations. Wildcard certificates pose security risks because if one server or sub-domain is compromised, all sub-domains may be compromised. Wildcard certificates also introduce a new style of impersonation attack. In this attack, the victim is lured to a fraudulent resource in the certified domain through phishing. Conventional certificates detect this attack, because the user's browser checks that the private key is hosted on a server whose name matches the one displayed in the browser's address window.

Run the Certificates Snap-in for Microsoft Management Console

Some of the following procedures use the Certificates snap-in for Microsoft Management Console (MMC). If you do not already have this snap-in installed, you can add it to the MMC. For information see Add the Certificates Snap-in to an MMC.

Install and Configure the Certificates

You install the security certificates on the computers running Business Central Server, Business Central Web Server, and Dynamics NAV Client connected to Business Central. The root CA certificate and the service certificate are used in the configuration, but client certificates are not.

Install Certificates on components

1. Follow the installation instructions that are available from your certificate provider to install the root CA and service certificates on the following computers:

   - Install the root CA on the computer that is running Business Central Server and all computers that are running Business Central Web Server instances and Dynamics NAV Client connected to Business Central.
   - Install the service certificate on the computer that is running Business Central Server only.

2. Make sure that the **Server Authentication** and **Client Authentication** certificate purposes are enabled for the service certificate.

   A certificate can be enabled for several different purposes. The **Server Authentication** and **Client Authentication** purposes must be enabled. You can enable or disable other purposes to suit your requirements.

   You enable certificate purposes by using the Certificates Snap-in for MMC. For more information, see Modify the Properties of a Certificate.

Grant access to the Business Central Server service account

After you have installed the root CA and the service certificate on the computer running Business Central Server, you must grant access to the service account that is associated with the server so that the service account can access the service certificate's private key.

1. In the left pane of MMC, expand the **Certificates (Local Computer)** node, expand the **Personal** node, and then select the **Certificates** subfolder.
2. In the right pane, right-click the certificate, select **All Tasks**, and then choose **Manage Private Keys**.
3. In the **Permissions** dialog box for the certificate, choose **Add**.
4. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, enter the name of the dedicated domain user account that is associated with Business Central Server, and then choose the **OK** button.
5. In the **Full Control** field, select **Allow**, and then choose the **OK** button.

6. In the right pane, select the certificate.
7. In the **Certificate** dialog box, choose the **Details** tab, and then select the **Thumbprint** field.
8. Copy the value of **Thumbprint** field.

   For example, copy the hexadecimal characters to text editor, such as Notepad. Delete all spaces from the thumbprint string. If the thumbprint is `c0 d0 f2 70 95 b0 3d 43 17 e2 19 84 10 24 32 8c ef 24 87 79`, then change it to `c0d0f27095b03d4317e219841024328cef248779`.

   > **TIP**
   > It is important that the thumbprint does not contain any invisible extra characters; otherwise you will experience problems when using it later. To avoid this, see Certificate thumbprint displayed in MMC certificate snap-in has extra invisible unicode character.

Configure the Business Central Server instance

The Business Central Server instance configuration includes several settings for certificates and enabling remote logins. You can modify a server instance by using Business Central Server Administration tool or Business Central Administration Shell. For details about how to modify a server instance, see Configuring Business Central Server.

1. Run the Business Central Server Administration tool.
2. Under **General**, change the following settings for the Business Central Server instance.

SETTING	NEW VALUE	DESCRIPTION
Credential Type	`NavUserPassword`, `Username`, or `AccessControlService`	The default value is `Windows`. When you change it to `NavUserPassword`, `Username`, or `AccessControlService`, client users who connect to the server are prompted for user name and password credentials.
Certificate Thumbprint	Value of the **Thumbprint** field in the previous procedure.	Remove any leading or trailing spaces in the thumbprint.

3. If you want to use secure web services, then under **SOAP Services** and **OData Services**, select the **Enable SSL** check box.
4. Save and the new values for the server instance.
5. Restart the Business Central Server instance.

   If there is a problem, see Windows Event Viewer.

Configure the Business Central Web Server and Dynamics NAV Client connected to Business Central

The chain trust configuration allows client users to log on to one or more instances of Business Central Server as long as their login credentials have been associated with user accounts in Business Central. The client validates that the server certificate is signed with the root CA.

After you have installed the root CA on the computer running the Business Central Web Server or Dynamics

NAV Client connected to Business Central, you must modify the client configuration file.

Modify the Business Central Web client configuration file

1. On the computer that is installed the Business Central Web Server, open the navsetting.json configuration file in a text editor, such as Notepad.

2. Change the following settings:

KEY	NEW VALUE	DESCRIPTION
ClientServicesCredentialType	`NavUserPassword`, `Username`, or `AccessControlService`	The default value is `Windows`. When you change it to `NavUserPassword`, `Username`, or `AccessControlService`, client users who connect to the server are prompted for user name and password credentials.
DnsIdentity	The subject name of the service certificate	The default value is <identity>. Replace this with the subject name or common name (CN) of the certificate that is used on the computer that is running Business Central Server.

3. Save the navsettings.json configuration file.

Modify the Dynamics NAV Client connected to Business Central configuration file

1. Open the ClientUserSettings.config configuration file.

   The location of this file is *Users\<username>\AppData\RoamingLocal\Microsoft\Dynamics 365 Business Central*.

   By default, this file is hidden. Therefore, you may have to change your folder options in Windows Explorer to view hidden files.

   > **NOTE**
   > If you want to change default Dynamics NAV Client connected to Business Central settings for all future users, edit the default ClientUserSettings.config file — that is, the one in C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160. Be sure that you run your text editor with Administrator privileges when you do so.

2. Modify the following settings.

KEY	NEW VALUE	DESCRIPTION
ClientServicesCredentialType	`NavUserPassword`, `Username`, or `AccessControlService`	The default value is `Windows`. When you change it to `NavUserPassword`, `Username`, or `AccessControlService`, client users are prompted for user name and password credentials.

KEY	NEW VALUE	DESCRIPTION
DnsIdentity	The subject name of the service certificate.	The default value is <identity>. Replace this with the subject name or common name (CN) of the certificate that is used on the computer that is running Business Central Server.

3. Save and close the ClientUserSettings.config file.

When starting the Dynamics NAV Client connected to Business Central, users are prompted for a valid user name and password.

See Also

Authentication and User Credential Types

Web Services Best Practices

2/17/2021 • 2 minutes to read • Edit Online

This article provides recommendations that you can implement to make your web services applications easier to understand and maintain.

RECOMMENDATION	EXAMPLE
Use the HTTPS protocol to send data between Business Central and the web service consumer. The examples in this section use the HTTP protocol to illustrate the setup, but we recommend that your solution uses transport-level security.	In the application that consumes the Business Central web service, require that URIs are accessed by using HTTPS. For example, a more secure URI for the OData web services on your local computer is `https://localhost:7048/BC130/OData/`.
Use singular forms of names. This provides meaningful singular entity names in the generated proxy classes.	When publishing page 21, Customer Card, use **Customer** as the service name instead of **Customers** or **CustomerCard**.
Avoid using spaces and other characters because they are transformed to underscores or other characters that may not be displayed as you want and could lead to ambiguity.	When publishing page 42, Sales Order, remove the space and use **SalesOrder** as the service name.
Use Pascal casing when you combine words. Pascal casing capitalizes the first character of each word, including acronyms and initialisms that are more than two letters long.	Use **SalesOrder** or **ContactPerson** as the service name.

See Also

Web Services Overview

SOAP Web Services

2/17/2021 • 2 minutes to read • Edit Online

SOAP web services enable full flexibility for building operation-centric services. They provide industry-standard interoperability and channel and host pluggability.

You can use SOAP to interact with page or codeunit web services in Business Central .

TO	SEE
Review the different options for creating URIs to interact with SOAP web services.	SOAP Web Service URIs
Review the set of operations that are available when a page is exposed as a web service.	Basic Page Operations
Learn how to write code that provides a list of existing companies in a Business Central database.	Using SystemService to Find Companies
Ensure that field values are updated from web services.	Using Properties to Indicate Field Value Presence

See Also

Basic Page Operations

SOAP Web Service URIs

2/17/2021 • 2 minutes to read • Edit Online

Web service users can discover published web services by pointing a browser or a tool such as the Web Services Discovery Tool at the computer running Business Central Server and getting a list of available services. For SOAP web services, you typically enter a URI in a browser to view a list of available Business Central web services or to view a schema for a particular web service.

URIs for SOAP Web Services

To display all published SOAP web services that are exposed by a Business Central Server instance, use a URI of the following type:

```
https://<Server>:<Port>/<ServerInstance>/WS/<CompanyName>/services
```

The following example displays all published SOAP web services that are exposed for the CRONUS International Ltd. demonstration database.

```
https://localhost:7047/BC130/WS/CRONUS%20International%20Ltd/services
```

To view the schema for a particular service, use a URI of the following type:

```
https://<Server>:<Port>/<ServerInstance>/WS/<CompanyName>/Page/<servicename>
```

The following example displays the schema for the Customer service for the CRONUS International Ltd. demonstration database.

```
https://localhost:7047/BC130/WS/CRONUS%20International%20Ltd/Page/Customer
```

You can also use a URI for a codeunit web service, as shown in the following example:

```
https://localhost:7047/BC130/WS/CRONUS%20International%20Ltd/Codeunit/Letters
```

Basic Page Operations

2/17/2021 • 2 minutes to read • Edit Online

When you publish a page as a SOAP web service, it has a set of default operations that are exposed to consumers of the web service.

These operations match the actions a user can perform by interacting with a page using the RoleTailored client. The same page- and table-based business logic is executed.

The following table lists the operations and provides links to reference pages.

Pages that are backed by virtual tables are not editable through web services. When such pages are published as SOAP web services, the only operations that are available are **Read**, **ReadMultiple**, and **IsUpdated**.

The term *entity* that is used in the operation signatures describes the data type that is used. The actual entity is defined by the page that is exposed because it contains all controls that are defined on the page. The controls are normally bound to the page's source table. The entity also contains a key that is a special string that uniquely identifies the source table record with a timestamp. This key is used as an argument in many page operations.

All basic page operations are atomic, which means that either all relevant records are affected or no records are affected, even if there was a faulty condition in only one record.

OPERATION	DESCRIPTION AND SIGNATURE
Create Operation	Creates a single record. `void Create(ref Entity entity)`
CreateMultiple Operation	Creates a set of records. `void CreateMultiple(ref Entity[] entity)`
Delete Operation	Deletes a single record. `bool Delete(string key)`
Delete_*part* Operation	Deletes a subpage of the current page. `bool Delete_<part>(string key)`
GetRecIdFromKey	Converts a key, which is always part of the page result, to a record ID. `string GetRecIdFromKey(string key)`
IsUpdated Operation	Checks if an object has been updated since the key was obtained. `bool IsUpdated(string key)`
Read Operation	Reads a single record. `Entity Read(string no)`

OPERATION	DESCRIPTION AND SIGNATURE
ReadByRecId Operation	Reads the record that is identified by RecId. You can use GetRecIdFromKey to obtain a record ID. If the record is not found, then the operation returns null. `Entity ReadByRecId(string formattedRecId)`
ReadMultiple Operation	Reads a filtered set of records, paged. `Entity [] ReadMultiple(Entity_Filter[] filterArray, string bookmarkKey, int setSize)`
Update Operation	Updates a single record. `void Update(ref Entity entity)`
UpdateMultiple Operation	Updates a set of records. `void UpdateMultiple(ref Entity[] entity)`

See Also

SOAP Web Services

Create Operation

2/17/2021 • 2 minutes to read • Edit Online

Creates a single record. The supplied record object is overwritten with the version that is created by the page.

Method Signature

```
void Create(ref Entity entity)
```

Parameters

PARAMETER	DESCRIPTION
entity	Type: Entity A variable of a specific object type that represents the page.

Results

RESULT NAME	DESCRIPTION
entity	Type: Entity A variable of a specific object type that represents the page. Contains the latest values that are present on the page after the record has been inserted into the table.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
The [*record name*] already exists. Identification fields and values: [*field*]=[*value*]	Indicates that the record insertion would violate key constraints.

Other faults are possible if they are generated by the AL code.

Usage Example

```
Customer cust = new Customer();
cust.Name = "Customer Name";
service.Create(ref cust);
```

See Also

Basic Page Operations

CreateMultiple Operation

2/17/2021 • 2 minutes to read • Edit Online

Creates a set of records. The supplied record object is overwritten with the version that is created by the page.

Method Signature

```
void CreateMultiple(ref Entity[] entity)
```

Parameters

PARAMETER	DESCRIPTION
entity[]	Type: An array of Entities An array of a specific object type that represents the page.

Results

RESULT NAME	DESCRIPTION
entity[]	Type: An array of Entities An array of a specific object type that represents the page. Contains the latest values that are present on the page after the records have been inserted into the table.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
The [*record name*] already exists. Identification fields and values: [*field*]=[*value*]	Indicates that the insertion of at least one of the records would violate key constraints.

Other faults are possible if they are generated by the AL code.

The CreateMultiple operation is executed as a single transaction unless the AL code explicitly commits the transaction. Either all or none of the records are inserted unless the application code does not explicitly call COMMIT.

Usage Example

```
Customer[] custArray = new Customer[3];
for (int i = 0; i < custArray.Length; i++)
{
  custArray[i] = new Customer();
  custArray[i].Name = "Customer Name " + i.ToString();
}
service.CreateMultiple(ref custArray);
```

See Also

Basic Page Operations

Delete Operation

2/17/2021 • 2 minutes to read • Edit Online

Deletes a single record.

Executing the Delete operation in a web service first executes the OnDeleteRecord Trigger on the designated page. If application code in the OnDeleteRecord trigger for the page returns **true**, then the OnDelete Trigger for the corresponding table is executed. If no fault occurs, then the record is deleted from the database.

If the application code in the trigger returns **false**, then the OnDelete trigger is not executed. This does not necessarily mean that the record has not been deleted because it may have been deleted explicitly by the application code for the page's OnDeleteRecord trigger.

The return value from the Delete operation is the return value from the page's OnDeleteRecord trigger.

If Delete returns **true**, then the record has been deleted. If Delete has thrown a fault, then the record has not been deleted. But when Delete returns **false**, then this indicates that there was application logic involved, and the record may or may not have been deleted.

Method Signature

```
bool Delete(string key)
```

Parameters

PARAMETER	DESCRIPTION
key	Type: String The bookmark of the record, including both primary key and concurrency information.

Results

RESULT NAME	DESCRIPTION
Delete_Result	Type: Boolean The value that is returned by the OnDeleteRecord page trigger.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
Other user has modified [*record name*].	Indicates that another user or process has modified the record after it has been retrieved for this delete operation.
[*record name*] [*field*] [*value*] does not exist.	Indicates that the record has been deleted by another user or process after it has been retrieved for this delete operation.

SOAP FAULT MESSAGE	DESCRIPTION

Other faults are possible if they are generated by the AL code.

Usage Example

```
      using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication
{
    // Imports newly generated web service proxy.
    using WebService;

    class Program
    {
        static void Main(string[] args)
        {
            // Creates instance of service and sets credentials.
            Customer_Service service = new Customer_Service();
            service.UseDefaultCredentials = true;
            Customer cust = new Customer();
            cust.Name = "Customer Name";
            service.Create(ref cust);
            cust = service.Read(cust.No);
            service.Delete(cust.Key);
        }
    }
}
```

See Also

Basic Page Operations

Delete_<part> Operation

2/17/2021 • 2 minutes to read • Edit Online

Deletes records on a subpage of the current page.

This operation is exposed only by pages that have subpages, which are pages that have parts of type Page. For example, the Sales Order page has a part with the name SalesLines, which has PartType equal to Page and PagePartID equal to "Sales Order Subform." The name of the operation that is exposed by the Sales Order page is Delete_SalesLines.

When you call this operation, you delete a record of the subpage. When you call Read on a page that has a subpage, you get the records of the top-level page and the corresponding records of the subpage. To modify them or add to them, you must modify the whole top-level record.

Executing the Delete_<part> operation in a web service first executes the OnDeleteRecord Trigger on the designated record of the subpage. If application code in the OnDeleteRecord trigger for the record of the subpage returns **true**, then the OnDelete Trigger for the corresponding table is executed. If no fault occurs, then the record of the subpage is deleted from the database.

If the application code in the trigger returns **false**, then the OnDelete trigger is not executed. This does not necessarily mean that the record of the subpage has not been deleted because it may have been deleted explicitly by the application code for the page's OnDeleteRecord trigger.

The return value from the Delete_<part> operation is the return value from the page's OnDeleteRecord trigger.

If Delete returns **true**, then the record of the subpage has been deleted. If Delete has thrown a fault, then the record of the subpage has not been deleted. When Delete returns **false**, there was application logic involved, and the record of the subpage may or may not have been deleted.

Method Signature

```
bool Delete_<part>(string key)
```

Parameters

PARAMETER	DESCRIPTION
key	Type: String The bookmark of the record of the subpage, including both primary key and concurrency information.

Results

RESULT NAME	DESCRIPTION
Delete_Result	Type: Boolean The value that is returned by the OnDeleteRecord page trigger.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
Other user has modified [*record name*].	Indicates that another user or process has modified the record of the subpage after it has been retrieved for this delete operation.
[*record name*] [*field*] [*value*] does not exist.	Indicates that the record of the subpage has been deleted by another user or process after it has been retrieved for this delete operation.

Other faults are possible if they are generated by the AL code.

Usage Example

This example presents a console application that is created in Visual Studio after registering and publishing the Sales Order page. After you publish the page as a web service, you must also add a web reference to it. This task is also described in the walkthrough.

```
using Delete_SalesLinesSample.DynamicsNAVService;

namespace Delete_SalesLinesSample
{
    class Program
    {
        static void Main(string[] args)
        {
            DynamicsNAVService.SalesOrder_Binding so = new SalesOrder_Binding();
            so.UseDefaultCredentials = true;

            SalesOrder salesOrder = so.Read("2001");
            Sales_Order_Line[] salesLines = salesOrder.SalesLines;

            if (salesLines.Length > 0)
            {
                string key = salesLines[0].Key;
                so.Delete_SalesLines(key);
            }
        }
    }
}
```

See Also

Basic Page Operations

GetRecIdFromKey

2/17/2021 • 2 minutes to read • Edit Online

Converts a key to a record ID. The key is always part of the page result.

Method Signature

```
string GetRecIdFromKey(string key)
```

Parameters

PARAMETER	DESCRIPTION
key	Type: String The bookmark of the record that includes both primary key and concurrency information.

Results

RESULT NAME	DESCRIPTION
String	Type: String The record ID that was obtained from the key.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
[*record name*] [*field*] [*value*] does not exist.	Indicates that the record has been deleted by another user or process after it has been retrieved for this operation.

Usage Example

```
Customer_Service service = new Customer_Service();
Customer cust = new Customer();
service.UseDefaultCredentials = true;
string id = service.GetRecIdFromKey(cust.No);
cust = service.ReadByRecId(id.ToUpper());
```

See Also

Basic Page Operations

IsUpdated Operation

2/17/2021 • 2 minutes to read • Edit Online

Checks if an object has been updated since the key was obtained. This operation returns **true** if the object has been updated by any user; otherwise, **false**. Concurrency management prevents a record being changed if it has been subsequently updated. This check proactively prevents that failure.

Method Signature

```
bool IsUpdated(string key)
```

Parameters

PARAMETER	DESCRIPTION
key	Type: String The bookmark of the record, including both primary key and concurrency information.

Results

RESULT NAME	DESCRIPTION
IsUpdated_Result	Type: Boolean Returns **true** if and only if another user has modified the record.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
[*record name*] [*field*] [*value*] does not exist.	Indicates that the record has been deleted by another user or process after it has been retrieved for this operation.

Usage Example

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication
{
    // Imports newly generated web service proxy.
    using WebService;

    class Program
    {
       static void Main(string[] args)
           {
              // Creates instance of service and sets credentials.
              Customer_Service service = new Customer_Service();
              service.UseDefaultCredentials = true;

              Customer cust = new Customer();
              cust.Name = "Customer Name";
              service.Create(ref cust);
              cust = service.Read(cust.No);
              if (!service.IsUpdated(cust.Key))
           {
              // Add code here to modify record.
           }
        }
    }
}
```

See Also

Basic Page Operations

Read Operation

2/17/2021 • 2 minutes to read • Edit Online

Reads a single record.

> **NOTE**
> The signature of an operation changes if the page contains one or more unbound fields in the root content area. If there are unbound fields, then the names from the `SourceExpr` property for the unbound fields are added as the first parameters to the operation. They are added according to the tab order of the page. For an example, see the Item Journal.

Method Signature

```
Entity Read(string no)
```

Parameters

PARAMETER	DESCRIPTION
no	Type: String The primary key of the requested record. This argument is a filter on the primary key and returns the first record that matches the filter. When special characters, such as an apostrophe, equal sign, or ampersand, are part of the key, you must enclose the argument with quotation marks. For example, when you write C# code, you should use standard quotation marks for strings and also use additional single quotation marks around a value, such as `"'A&B'"`.

Results

RESULT NAME	DESCRIPTION
entity	Type: Entity A variable of a specific object type that represents the page. Contains the latest values that are present on the page. If the record with the given primary key does not exist, then no value is returned. In the .NET Framework, a null reference is returned.

Faults

This operation does not return a fault when a matching record is not present. Instead, it returns no value. In C#, you should test whether the result is a null reference. Otherwise, you may get a NullReferenceException

exception.

Faults are possible if they are generated by the AL code.

Usage Example

```
Customer customer = new Customer();
customer.Name = "Customer Name";
service.Create(ref customer);
customer = service.Read(customer.No);
```

See Also

Basic Page Operations

ReadByRecId Operation

2/17/2021 • 2 minutes to read • Edit Online

Reads the record that is identified by the record ID.

Method Signature

```
Entity ReadByRecId(string formattedRecId)
```

Parameters

PARAMETER	DESCRIPTION
formattedRecId	Type: String The RecId that is used to read the record.

Results

RESULT NAME	DESCRIPTION
entity	Type: Entity A variable of a specific object type that represents the page. Contains the latest values that are present on the page. If the record is not found, then the operation returns **null**.

Faults

This operation does not return a fault when a matching record is not present. Instead, it returns **null**. In C#, you should test whether the result is a null reference. Otherwise, you may get a NullReferenceException exception.Faults are possible if they are generated by the AL code.

Usage Example

```
public void ReadById(String id)
{
  Customer_Service service = new Customer_Service();
  Customer cust = new Customer();
  service.UseDefaultCredentials = true;
  cust = service.ReadByRecId(id.ToUpper());
}
```

See Also

Basic Page Operations

ReadMultiple Operation

2/17/2021 • 2 minutes to read • Edit Online

Reads a filtered set of records. This operation returns an array of entities. The ReadMultiple operation allows the consumer of a web service to specify the number of records to be returned at one time. This can reduce load on the server.

> **NOTE**
> Records on a page that were inserted after the page was retrieved are not read. Records on a page may be incorrectly included in the retrieved dataset if they were deleted after the page was retrieved.

Method Signature

```
Entity [] ReadMultiple(Entity_Filter[] filterArray, string bookmarkKey, int setSize)
```

Parameters

PARAMETER	DESCRIPTION
filterArray	Type: Entity_Filter[] An array of record filters.
bookmarkKey	Type: String The last record bookmark of the page that was previously read. To return the first page of results, set *bookmarkKey* to NULL.
setSize	Type: Integer The size of the set to be returned. To return the complete set of results, set *setSize* to zero. To reverse the order of the results, set *setSize* to negative.

Results

RESULT NAME	DESCRIPTION

RESULT NAME	DESCRIPTION
entity[]	Type: An array of Entities An array of a specific object type that represents the page. Contains the latest values that are present on the page. The server will return at most *setSize* records. If all records have been already returned, then subsequent calls will return no records (a 0-element array in C#). You should keep calling the ReadMultiple method until no records are returned.
entity.Key	Type: String The key of the last record read. In C#, you can access it with `Entity[Entity.Length-1].Key`. Pass this as *bookmarkKey* for the next ReadMultiple call.

Faults

This operation does not throw faults when no matching records are present. Instead, it returns an empty record list.

Faults are possible if they are generated by the AL code.

Remarks

Retrieving Last Page

To retrieve the last record of a page, set *setSize* to -1. Using negative numbers in *setSize* sorts the records in descending order.

Entity_Filter

You use the *Entity_Filter* parameter with the ReadMultiple operation. It describes a filter that can be applied on a specific field in a record. The *Entity_Filter* parameter contains two members: Field and Criteria.

- Field contains the name of the field that the filter is applied to. This name comes from the Entity_Fields enum.
- Criteria is of type string and can contain any valid Business Central style filter that is specified in a standard Business Central filter format.

Usage Examples

The following example returns the first 100 customer names that start with an S.

```
List<Customer_Filter> filterArray = new List<Customer_Filter>();
Customer_Filter nameFilter = new Customer_Filter();
nameFilter.Field = Customer_Fields.Name;
nameFilter.Criteria = "S*";
filterArray.Add(nameFilter);
Customer[] custList = service.ReadMultiple(filterArray.ToArray(), null, 100);
```

This example uses the *bookmarkKey* argument to read customer records in batches of 10:

```
Customer_Service service = new Customer_Service();
service.UseDefaultCredentials = true;

const int fetchSize = 10;
string bookmarkKey = null;
List<Customer> customerList = new List<Customer>();

// Reads customer data in pages of 10.
Customer[] results = service.ReadMultiple(new Customer_Filter[] { }, bookmarkKey, fetchSize);
while (results.Length > 0)
{
    bookmarkKey = results.Last().Key;
    customerList.AddRange(results);
    results = service.ReadMultiple(new Customer_Filter[] { }, bookmarkKey, fetchSize);
}

// Prints the collected data.
foreach (Customer customer in customerList)
{
    Console.WriteLine(customer.Name);
}
```

See Also

Basic Page Operations

Update Operation

2/17/2021 • 2 minutes to read • Edit Online

Updates a single record. The updated record is passed as a reference and is updated with the latest version.

Method Signature

```
void Update(ref Entity entity)
```

Parameters

PARAMETER	DESCRIPTION
entity	Type: Entity A variable of a specific object type that represents the page.

Results

RESULT NAME	DESCRIPTION
entity	Type: Entity A variable of a specific object type that represents the page. Contains the latest values that are present on the page after the record has been updated by the business logic or another concurrent process.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
Other user has modified [*record name*].	Indicates that another user or process has modified the record after it has been retrieved for this update operation.
[*record name*] [*field*] [*value*] does not exist.	Indicates that the record has been deleted by another user or process after it has been retrieved for this update operation.
The [*record name*] already exists. Identification fields and values: [*field*]=[*value*]	Indicates that the renaming of the record would violate key constraints.

Other faults are possible if they are generated by the AL code.

Usage Example

```
Customer cust = new Customer();
cust.Name = "Customer Name ";
service.Create(ref cust);
cust.Name = cust.Name + "Updated";
service.Update(ref cust);
```

See Also

Basic Page Operations

UpdateMultiple Operation

2/17/2021 • 2 minutes to read • Edit Online

Updates a set of records. The updated array of records is passed as a reference and is updated with the latest version.

> **NOTE**
> The UpdateMultiple operation attempts to update one field at a time. This can cause errors if the updated value in one field prevents another field's value from being valid. You can resolve this by deleting and inserting lines instead of updating fields.

Method Signature

```
void UpdateMultiple(ref Entity[] entity)
```

Parameters

PARAMETER	DESCRIPTION
entity[]	Type: An array of Entities. An array of a specific object type that represents the page.

Results

RESULT NAME	DESCRIPTION
entity[]	Type: An array of Entities An array of a specific object type that represents the page. Contains the latest values that are present on the page after the records have been updated by the business logic or another concurrent process.

Faults

SOAP FAULT MESSAGE	DESCRIPTION
Other user has modified [*record name*].	Indicates that another user or process has modified the record after it has been retrieved for this update operation.
[*record name*] [*field*] [*value*] does not exist.	Indicates that the record has been deleted by another user or process after it has been retrieved for this update operation.
The [*record name*] already exists. Identification fields and values: [*field*]=[*value*]	Indicates that the renaming of the record would violate key constraints.

Other faults are possible if they are generated by the AL code.

The UpdateMultiple operation is executed as a single transaction unless the AL code explicitly commits the transaction. Either all or none of the records are updated provided that the application code does not explicitly call COMMIT.

Usage Example

The following example adds the string "Updated" to the name of the first 100 customer names that start with S.

```
List<Customer_Filter> filterArray = new List<Customer_Filter>();
Customer_Filter nameFilter = new Customer_Filter();
nameFilter.Field = Customer_Fields.Name;
nameFilter.Criteria = "S*";
filterArray.Add(nameFilter);
Customer[] custList = service.ReadMultiple(filterArray.ToArray(), null, 100);
for (int i = 0; i < custList.Length; i++)
{
  custList [i].Name = custList [i].Name + " Updated";
}
service.Update(ref custList)
```

See Also

Basic Page Operations

Using SystemService to Find Companies

2/17/2021 • 2 minutes to read • Edit Online

You can use the SystemService service in a SOAP web service application to retrieve a list of companies available in a specific database. A company name is typically part of the URI when you access a Business Central web service, and the system service lets you retrieve names of available companies. If you do not specify a company name in a URI, then the default company is used.

In this procedure, you use the SystemService service to retrieve and print a list of companies in Visual Studio.

Use the SystemService service to find companies

1. In Visual Studio, on the **File** menu, point to **New**, and then choose **Project**.
2. Expand the **Visual C#** node, select **Windows**, and then select **Console Application**.

   Caution

   Do not double-click or otherwise dismiss the **New Project** dialog box.
3. Enter **FindingCcompanies** as the **Name** for the application, and then choose **OK**.
4. In Solution Explorer, right-click the **References** node in the project, and then choose **Add Service Reference**.
5. In the **Add Service Reference** dialog box, choose the **Advanced** button, choose the **Add Web Reference** button, type or paste the URL that you used when checking the WSDL, such as `https://localhost:7047/BC130/WS/Services`, and then choose the green arrow to visit the URL.
6. When the **SystemService** service is displayed, choose **View Service**, wait for the service to be displayed, and then choose **Add Reference**. Rename the Web reference name from **localhost** to **NavSOAPService**.
7. On the **Program.cs** tab, replace the stub code with the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FindingCompanies
{
    using System;
    using BCSOAPService;

    public class Program
    {
        static void Main(string[] args)
        {
            // Creates instance of service and set credentials.
            var systemService = new SystemService
            {
                UseDefaultCredentials = true
            };

            // Loads all companies into an array.
            var companies = systemService.Companies();

            // Runs through and print all companies.
            // Also prints company name in encoded form.
            foreach (string company in companies)
            {
                Console.WriteLine(company);
                Console.WriteLine((Uri.EscapeDataString(company)));
            }
            Console.ReadLine();
        }
    }
}
```

8. Save (press Ctrl+F6) and compile (press F6) the **FindingCompanies** application.
9. Press F5 to run the application in debug mode.

   A list of all companies in the current database is presented in a command session.

See Also

SOAP Web Services

Using Properties Indicate the Presence of a Value in a Field

2/17/2021 • 2 minutes to read • Edit Online

When you set a field to a value in the code for a web service client and then send the value back to the web service, such as when you use the Create Operation or the Update Operation, the field value is not actually updated to anything other than its default value. This issue can occur if you are using proxies that are generated by Visual Studio or the tools in the .NET Framework SDK. This topic describes the issue and shows how to avoid the problem.

Using Properties to Indicate the Presence of Values

The problem occurs because of the way proxies are auto-generated and the way the .NET Framework handles values in XML documents that are exchanged between the web service and the client. The issue is not specific to Business Central.

For example, the following code shows the schema for the two fields on this Customer Card web service:

```
<xsd:element minOccurs="0" maxOccurs="1" name="Credit_Limit_LCY" type="xsd:decimal" />
<xsd:element minOccurs="0" maxOccurs="1" name="Salesperson_Code" type="xsd:string" />
```

Both fields are optional, which means that they do not have to be present in the XML document. One field is declared as a Decimal data type in Business Central. The other field is a string because it is declared as a Text data type in Business Central.

If both values are present, then the XML document should contain the following elements:

```
<Credit_Limit_LCY>1000</Credit_Limit_LCY>
<Salesperson_Code>JR</Salesperson_Code>
```

But if there is no information about the credit limit, then the document should contain the following element:

```
<Salesperson_Code>JR</Salesperson_Code>
```

If the credit limit is zero, then the document should contain the following line:

```
<Credit_Limit_LCY>0</Credit_Limit_LCY>
```

For the decimal type and all .NET Framework value types, you can handle this state in the proxy objects by using a **Boolean *Specified** property:

```
if (salesOrder.Credit_Limit_LCYSpecified)
    there is a value present in salesOrder.Credit_Limit_LCY
else
    there is no useful value in salesOrder.Credit_Limit_LCY
```

If you assign a non-default value in the value type, then you should confirm this by setting the accompanying **Boolean *Specified** property to `true`:

```
salesOrder.Credit_Limit_LCY = 1000;
salesOrder.Credit_Limit_LCYSpecified = true;
```

To specify that there is no meaningful value in the `salesOrder.Credit_Limit_LCY` value type, set the accompanying **Boolean *Specified** property to `false`:

```
salesOrder.Credit_Limit_LCYSpecified = false;
```

The value in the `salesOrder.Credit_Limit_LCY` value type will now be disregarded.

.NET Framework reference types, such as the String class, are handled differently because .NET Framework declarations that are based on those types can be null, which means that they can have an explicit expression of no value present:

```
if (salesOrder.Salesperson_Code != null)
    there is a value present in salesOrder.Salesperson_Code
else
    there is no value in salesOrder.Salesperson_Code
```

If you assign a value, then you implicitly also define it as present:

```
salesOrder.Salesperson_Code = "JR";
```

To specify that there is no useful value in `salesOrder.Salesperson_Code`, you should set it to null:

```
salesOrder.Salesperson_Code = null;
```

Example

```
SalesOrder salesOrder = new SalesOrder();
salesOrder.Order_Date = DateTime.Today;
salesOrder.Order_DateSpecified = true;
salesOrder.Currency_Code = "SEK";

salesOrderService.Create(ref salesOrder);
```

OData Web Services

2/17/2021 • 2 minutes to read • Edit Online

The Open Data Protocol (OData) is a web protocol that is designed for querying tabular data and provides you with an alternative to SOAP-based web services. OData builds on web technologies such as HTTP, the Atom Publishing Protocol (AtomPub), and JavaScript Object Notation (JSON) to provide access to information from different applications, services, and stores. OData uses URIs for resource identification and commits to an HTTP-based, uniform interface for interacting with resources. This commitment to core Web principles allows for OData to enable a new level of data integration and interoperability across a broad range of clients, servers, services, and tools.

You can use OData web services to show Business Central data, and you can update data in a Business Central database using OData web services.

OData can be found in other Microsoft products and technologies, including the following:

- Microsoft Excel implements OData for its PowerPivot add-in.
- Microsoft SharePoint can expose its list-oriented data with OData.
- Microsoft Azure Table Services are based on OData.

The topics in this section describe the key concepts and techniques for accessing Business Central data from OData applications that are supported by Business Central .

TO	SEE
Use OData to obtain an AtomPub document.	Using OData to Return or Obtain an AtomPub Document
Use OData to obtain a service metadata (EDMX) document.	Using OData to Return or Obtain a Service Metadata (EDMX) Document
Use OData to obtain a JavaScript Object Notation (JSON) document.	Using OData to Return-Obtain a JSON Document
Use filter expressions in OData URIs.	Using Filter Expressions in OData URIs
Use FlowFilters in OData URIs.	Using FlowFilters in OData URIs
Use server-driven paging in OData URIs.	Server-Driven Paging in OData Web Services
Navigate in an OData web service application by using resource properties.	Using Containments and Associations
Write to the database through an OData web service that exposes a writable page.	Using OData Web Services to Modify Data

Enabling and Configuring OData on the Business Central Server

The Business Central Server instance has several configurations settings that enable and control OData services. For more information, see OData Services Settings.

Known Limitations

Filters

You can specify filters in OData web services in general that are not supported in Business Central , such as using an OR operator to filter on two different fields. In those cases, you will see the following error:

```
An error occurred while processing this request.
 The 'OR' operator is not supported on distinct fields on an OData filter.
```

Lambda operators

Lambda operators are not supported by Business Central OData APIs. If lambda operators are used, the filter expression will be ignored.

UI pages

If you use Web services that are based on UI pages, you must expect the same behavior from the Web service as from the UI page. If you want to have full control and separation of concern it is recommended to use the Business Central APIs instead.

Deep patching

Business Central supports deep insert, but not deep patching. Multiple requests will need to be issued when patching nested entities.

See Also

SOAP Web Services

Using OData V3 to Return-Obtain an AtomPub Document

2/17/2021 • 3 minutes to read • Edit Online

When you register an OData web service, you expose an OData V3 service endpoint that can be accessed from a uniform resource identifier (URI) by using a web browser or any other HTTP client. OData V3 clients can use Atom Publishing Protocol (AtomPub) documents to interact with Business Central data. AtomPub is a simple HTTP-based protocol for creating and updating web resources. It is related to the Atom Syndication Format, which is XML for web feeds. In these procedures, you obtain different kinds of AtomPub documents or feeds from a Business Central OData web service. AtomPub documents and feeds are XML.

> **NOTE**
> This article applies only to OData V3. You can't use the article with OData V4 endpoints.

Obtain an AtomPub Document or Feed

Depending on how you construct your URI, you can return an AtomPub document or an AtomPub feed. A feed is a request for data that can change over time. For example, this can be news content or other kinds of information. In the case of Business Central, the information is database content.

1. Register and publish a page web service by using the Business Central Web client. See Publishing a Web Service.

   The AtomPub documents that are shown in this article are based on the page 21, the **Customer Card** page, with **Customer** as the service name. The concepts and steps are the same for any Business Central Web client page that you register and publish as a web service.

   > **NOTE**
   > You can also register and publish a Business Central query as a web service.

2. Start Windows Internet Explorer. In the **Address** field, enter a URI in this format:

   ```
   https://<Server>:<WebServicePort>/<ServerInstance>/OData
   ```

   If Business Central Server is running on the local computer with the default Business Central Server instance and OData port, then the address is:

   ```
   https://localhost:7048/<server instance>/OData
   ```

   The browser should now show the web service that you have published in the format of an AtomPub document:

3. If you have multiple companies, then you can modify your URI to return a feed that enumerates all available companies:

```
https://localhost:7048/<server instance>/OData/Company
```

> **IMPORTANT**
>
> You must modify your Internet Explorer settings to display the actual XML for a feed instead of the feed content that has changed. Choose **Internet Options**, choose **Content**, choose **Feeds and Web Slices**, and then clear the **Turn on feed reading view** check box. Restart Internet Explorer to enable the new setting.

Obtain a Keyed Service Entry

With a keyed service entry, you specify content from a particular row in a Business Central table. The AtomPub document will contain information that is specific to that row. This procedure assumes that you have registered and published a page web service in the previous procedure.

1. Start Windows Internet Explorer. In the **Address** field, enter a URI in the following format to get the entry for the CRONUS International Ltd. company:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-Ltd.')
```

2. To get the data feed for the Customer table in the CRONUS International Ltd. company database, enter a URI in the following format:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-Ltd.')/Customer
```

3. To additionally constrain data to a specific keyed customer in the Customer table, enter a URI in the following format, using the customer no. for the record you want. The example uses customer no. 01121212:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-
Ltd.')/Customer('01121212')
```

Obtain a Filtered Data Feed

With a filtered data feed, you use special syntax in the URI to define a query on the available data. For details on the specific filters available for Business Central OData web service applications and the syntax for using them, see Using Filter Expressions in OData URIs.

1. Start Windows Internet Explorer. In the **Address** field, enter a URI in the following format to get the entry for the CRONUS International Ltd. company:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-Ltd.')/Customer?
$filter=City-eq-'Birmingham'
```

See Also

OData Web Services

Use OData to Return and Obtain a Service Metadata Document

2/17/2021 • 2 minutes to read • Edit Online

The Entity Data Model (**EDM**) is a specification for defining the data that is used by applications that are built on the Entity Framework. **EDMX** is an XML-based file format that is the packaging format for the service metadata of a data service. When you interact with an OData service that is published from Business Central, you can request EDM-based proxies and then use tools such as LINQ to create data access logic. LINQ is a programming model that developers can use to query data from a variety of data sources, including OData. For more information, see LINQ (Language-Integrated Query)

The Business Central implementation of EDM follows the .NET 4.0 WCF Data Service Framework implementation.

The following guidelines have been implemented for EDM.

- Business Central field names are mapped to EDMX property names by replacing spaces with underscores.
- Primary key fields in tables are automatically defined as properties in the service metadata document even if they are not exposed on a page as controls.

Obtain a service metadata (EDMX) document

1. You can obtain service metadata documents for either page or query web services. This example uses a page web service. Register and publish a page web service by using the Business Central Web client. See Publishing a Web Service..
2. Start Windows Internet Explorer. In the **Address** field, enter a URI in this format:

   ```
   https://<Server>:<WebServicePort>/<ServerInstance>/OData/$metadata
   ```

   If Business Central Server is running on the local computer and is using the default Business Central Server instance and OData port, then the address is:

   ```
   https://localhost:7048/<server instance>/OData/$metadata
   ```

   The browser should now show the complete metadata for the page web service that you have published. The beginning of this document looks like this:

See Also

OData Web Services

Using OData to Return or Obtain a JSON Document

2/17/2021 • 2 minutes to read • Edit Online

You can publish a page as a web service and consume it using JavaScript Object Notation (JSON).

Obtain a document based on JSON

1. You can build applications that consume and display Business Central data using JSON. This example assumes that you have registered and published a page web service in Business Central.
2. Start Windows Internet Explorer. In the **Address** field, enter a URI in this format:

   ```
   https://<Server>:<WebServicePort>/<ServerInstance>/OData/<web service>?$format=json
   ```

   If Business Central Server is running on the local computer and is using the default Business Central Server instance and OData port, and you have published a web service that is based on page 21 that is called **Customer**, then the address is:

   ```
   https://localhost:7048/BC130/OData/Customer?$format=json
   ```

   This generates a text file that contains metadata and data from the web service. You can open the file from the browser, or you can save it to disk.

   > **NOTE**
   > The value of the format attribute must be lowercase: `?$format=json`.

   If you want to consume the web service as JSON-P, you can add the `?$callback=<callback function name>` parameter.

   You can use a similar URI to return the web service as an AtomPub document, in which case the attribute is `?$format=atom`. For more information, see Using OData to Return-Obtain an AtomPub Document.

See Also

Using OData Web Services to Modify Data

Using Filter Expressions in OData URIs

2/17/2021 • 3 minutes to read • Edit Online

You can use filter expressions in OData URIs to limit the results that are returned in an AtomPub document. This topic identifies the filter expressions that you can use, describes the equivalent field or table filter that you can use in AL, and presents examples to show the syntax for using filter expressions in OData web service URIs and applications.

Filter Expressions

To add a filter to an OData URI, add `$filter=` to the end of the name of the published web service. For example, the following URI filters the **City** field in the **Customer** page to return all customers who are located in Miami:

```
https://localhost:7048/BC130/OData/Company('CRONUS International Ltd.')/Customer?$filter=City eq 'Miami'
```

The following table shows the filters that are supported in Business Central OData web services and the equivalent AL filter expressions. All examples are based either on page 21, Customer (published as **Customer**), or on page 20, General Ledger Entry (published as **GLEntry**).

> **NOTE**
> Filters that do not have equivalent AL expressions might take longer to process compared to filters that do have equivalent AL expressions. The reason is that filters that do not have equivalent AL expressions are processed on the Business Central Server tier, while filters that do have equivalent AL expressions are processed on the Business Central database tier.

DEFINITION	EXAMPLE AND EXPLANATION	EQUIVALENT AL EXPRESSION
Select a range of values	`$filter=Entry_No gt 610 and Entry_No lt 615` Query on GLEntry service. Returns entry numbers 611 through 614.	..
And	`$filter=Country_Region_Code eq 'ES' and Payment_Terms_Code eq '14 DAYS'` Query on Customer service. Returns customers in Spain where Payment_Terms_Code=**14 DAYS**.	&

DEFINITION	EXAMPLE AND EXPLANATION	EQUIVALENT AL EXPRESSION
Or	`$filter= Country_Region_Code eq 'ES' or Country_Region_Code eq 'US'` Query on Customer service. Returns customers in Spain and the United States. **Alert:** You can use OR operators to apply different filters on the same field. However, you cannot use OR operators to apply filters on two different fields.	|
Less than	`$filter=Entry_No lt 610` Query on GLEntry service. Returns entry numbers that are less than 610.	<
Greater than	`$filter= Entry_No gt 610` Query on GLEntry service. Returns entry numbers 611 and higher.	>
Greater than or equal to	`$filter=Entry_No ge 610` Query on GLEntry service. Returns entry numbers 610 and higher.	>=
Less than or equal to	`$filter=Entry_No le 610` Query on GLEntry service. Returns entry numbers up to and including 610.	<=
Different from (not equal)	`$filter=VAT_Bus_Posting_Group ne 'EXPORT'` Query on Customer service. Returns all customers with VAT_Bus_Posting_Group not equal to EXPORT.	<>
endswith	`$filter=endswith(VAT_Bus_Posting_Group,'RT')` Query on Customer service. Returns all customers with VAT_Bus_Posting_Group values that end in RT.	
startswith	`$filter=startswith(Name, 'S')` Query on Customer service. Returns all customers names beginning with “S”.	

DEFINITION	EXAMPLE AND EXPLANATION	EQUIVALENT AL EXPRESSION
substringof	`$filter=substringof(Name, 'urn')` Query on Customer service. Returns customer records for customers with names containing the string "urn".	
indexof	`$filter=indexof(Location_Code, 'BLUE') eq 0` Query on Customer service. Returns customer records for customers having a location code beginning with the string BLUE.	
replace	`$filter=replace(City, 'Miami', 'Tampa') eq 'CODERED'`	
substring	`$filter=substring(Location_Code, 5) eq 'RED'` Query on Customer service. Returns true for customers with the string RED in their location code starting as position 5.	
tolower	`$filter=tolower(Location_Code) eq 'code red'`	
toupper	`$filter=toupper(FText) eq '2ND ROW'`	
trim	`$filter=trim(FCode) eq 'CODE RED'`	
concat	`$filter=concat(concat(FText, ', '), FCode) eq '2nd row, CODE RED'`	
round	`$filter=round(FDecimal) eq 1`	
floor	`$filter=floor(FDecimal) eq 0`	
ceiling	`$filter=ceiling(FDecimal) eq 1`	

Referencing Different Data Types in Filter Expressions

You must use the appropriate notation for different data types with filter expressions.

- String values must be delimited by single quotation marks.
- Numeric values require no delimiters.

For more information about data types and other information about conventions and standards for OData URIs, see Atom Publishing Protocol: URI Conventions. Conventions for data types are addressed in section 2.2.2, "Abstract Type System."

See Also

OData Web Services

Microsoft OData Docs - Query options overview

Using FlowFilters in OData URIs

2/17/2021 • 2 minutes to read • Edit Online

You can set FlowFilters on the data that your OData web service extracts from the Business Central database.

FlowFilters are a special kind of filter that you use to set ranges on calculations that are shown in FlowFields. For more information, see FlowFilter Overview. FlowFilters for a page are included in the metadata for that page when it is published as a web service. You can then use FlowFilters as filters in a URI that specifies a query against page data. However, only those FlowFilters that are required to calculate the FlowFields that are exposed on the page as controls are included.

Use FlowFilters to Query Data on the Item Card Page

In this procedure, you create and publish a web service from the **Item Card** page in Business Central and then query the data in that web service by using a FlowFilter.

1. Register and publish a page web service by using the Business Central Web client. See Publishing a Web Service.

   Register and publish page 30, Item Card, and name the service **ItemCard**.

2. Start Windows Internet Explorer, and then in the **Address** field, enter a URI in this format:

   ```
   https://<Server>:<WebServicePort>/<ServerInstance>/OData/$metadata
   ```

   If Business Central Server is running on the local computer and uses the default Business Central Server instance and the default OData port, then the address is:

   ```
   https://localhost:7048/BC130/OData/$metadata
   ```

3. Examine the metadata that is returned by this URI. At the end of the list is a set of parameters that end in the word `Filter`. This is the list of FlowFilters for the page:

   ```
   <Property Type="Edm.String" Name="Location_Filter" Nullable="true"/>
   <Property Type="Edm.String" Name="Drop_Shipment_Filter" Nullable="true"/>
   <Property Type="Edm.String" Name="Variant_Filter" Nullable="true"/>
   <Property Type="Edm.String" Name="Lot_No_Filter" Nullable="true"/>
   <Property Type="Edm.String" Name="Serial_No_Filter" Nullable="true"/>
   <Property Type="Edm.String" Name="Date_Filter" Nullable="true"/>
   ```

   NOTE

   The set of FlowFilters that is listed in the page metadata may not match the set of FlowFilters on the equivalent page in the Business Central Web client. This is because the Business Central Web client shows all FlowFilter fields that are defined on the table on which the page is based. The metadata only shows the FlowFilters that are used to calculate the FlowField controls that are exposed on the page.

4. Create a URI that returns information for a single item card. For example:

```
https://localhost:7048/BC130/OData/Company('CRONUS-International-Ltd.')/ItemCard('1906-S')
```

This is the "ATHENS Mobile Pedestal" item. The value for the *Qty_on_Sales_Order* parameter is 33:

```
<d:Qty_on_Sales_Order m:type="Edm.Decimal">33</d:Qty_on_Sales_Order>
```

5. Apply a FlowFilter to that item and specify **GREEN** as the value for the **Location_Filter**:

```
https://localhost:7048/BC130/OData/Company('CRONUS-International-Ltd.')/ItemCard('1906-S')?
$filter=Location_Filter eq 'GREEN'
```

The item is returned as before, the value of the FlowField that has changed. The value for the *Qty_on_Sales_Order* parameter is now 27:

```
<d:Qty_on_Sales_Order m:type="Edm.Decimal">27</d:Qty_on_Sales_Order>
```

This indicates that there are 27 ATHENS Mobile Pedestals on sales orders designated for the GREEN location.

See Also

OData Web Services

Server-Driven Paging in OData Web Services

2/17/2021 • 2 minutes to read • Edit Online

Server-driven paging ensures that the quantity of data that is returned by an OData URI does not overwhelm Business Central Server or client program that you use to capture data, while optimizing performance.

> **NOTE**
> The term *page* in this topic refers only to a page that contains OData results and is not related to Business Central page objects.

Configuring Server-Driven Paging

You configure server-driven paging with the **Max Page Size** setting in the configuration for the Business Central Server instance that you are using for OData services. To modify the setting, you can use Server Administration Tool or [Business Central Windows PowerShell Cmdlets](/powershell/business-central/overview). For more information about **Max Page Size** and other Business Central Server parameters, see Configuring Business Central Server.

The **Max Page Size** setting specifies the maximum number of entities returned per page of OData results. The default value is 1000. You can consider a page to be a chunk of data. A large data feed is divided into chunks of data. Each chunk contains no more entities than the value of **Max Page Size**. An increase in the value of **Max Page Size** creates fewer chunks (or pages) per request, which in turn, decreases the processing time. However, an increase in the **Max Page Size** will increase the memory consumption on the Business Central Server or client. If the value is too large, it can overload the memory on Business Central Server. For performance reasons, you should try to set the value of the **Max Page Size** as large as possible without overloading Business Central Server. If the computer that is running Business Central Server is returning out of memory exceptions, then you should reduce the value of **Max Page Size** until the errors stop.

When using OData with queries that are set with a top number of rows by either the TopNumberOfRows Property and TopNumberOfRows Method, you should set the **Max Page Size** value greater than the value of the **TopNumberOfRows** property and **TopNumberOfRows** method. For more information, see Using OData with Queries That are Set with a Top Number of Rows.

> **NOTE**
> In the CustomSettings.config file for Business Central Server, the **Max Page Size** setting is called **ODataServicesMaxPageSize**.

See Also

OData Web Services

Using Containments and Associations

2/17/2021 • 3 minutes to read • Edit Online

Containments and associations are relationships between pages in Business Central. OData web services support navigation between pages using containments and associations.

- **Containments**: Some pages in Business Central contain subpages. When you publish such a page, the subpages are automatically available in the web service as containments.
- **Associations**: When a field on a page has a **TableRelation** property, the specified table has a **LookupPageId** property that points to a different page. When you publish a page containing such a field as a web service, you must also publish the page that is pointed to by **LookupPageId** property. You can then link from the first page to the second page in a single URI.

Identifying Containments and Associations in Metadata

OData metadata shows containments and associations with the **NavigationProperty** tag. For example, if you published page 42, Sales Order, and page 22, Customer List, and then obtained the service metadata document for the server instance, you would see this tag in the metadata:

```
<NavigationProperty Name="SalesOrderSalesLines" ToRole="SalesOrderSalesLines" FromRole="SalesOrder"
Relationship="NAV.SalesOrder_SalesOrderSalesLines"/>
```

This tag is followed by metadata that's specific to the **SalesOrderSalesLines** subpage. This page metadata indicates that this **NavigationProperty** tag is a containment:

```
<EntityType Name="SalesOrderSalesLines">
<Key>
     <PropertyRef Name="Document_No"/>
     <PropertyRef Name="Document_Type"/>
     <PropertyRef Name="Line_No"/>
</Key>
     <Property Name="Document_Type" Nullable="false" Type="Edm.String"/>
     <Property Name="Document_No" Nullable="false" Type="Edm.String"/>
     .......many additional properties........
     <Property Name="ShortcutDimCode_x005B_7_x005D_" Nullable="true" Type="Edm.String"/>
     <Property Name="ShortcutDimCode_x005B_8_x005D_" Nullable="true" Type="Edm.String"/>
</EntityType>
```

Elsewhere in the metadata, you would see additional **NavigationProperty** lines and a series of **Association** tags. Two of these tags have a **Multiplicity** parameter with a value of **0..1**:

```
<Association Name="SalesOrder_Sell_to_Customer_No_Link">
     <End Type="NAV.SalesOrder" Multiplicity="*" Role="SalesOrder"/>
     <End Type="NAV.CustomerList" Multiplicity="0..1" Role="Sell_to_Customer_No_Link"/>
</Association>
<Association Name="SalesOrder_Bill_to_Customer_No_Link">
     <End Type="NAV.SalesOrder" Multiplicity="*" Role="SalesOrder"/>
     <End Type="NAV.CustomerList" Multiplicity="0..1" Role="Bill_to_Customer_No_Link"/>
</Association>
```

These tags describe two associations from the Sales Order page to the Customer List page:

- The Sell-to Customer No. field.
- The Bill-to Customer No. field.

Using Containments

When you publish a page that has a subpage, you can identify that subpage in the AtomPub document that is returned for the published page. For example, when you publish page 42, Sales Order, you can access a single record on the page using a URI such as the following:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-
Ltd.')/SalesOrder(Document_Type='Order',No='101005')/
```

The following line in the returned AtomPub document for the record provides link information for a containment:

```
<link rel="https://schemas.microsoft.com/ado/2007/08/dataservices/related/SalesOrderSalesLines"
            type="application/atom+xml;type=feed" title="SalesOrderSalesLines"
            href="SalesOrder(Document_Type='Order',No='101005')/SalesOrderSalesLines" />
```

Notice the `type=feed` in the line above. To access the subpage data feed, use a URI that incorporates the link that was identified in the previous document:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-
Ltd.')/SalesOrder(Document_Type='Order',No='101005')/SalesOrderSalesLines
```

Using Associations

Associations are possible when two published pages are linked. Here is an example:

- Page 42, Sales Order, has its **SourceTable** property set to table 36, Sales Header. The source expression for the **Sell_to_Customer_No** control on page 42 is field 2, Sell-to Customer No., in table 36.
- Field 2, Sell-to Customer No., in table 36 has a **TableRelation** property set to table 18, Customer, field No.
- Table 18, Customer, has a **LookupPageId** property set to page 22, Customer List.

Thus if both page 42, Sales Order, and page 22, Customer List, are published as web services, then an OData URI can link from the **Sell_to_Customer_No** control on page 42 to the related entity on page 22.

Because of this association, you can create OData URIs to access data on the Customer List page as you work with data on the Sales Order page.

If you publish pages 42 and 22 as web services, then you can return an AtomPub document for the Sales Order page. The following URI returns data for a single record on the page, which is order number 101005:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-
Ltd.')/SalesOrder(Document_Type='Order',No='101005')/
```

A set of three tags near the top of the returned document show one containment (**SalesOrderSalesLines**) and two associations (**Sell_to_Customer_No** and **Bill_to_Customer_No**) on the page. Notice the `type=entry` in the following lines:

```
<link rel="https://schemas.microsoft.com/ado/2007/08/dataservices/related/SalesOrderSalesLines"
             type="application/atom+xml;type=feed" title="SalesOrderSalesLines"
             href="SalesOrder(Document_Type='Order',No='101005')/SalesOrderSalesLines" />
<link rel="https://schemas.microsoft.com/ado/2007/08/dataservices/related/Sell_to_Customer_No_Link"
             type="application/atom+xml;type=entry" title="Sell_to_Customer_No_Link"
             href="SalesOrder(Document_Type='Order',No='101005')/Sell_to_Customer_No_Link" />
<link rel="https://schemas.microsoft.com/ado/2007/08/dataservices/related/Bill_to_Customer_No_Link"
              type="application/atom+xml;type=entry" title="Bill_to_Customer_No_Link"
              href="SalesOrder(Document_Type='Order',No='101005')/Bill_to_Customer_No_Link" />
```

This information provides the necessary information to create a URI to access a record on the Customer List page by using an association:

```
https://localhost:7048/<server instance>/OData/Company('CRONUS-International-Ltd.')
             /SalesOrder(Document_Type='Order',No='101005')/Sell_to_Customer_No_Link
```

The following URI returns the same information with direct access to the Customer List page:

```
https://localhost:7048/<server instance>/OData/CustomerList('30000')
```

See Also

OData Web Services

Using OData with Queries That are Set with a Top Number of Rows

2/17/2021 • 2 minutes to read • Edit Online

Business Central queries include the TopNumberOfRows Property and TOPNUMBEROFROWS Method that can be used to specify the maximum number of rows to include in the resulting dataset. The OData configuration includes the **Max Page Size** setting that specifies the maximum number of entities returned per page of OData results. The default value is 1000.

To ensure that the OData results include the correct number of entities when you are using a query that is set with a top number of rows, you should set the **Max Page Size** value greater than the value that is set by the **TopNumberOfRows** property and **TopNumberOfRows** method. Otherwise, the **TopNumberOfRows** property and **TopNumberOfRows** method are ignored and the query dataset will be returned in the OData results.

> **NOTE**
>
> Typically, the **TopNumberOfRows** property or **TopNumberOfRows** method are used to return a relatively small number of entities, such as the top five, ten, or 100 entities. Therefore, in most cases, the value of the **TopNumberOfRows** property and **TopNumberOfRows** method will be less than the **Max Page Size**, so that you will not have to change the **Max Page Size** setting.

For information about how to change the **Max Page Size** setting, see Configuring Business Central Server and Server-Driven Paging in OData Web Services.

See Also

Basic Page Operations

Using OData Web Services to Modify Data

2/17/2021 • 2 minutes to read • Edit Online

You can write to the Business Central database using an OData web service that exposes a writable page. For example, you can expose a page as an OData web service and implement it in a portal that is based on Microsoft SharePoint Online. Users of the portal can then modify the data.

Modifying Data Using OData Web Services

If an editable page is exposed as a web service, the data in the underlying table can be accessed and modified by an OData call. Business Central supports the following OData operations for modifying data.

ODATA CALL	DATA IMPACT	TRIGGERS RUN ON PAGE AND TABLE IN BUSINESS CENTRAL
`POST`	Creates a new entity.	`OnNewRecord` and `OnInsert`
`PATCH`	Modifies the specified existing entity.	`OnModify`
`DELETE`	Deletes the specified existing entity.	`OnDelete`

All calls fail if the user does not have the relevant permissions, and if the relevant property on the page, **InsertAllowed**, **ModifyAllowed**, or **DeleteAllowed**, is set to **No**.

You can use an OData web service in applications where you want users to be able to modify Business Central data the Business Central Web client. For example, you can show fields from the **Customer** table on a mobile device or in a browser so that a user can create, update, or delete customers in the Business Central database.

PATCH operations requires the 'If-Match' header to be set, either with a retrieved ETag or with '*'.

Company-Specific and Tenant-Specific OData Calls

In your implementation of the web service, you can specify which company in the database that a user can write to in the URIs that expose the web services. Similarly, you can specify the specific tenant that the change applies to if the database handles more than one tenant.

If you do not specify a company, Business Central will identify a default company. The default company is found in the following order of sequence:

1. The ServicesDefaultCompany setting in the Tenants.config file.
2. The ServicesDefaultCompany setting in the CustomSettings.config file for Business Central Server.
3. The company in the current tenant when there is only one company.

   If the OData request is for modifying metadata, Business Central will return the first company in the tenant database because metadata applies to all companies in the database.

If no default company can be found based on the criteria, an error message appears.

See Also

OData Web Services
Using OData to Return-Obtain a JSON Document

Creating and Interacting with an OData V4 Bound Action

2/17/2021 • 2 minutes to read • Edit Online

This topic provides an overview of how to expose a procedure as an OData V4 web service action and how to verify that the service is working as expected.

Declaring the OData bound action

The following example shows you how you can declare an OData bound action on a page exposed as a web service. For that, you need to add a procedure to the `SalesInvoiceCopy` page, expose the procedure using the `[ServiceEnabled]` attribute, and use the `WebServiceActionContext` and `WebServiceActionResultCode` AL types to set the result of the function.

> **NOTE**
> Bound actions cannot be added by extending an existing page that has been exposed as a web service.

```
page 50110 SalesInvoiceCopy
{
    ODataKeyFields = "Id";
    SourceTable = "Sales Header";

    layout
    {
        area(Content)
        {
            group(GroupName)
            {
                field(Id; Id)
                {
                    ApplicationArea = All;
                }

                field("No."; "No.")
                {
                    ApplicationArea = All;
                }

                field("Sell-to Customer No."; "Sell-to Customer No.")
                {
                    ApplicationArea = All;
                }
            }
        }
    }

    trigger OnOpenPage()
    begin
        SetRange("Document Type", "Document Type"::Invoice);
    end;

    [ServiceEnabled]
    procedure Copy(var actionContext: WebServiceActionContext)
    var
        FromSalesHeader: Record "Sales Header";
        ToSalesHeader: Record "Sales Header";
        SalesSetup: Record "Sales & Receivables Setup";
        CopyDocMgt: Codeunit "Copy Document Mgt.";
        DocType: Option Quote,"Blanket Order",Order,Invoice,"Return Order","Credit Memo","Posted
Shipment","Posted Invoice","Posted Return Receipt","Posted Credit Memo";
    begin
        SalesSetup.Get;
        CopyDocMgt.SetProperties(true, false, false, false, false, SalesSetup."Exact Cost Reversing
Mandatory", false);

        FromSalesHeader.Get("Document Type", "No.");
        ToSalesHeader."Document Type" := FromSalesHeader."Document Type";
        ToSalesHeader.Insert(true);

        CopyDocMgt.CopySalesDoc(DocType::Invoice, FromSalesHeader."No.", ToSalesHeader);

        actionContext.SetObjectType(ObjectType::Page);
        actionContext.SetObjectId(Page::SalesInvoiceCopy);
        actionContext.AddEntityKey(Rec.FIELDNO(Id), ToSalesHeader.Id);

        // Set the result code to inform the caller that an item was created.
        actionContext.SetResultCode(WebServiceActionResultCode::Created);
    end;
}
```

Registering and publishing the page as a web service

1. Open the Business Central Web Client.
2. In the **Search** box, enter **Web Services**, and choose the related link.
3. In the **Web Services** page, on the **Home** tab, choose **New**.
4. In the **Object Type** column, select **Page**. In the **Object ID** column, enter 43, and in the **Service Name** column, enter `SalesInvoiceCopy`.
5. Select the check box in the **Published** column.
6. When you publish the web service, in the **OData URL** and **SOAP URL** fields, you can see the URLs that are generated for the web service.

Verifying the web service availability

HTTP request

```
POST /ODataV4/Company({companyName})/SalesInvoiceCopy({id})/NAV.Copy
```

Request headers

HEADER	VALUE
Authorization	Bearer {token}. Required.

Example

```
{baseurl}/ODataV4/Company('CRONUS%20USA%2C%20Inc.')/SalesInvoiceCopy('S-ORD101001')/NAV.Copy
```

See Also

AL Development Environment
Getting started with Microsoft .NET Interoperability from AL
Devoloping for Multiple Platform Versions
Exporting Permission Sets
Discover Events Using the Event Recorder

Walkthrough: Creating and Interacting With an OData V4 Bound Action

2/17/2021 • 4 minutes to read • Edit Online

This walkthrough illustrates how you can publish a Business Central function as an OData V4 web service action.

About This Walkthrough

This walkthrough provides an overview of how to expose a function as a web service action and how to verify that the service is working as expected. The walkthrough illustrates the following tasks:

- Publishing a Business Central function as a web service.
- Verifying web service availability from a browser.

Prerequisites

To complete this walkthrough, you will need:

- Microsoft Dynamics NAV 2017 CTP 10 with a developer license.
- CRONUS International Ltd. demonstration database.
- The **Postman** app for testing the web service URI.

Publishing a Function as a Web Service

You publish a function as a Web service action by using the Dynamics NAV Development Environment to create the function and the Business Central Windows client or the Business Central Web client for publishing the objects the function is for. The tutorial provides an example that will return a location header. A location header would be used to later issue a get request for the resulting object. Refer to the **Return a Value** section for an example that will return a value specified in your function.

To create the function

Create a Copy action on the Sales Invoice page.

1. Open the Dynamics NAV Development Environment and then connect to the CRONUS International Ltd. company.

   Object Designer opens automatically in the development environment.

2. Open page 43, **Sales Invoice**.
3. Create a new function on the page named `Copy`.
4. Open the properties for the function and set the properties to the following values.

PROPERTY	VALUE
Local	No
FunctionVisibility	External
ServiceEnabled	Yes

5. Open **Locals** for the function and set the parameters to the following values.

PARAMETER	VALUE
VAR	Yes
Name	ActionContext
DataType	DotNet
SubType	Microsoft.Dynamics.Nav.Runtime.WebServiceActionContext.'Microsoft.Dynamics.Nav.Ncl, Culture=neutral, PublicKeyToken=31bf3856ad364e35'

6. Select the **Variables** tab and add the following variables.

NAME	DATATYPE	SUBTYPE
ToSalesHeader	Record	36
FromSalesHeader	Record	36
SalesSetup	Record	311
ODataActionManagement	Codeunit	6711
CopyDocMgt	Codeunit	6620
DocType	Option	OptionString = Quote,Blanket Order,Order,Invoice,Return Order,Credit Memo,Posted Shipment,Posted Invoice,Posted Return Receipt,Posted Credit Memo

7. Add the code that copies the sales document, for example.

```
SalesSetup.GET;
CopyDocMgt.SetProperties(
TRUE,FALSE,FALSE,FALSE,FALSE,SalesSetup."Exact Cost Reversing Mandatory",FALSE);

FromSalesHeader.GET(Rec."Document Type",Rec."No.");
ToSalesHeader."Document Type" := FromSalesHeader."Document Type";
ToSalesHeader.INSERT(TRUE);
CopyDocMgt.CopySalesDoc(DocType::Invoice,FromSalesHeader."No.",ToSalesHeader);

// Add the necessary keys of the newly created entity using that entity's backing table to identify
the field no. and it's value.
ODataActionManagement.AddKey(Rec.FIELDNO(Id),ToSalesHeader.Id);

// Depending on what the result was of the action
// - Created: SetCreatedPageResponse(ActionContext, PAGE::"Sales Invoice");
// - Updated: SetUpdatedPageResponse(ActionContext,PAGE::"Sales Invoice");
// - Deleted: SetDeleteResponse(ActionContext);
ODataActionManagement.SetCreatedPageResponse(ActionContext,PAGE::"Sales Invoice");
```

8. Save and compile the **SalesInvoice** page.

To register and publish a page as a Web service

1. Open Business Central and connect to the CRONUS International Ltd. company.
2. In the **Search** box, enter **Web services**, and then choose the related link.
3. In the **Web Services** page, on the **Home** tab, choose **New**.
4. In the **Object Type** column, select **Page**. In the **Object ID** column, enter `43`, and in the **Service Name** column, enter `SalesInvoice`.
5. Select the check box in the **Published** column.
6. Choose the **OK** button.

Verifying the Web Service Availability

After publishing a web service, verify that the port that web service applications will use to connect to your web service is open. The default port for OData V4 web services is 7047. You can configure this value by using the Server Administration Tool.

To verify availability of a Microsoft Dynamics NAV Web service action

1. Start **Postman** or another tool that can execute a POST command against the web service URI.
2. In the **Address** field, enter a URI in this format:
   ```
   https://<Server>:
   <WebServicePort>/<ServerInstance>/api/beta/companies(<companyid>)/salesInvoices(<invoiceid>)/Microsoft.NAV.Copy)
   ```
   .
   - `<Server>` is the name of the computer that is running Business Central Server.
   - `<WebServicePort>` is the port that OData V4 is running on. The default port is 7047.
   - `<ServiceInstance>` is the name of the Business Central Server instance for your solution. The default name is DynamicsNAV90.

   Example if the default Business Central Server is running on your local computer.
   ```
   https://localhost:7047/BC130/api/beta/companies(b9248a6e-966d-478c-a25d-
   d91d28610397)/salesInvoices(8cc52602-3aa4-4256-b2c7-fdfef5248cbf)/Microsoft.NAV.Copy)
   ```
3. Postman should now show the web service function that you have published, and perform the action of copying an invoice.

Return a value

1. Open the Dynamics NAV Development Environment and connect to the application database.
2. Open page 43, **Sales Invoice**.
3. Create a new function called **Example**.
4. Open **Properties** for the function and set the properties to the following values.

PROPERTY	VALUE
Local	No
FunctionVisibility	External
ServiceEnabled	Yes

5. Open **Locals** for the function parameters to the following values.
6.

PARAMETER	VALUE	TEST
inParam	Text	test

7. Select the **Return Value** tab and then add the following values.
8.

NAME	RETURNTYPE
outParam	Text

9. Then add the following code for the **Example** function:

```
outParam := inParam + ' Completed';
```

10. You can now issue a post request:
```
https://localhost:7047/Navision_NAV/ODataV4/Company('CRONUS International
Ltd.')/SalesInvoice('Invoice', '1004')/NAV.Example
```
with a JSON body of:
```
{ "inParam": "Hello World" }
```

11. The returned value will be returned in the body of the message.

```
{
    "@odata.context":
"https://farpedro.northamerica.corp.microsoft.com:7047/Navision_NAV/ODataV4/$metadata#Edm.String",
    "value": "Hello World Completed"
}
```

You have now published a Business Central function as an OData V4 web service action and verified that the service works as expected. To read more about web services, see the **See Also** section below.

See Also

Web Services
SOAP Web Services
Publish a Web Service
Securing Web Service Connections Using Certificates

Creating and Interacting with an OData V4 Unbound Action

2/17/2021 • 2 minutes to read • Edit Online

Unbound actions represent reusable operations that you can perform using an OData request. Use unbound actions when there is no particular entity that the action needs to be bound to.

Declaring, registering, and publishing the OData unbound action

To declare an OData unbound action define a codeunit with a procedure with the desired business logic.

The following example illustrates a simple codeunit with three procedures that can be exposed as a web service and called as OData unbound actions.

```
codeunit 50100 "MiscOperations"
{
    procedure Ping(input: Integer): Integer
    begin
        exit(-input);
    end;

    procedure Delay(delayMilliseconds: Integer)
    begin
        Sleep(delayMilliseconds);
    end;

    procedure GetLengthOfStringWithConfirmation(inputJson: Text): Integer
    var
        c: JsonToken;
        input: JsonObject;
    begin
        input.ReadFrom(inputJson);
        if input.Get('confirm', c) and c.AsValue().AsBoolean() = true and input.Get('str', c) then
            exit(StrLen(c.AsValue().AsText()))
        else
            exit(-1);
    end;
}
```

Registering and publishing the codeunit as a web service

Registering and publishing the codeunit is identical to how you work with other web services. For more information, see Publish a Web Service.

Verifying web service availability

HTTP Request

To call specific procedure on a codeunit use the base OData URL for the codeunit and procedure name separated by an underscore.

```
POST /ODataV4/{serviceName}_{procedureName}?company={companyName|companyId} HTTP/1.1
{requestBody}
```

Request Headers

HEADER	VALUE
Authorization	Bearer {token}. Required.

Example Request

```
POST {baseUrl}/ODataV4/MiscOperations_GetLengthOfStringWithConfirmation?company=CRONUS%20USA%20Inc. HTTP/1.1
{
    "inputJson": "{\"str\":\"Hello world!\",\"confirm\":true}"
}
```

Example Response

```
HTTP/1.1 200 OK
{
  "@odata.context": "{baseUrl}/ODataV4/$metadata#Edm.Int32",
  "value": 12
}
```

See Also

AL Development Environment
Creating and Interacting with an OData V4 Bound Action
Getting started with Microsoft .NET Interoperability from AL
Devoloping for Multiple Platform Versions
Exporting Permission Sets
Discover Events Using the Event Recorder

Getting Started Developing Connect Apps for Dynamics 365 Business Central

2/17/2021 • 7 minutes to read • Edit Online

A Connect app establishes a point-to-point connection between Dynamics 365 Business Central and a 3rd party solution or service and is typically created using standard REST API to interchange data. Any coding language capable of calling REST APIs can be used to develop your Connect app. In the following section you can read about how you get started exploring the available APIs for Dynamics 365 Business Central.

To explore and develop against APIs in Dynamics 365 Business Central, you must first sign up for a trial tenant and then you have to connect and authenticate. To do that, follow the steps below.

1. Sign up for Dynamics 365 Business Central.
   When you have your tenant, you can sign into the UI to play with the product, as well as explore the APIs
2. There are two different ways to connect to and authenticate against the APIs.
   - Use Azure Active Directory (AAD) based authentication against the common API endpoint: `https://api.businesscentral.dynamics.com/v2.0/<environment name>/api/v2.0`
   - Use basic authentication with username and password (a so-called web service access key) against the common API endpoint that includes the user domain, for example `https://api.businesscentral.dynamics.com/v2.0/production/cronus.com/api/v2.0`.

     IMPORTANT

     When going into production, you should use Azure Active Directory (AAD)/OAuth v2 authentication and the common endpoint `https://api.businesscentral.dynamics.com/v2.0/production/api/v2.0`. For exploring and initial development, you can use basic authentication.

For constructing the URL to the environment, the path needs to contain the environment name. To get all environments for the tenant call `GET https://api.businesscentral.dynamics.com/environments/v2.0/`. That will return the name for all environments in the tenant. OAuth required for this endpoint. See Exploring the APIs with Postman and AAD authentication below.

In the following sections you can read more about setting up the two types of authentication and using both authentication methods in Postman.

APIs can also be explored through the OpenAPI specification for Business Central.

Setting up basic authentication

If you prefer to set up an environment with basic authentication just to explore the APIs, you can skip setting up the AAD based authentication for now and proceed with the steps below. If you, however, want to go into production, you must use AAD/Oauth v2 authentication, see the section Setting up Azure Active Directory (AAD) based authentication.

1. To set up basic authentication, log into your tenant, and in the **Search** field, enter **Users** and then select the relevant link.
2. Select the user to add access for, and on the **User Card** page, in the **Web Service Access Key** field, generate a key.
3. Copy the generated key and use it as the password for the username.

Now that we have the username and password, we can connect and authenticate. You can do this from code, or API explorers such as Postman or Fiddler. In the Exploring the APIs with Postman and basic authentication section we will use Postman.

Setting up Azure Active Directory (AAD) based authentication

Sign in to the Azure Portal to register Dynamics 365 Business Central as an app and thereby provide access to Dynamics 365 Business Central for users in the directory.

1. Follow the instructions in the Integrating applications with Azure Active Directory article. The next steps elaborate on some of the specific settings you must enable.
2. On the **API permissions** page for your app, click the **Add a permission** button.
3. Make sure the **Microsoft APIs** tab is selected. In the *Commonly used Microsoft APIs* section, click on the **Dynamics 365 Business Central** and select **Delegated permissions**.
4. Ensure that the right permission is checked: **user_impersonation**. Use the search box if necessary.
5. Click the **Add permissions** button.

> **NOTE**
> If **Dynamics 365** does not show up in search, it's because the tenant does not have any knowledge of Dynamics 365. To make it visible, an easy way is to register for a free trial for Dynamics 365 Business Central with a user from the directory.

6. From the **Certificates & secrets** page, in the **Client secrets** section, choose **New client secret**:
   - Type a key description (of instance app secret),
   - Select a key duration of either In 1 year, In 2 years, or Never Expires.
   - When you press the Add button, the key value will be displayed, copy, and save the value in a safe location.

> **NOTE**
> You'll need this key later to configure the project in Visual Studio. This key value will not be displayed again, nor retrievable by any other means, so record it as soon as it is visible from the Azure portal.

You have now set up the AAD based authentication. Next, you can go exploring the APIs, see the Exploring the APIs with Postman and AAD authentication section below.

Exploring the APIs with Postman and basic authentication

In this `Hello World` example, we are going over the basic steps required to retrieve the list of customers in our trial tenant. This example is based on running with basic authentication.

1. First, in Postman, set up a `GET` call to the base API URL.
   - When you call the base API URL, you will get a list of all the available APIs. You can append `$metadata` to the URL to also get information about the fields in the APIs. The list of supported APIs and fields information can also be found in the API documentation.
   - Since we are using basic authentication, we need to include the users domain in the URL, for example, call

```
GET https://api.businesscentral.dynamics.com/v2.0/<your tenant domain>/<environment
name>/api/v2.0
```

.

> **NOTE**
>
> The parameter `<your tenant domain>` is your default Azure Active Directory GUID.

2. On the **Authorization** tab in Postman select **Basic Auth** in the **Type** and provide the Username and **Web Service Access Key** from above as password.

3. Choose **Send** in Postman to execute the call, and inspect the returned body, which should include a list of the APIs.

Exploring the APIs with Postman and AAD authentication

In this `Hello World` example, we are going over the basic steps required to retrieve the list of customers in our trial tenant. This example is based on running with AAD authentication.

1. First, in Postman, set up a `GET` call to the base API URL.
   - When you call the base API URL, you will get a list of all the available APIs. You can append `$metadata` to the URL to also get information about the fields in the APIs. The list of supported APIs and fields information can also be found in the API documentation, for example, call
     `GET https://api.businesscentral.dynamics.com/v2.0/environment name/api/v2.0`
2. On the **Authorization** tab in Postman select **OAuth 2.0** in the **Type** and then choose **Get New Access Token**.
3. In the **GET NEW ACCESS TOKEN** window, enter the following information as specified below:
   - In the **Token name** field, choose a descriptive name.
   - In the **Grant type** field, choose **Authorization Code**.
   - In the **Callback URL** field, specify the URL specified as the sign-on URL/Reply URL in the Azure Portal.
   - In the **Auth URL** field, specify a URL such as
     ```
     https://login.windows.net/<your tenant domain>/oauth2/authorize?
     resource=https://api.businesscentral.dynamics.com
     ```
     .
   - In the **Access Token URL** field, specify a URL such as
     ```
     https://login.windows.net/<your tenant domain>/oauth2/token?
     resource=https://api.businesscentral.dynamics.com
     ```
     .
   - In the **Client ID** field, enter the Application ID from the registered app in Azure Portal.
   - In the **Client Secret** field, enter the key generated under **Keys** that you copied in step 6 in the Setting up Azure Active Directory (AAD) based authentication.
   - In the **Client Authentication** field, choose the **Send client credentials in body** option.
4. Choose the **Request token** button. The first time you log in, you will get prompted for consent.
5. Scroll down and choose **Use token** button.
   An Authorization request header is now added containing the Bearer token.
6. Choose **Send** in Postman to execute the call, and inspect the returned body, which should include a list of the APIs.

NOTE

For OAuth for testing purposes, a multi-tenant AAD app has been created. Admin consent is needed before the ADD app can be used. Information is as follows:

- Grant Type: Implict
- Callback URL: https://localhost
- Auth URL: https://login.microsoftonline.com/common/oauth2/authorize?resource=https://api.businesscentral.dynamics.com
- Client ID: 060af3ac-70c3-4c14-92bb-8a88230f3f38

Calling the API

Each resource is uniquely identified through an ID, see the following example of calling `GET <endpoint>/companies`:

```
{
    "@odata.context": "<endpoint>/$metadata#companies",
    "value": [
        {
            "id": "bb6d48b6-c7b2-4a38-9a93-ad5506407f12",
            "systemVersion": "18453",
            "name": "CRONUS USA, Inc.",
            "displayName": "CRONUS USA, Inc.",
            "businessProfileId": ""
        }
    ]
}
```

The resource ID must be provided in the URL when trying to read or modify a resource or any of its children. The ID is provided in parenthesis () after the API endpoint. For example, to GET the "CRONUS USA, Inc." company details, you must call `<endpoint>/companies(bb6d48b6-c7b2-4a38-9a93-ad5506407f12)/`.

All resources, such as customers, invoices etc., live in the context of a parent company, of which there can be more than one in the Dynamics 365 Business Central tenant. Therefore, it is a requirement to provide the company ID in the URL for all resource API calls. To GET all customers in the "CRONUS USA, Inc." company, we must call a GET on the URL `<endpoint>/companies(bb6d48b6-c7b2-4a38-9a93-ad5506407f12)/customers`.

See Also

Developing a Custom API
Using Filtering With APIs
Tips for Working with APIs

Tips for working with the APIs

2/17/2021 • 3 minutes to read • Edit Online

GET

- Call (GET) the endpoint to list all the API
- Call (GET) the endpoint with `$metadata` to list all metadata for the API
- Calling a resource API (GET) will return a list of all instances of the resource type
- Each resource is uniquely identified through an ID, see the following example:

```
{
    "@odata.context": "<endpoint>/$metadata#companies",
    "value": [
        {
            "id": "bb6d48b6-c7b2-4a38-9a93-ad5506407f12",
            "systemVersion": "18453",
            "name": "CRONUS USA, Inc.",
            "displayName": "CRONUS USA, Inc.",
            "businessProfileId": ""
        }
    ]
}
```

- The resource ID must be provided in the URL when trying to read or modify a resource or any of its children. The ID is provided in () after the API endpoint. For example, to GET the "CRONUS USA, Inc." company details, you must call `<endpoint>/companies(bb6d48b6-c7b2-4a38-9a93-ad5506407f12)/`
- All resources live in the context of a parent company, which means that the company ID must be provided in the URL for all resource API calls. For example, to GET all customers in the "CRONUS USA, Inc." company, you must call `<endpoint>/companies(bb6d48b6-c7b2-4a38-9a93-ad5506407f12)/customers`

Accept-Language

By specifying `Accept-Language` in the request header, you can set a specific language for your web service response. It's strongly recommended to use this setting, if your app is dependent on a web service response to be in a specific language. If `Accept-Language` is set, it will override default settings. This setting also controls the regional formatting settings, affecting behavior such as how date and time will be formatted.

One of the most common examples is showing error messages to the users in their language. To see which possible error messages to display, see Error Codes. Another common example is displaying reports in a specific language, see the example below for how to specify `Accept-Language`. The following example sets the language to always be `en-US`.

Example

```
GET
businesscentralPrefix/companies({id})/salesInvoices({salesInvoiceId})/pdfDocument({salesInvoiceId})/content
```

Request headers

HEADER	VALUE

HEADER	VALUE
Authorization	Bearer {token}. Required.
Accept-Language	en-US

Request body

Do not supply a request body for this method.

Reponse

If successful, this method returns a `200 OK` response code and a report PDF file in the response body.

OData transactional $batch requests

> **APPLIES TO:** Business Central 2020 release wave 2 (version 17.1) and later

It's possible to specify that all inner requests in a certain OData $batch request are processed in a transactional way. If one of the inner requests fails after another request (or requests) has committed changes, all changes within a batch will be reverted as if the batch request never happened. Transactional $batch requests are useful in scenarios where a single business operation spans multiple requests, because they prevent adverse effects if parts of the operation fail. Also, they can improve performance by reducing the number of requests the client needs to do when errors occur.

To enable transactional batch behavior, include the `Isolation: snapshot` header with the $batch request.

Example

```
POST businesscentralPrefix/api/v2.0/$batch HTTP/1.1
```

Request headers

HEADER	VALUE
Authorization	Bearer {token}. Required.
Content-Type	application/json. Required for JSON Batch content.
Accept	application/json
Isolation	snapshot

Request body

The following request body contains three inner requests. The first two requests should execute successfully. The last request includes content that triggers a validation failure, so the request fails.

```
{
	"requests": [
		{
			"method": "PATCH",
			"url": "items(7ce0af2e-0963-460f-9b1f-0014aea3e117)",
			"headers": {
				"Company": "CRONUS International Ltd.",
				"Content-Type": "application/json",
				"If-Match": "*"
			},
			"body": {
				"displayName": "Touring Bicycle v2"
			}
		},
		{
			"method": "PATCH",
			"url": "items(7ce0af2e-0963-460f-9b1f-0014aea3e117)/picture(7ce0af2e-0963-460f-9b1f-
0014aea3e117)/contentValue",
			"headers": {
				"Company": "CRONUS International Ltd.",
				"Content-Type": "application/json",
				"If-Match": "*"
			},
			"body": {
				"value": "Picture Blob"
			}
		},
		{
			"method": "PATCH",
			"url": "items(7ce0af2e-0963-460f-9b1f-0014aea3e117)",
			"headers": {
				"Company": "CRONUS International Ltd.",
				"Content-Type": "application/json",
				"If-Match": "*"
			},
			"body": {
				"type": "Invalid Type"
			}
		}
	]
}
```

Response

The following response includes successful responses for the first two inner requests, and an error for the last inner request. Because the `Isolation: snapshot` header was present in the batch request, none of the changes made by either of the successful inner requests are applied after completion of the batch request.

```
{
	"responses": [
		{
			"id": null,
			"status": 200,
			"headers": {
				"content-type": "application/json; odata.metadata=minimal",
				"odata-version": "4.0"
			},
			"body": {
				"displayName": "Touring Bicycle v2",
				// ...
			}
		},
		{
			"id": null,
			"status": 204,
			"headers": {}
		},
		{
			"id": null,
			"status": 400,
			"headers": {
				"content-type": "application/json; odata.metadata=minimal",
				"odata-version": "4.0"
			},
			"body": {
				"error": {
					"code": "Unknown",
					"message": "'Invalid Type' is not an option. The existing options are:
Inventory,Service,Non-Inventory  CorrelationId:  x."
				}
			}
		}
	]
}
```

See Also

Using Filtering With APIs
Performance Articles For Developers

Developing a Custom API

2/17/2021 • 7 minutes to read • Edit Online

This walkthrough describes how to develop a custom API page by using an AL extension and accessing it to retrieve responses and make updates through the API.

About this walkthrough

At a high-level, this process involves the following tasks:

1. Develop an AL extension with a new API page.
2. Add necessary fields, properties, and subpages to the API page.
3. Access the API page and get the response.

> **TIP**
> The code in this sample has also been published to the BCTech repo. For more information, see Introduction to Custom API.

Prerequisites

This walkthrough requires the following:

- Business Central, including the following:
  - The CRONUS International Ltd. demonstration data.
  - Visual Studio Code with the AL Language extension installed. For more information, see Getting Started with AL and AL Language Extension Configuration. The AL Language extension for Visual Studio is free, and you can download it from Marketplace.

Creating source tables for the API

To expose data in an API page, the first thing needed is a source table. For the purpose of this walkthrough we will create a table object that describes the schema for a car brand.

1. Create a new table. For more information, see Tables Overview.
2. Name the table **Car Brand**, and specify **50100** as the table ID.
3. Add any necessary fields for a car brand as shown below:

```
table 50100 "Car Brand"
{
    DataClassification = CustomerContent;
    Caption = 'Car Brand';

    fields
    {
        field(1; Name; Text[100])
        {
            Caption = 'Name';
        }
        field(2; Description; Text[100])
        {
            Caption = 'Description';
        }
        field(3; "Country"; Text[100])
        {
            Caption = 'Brand Id';
        }
    }

    keys
    {
        key(PK; Name)
        {
            Clustered = true;
        }
    }
}
```

4. Now, create a new table for **Car Model**, and specify **50101** as the table ID.
5. Add any necessary fields for a car model as shown in the example below. Make sure to have a field for **Brand Id** and that **TableRelation** is set to **"Car Brand".SystemId**.

```
table 50101 "Car Model"
{
    DataClassification = CustomerContent;
    Caption = 'Car Model';

    fields
    {
        field(1; Name; Text[100])
        {
            Caption = 'Name';
        }
        field(2; Description; Text[100])
        {
            Caption = 'Description';
        }
        field(3; "Brand Id"; Guid)
        {
            TableRelation = "Car Brand".SystemId;
            Caption = 'Brand Id';
        }
        field(4; Power; Integer)
        {
            Caption = 'Power (cc)';
        }
        field(5; "Fuel Type"; Enum "Fuel Type")
        {
            Caption = 'Fuel Type';
        }
    }

    keys
    {
        key(PK; Name, "Brand Id")
        {
            Clustered = true;
        }
    }
}

enum 50100 "Fuel Type"
{
    Extensible = true;
    value(0; Petrol)
    {
        Caption = 'Petrol';
    }
    value(1; Diesel)
    {
        Caption = 'Diesel';
    }
    value(2; Electric)
    {
        Caption = 'Electric';
    }
}
```

TIP

As it can be seen in field number 5 **"Fuel Type"**, make sure to use Enums instead of Options. When they are used in API pages, Options are generated as type strings in the metadata: `<Property Name="fuelType" Type="Edm.String"/>`. Whereas Enums have their own types and all available Enum members are generated in the metadata: `<Property Name="fuelType" Type="Microsoft.NAV.fuelType"/>`.

```
< EnumType Name="fuelType" Type="Microsoft.NAV.fuelType">
            <Member Name="Petrol" Value="0"/>
            <Member Name="Diesel" Value="1"/>
            <Member Name="Electric" Value="2"/>
   </EnumType>
```

Creating API pages

In the following, we will create two API pages for both **Car Brand** and **Car Model** tables. API pages are specific pages with the `PageType` property set to `API`. For more information, see API Page Type.

To create API pages to display Car Brand and Car Model

1. Create a new API page.
2. Name the page **API Car Model**, and specify **50101** as the page ID.
3. Specify the **Car Model** table as the source table.
4. Specify `APIVersion`, `APIPublisher`, `APIGroup`, `EntityName`, and `EntitySetName` for your API page. These properties will affect your custom endpoint:
   `https://api.businesscentral.dynamics.com/v1.0/<user domain name>/api/<API publisher>/<API group>/<API version>/companies(<company id>)/carModel`
   . For more information, see Business Central API endpoints and Calling the API.
5. Specify `EntityCaption` and `EntitySetCaption`. These two properties are generated in the entityDefinitions
   `https://api.businesscentral.dynamics.com/v1.0/<user domain name>/api/<API publisher>/<API group>/<API version>/entityDefinitions`
   which are localized and translatable.
6. Make sure to set the `ODataKeyFields` property to `SystemId`. A SystemId field is a GUID data type field that specifies a unique, immutable (read-only) identifier for records in the table. For more information, see Table Object.

```
page 50101 "API Car Model"
{
    PageType = API;

    APIVersion = 'v1.0';
    APIPublisher = 'bctech';
    APIGroup = 'demo';

    EntityCaption = 'CarModel';
    EntitySetCaption = 'CarModels';
    EntityName = 'carModel';
    EntitySetName = 'carModels';

    ODataKeyFields = SystemId;
    SourceTable = "Car Model";

    Extensible = false;
    DelayedInsert = true;

    layout
    {
        area(content)
        {
            repeater(Group)
            {
                field(id; Rec.SystemId)
                {
                    Caption = 'Id';
                    Editable = false;
                }
                field(name; Rec.Name)
                {
                    Caption = 'Name';
                }
                field(description; Rec.Description)
                {
                    Caption = 'Description';
                }
                field(brandId; Rec."Brand Id")
                {
                    Caption = 'Brand Id';
                }
                field(power; Rec.Power)
                {
                    Caption = 'Power';
                }
                field(fuelType; Rec."Fuel Type")
                {
                    Caption = 'Fuel Type';
                }
            }
        }
    }
}
```

7. Now, repeat the steps 1-6 for **API Car Brand** page.
8. You can define a **API Car Model** part in **API Car Brand** page. Make sure to use the SystemId field when defining the SubPageLink. This will generate **ReferentialConstraints** property in the metadata as below:

```
<NavigationProperty Name="carModels" Type="Collection(Microsoft.NAV.carModel)" Partner="carBrand"
ContainsTarget="true">
<ReferentialConstraint Property="id" ReferencedProperty="brandId"/>
</NavigationProperty>
```

And the **API Car Brand** page:

```
page 50100 "API Car Brand"
{
    PageType = API;

    APIVersion = 'v1.0';
    APIPublisher = 'bctech';
    APIGroup = 'demo';

    EntityCaption = 'CarBrand';
    EntitySetCaption = 'CarBrands';
    EntityName = 'carBrand';
    EntitySetName = 'carBrands';

    ODataKeyFields = SystemId;
    SourceTable = "Car Brand";

    Extensible = false;
    DelayedInsert = true;

    layout
    {
        area(content)
        {
            repeater(Group)
            {
                field(id; Rec.SystemId)
                {
                    Caption = 'Id';
                    Editable = false;
                }

                field(name; Rec.Name)
                {
                    Caption = 'Name';
                }
                field(description; Rec.Description)
                {
                    Caption = 'Description';
                }
                field(country; Rec.Country)
                {
                    Caption = 'Country';
                }
                part(carModels; "API Car Model")
                {
                    Caption = 'Car Models';
                    EntityName = 'carModel';
                    EntitySetName = 'carModels';
                    SubPageLink = "Brand Id" = Field(SystemId);
                }
            }
        }
    }
}
```

TIP

Parts are defined as 1-N relationship by default. You can, however, define it to be as 1-0, 1-1 relationship. In order to achieve that add the **CaptionML = ENU = 'Multiplicity=ZeroOrOne';** property in your part as shown below:

```
part(carModels; "API Car Model")
   {
       CaptionML = ENU = 'Multiplicity=ZeroOrOne';
       EntityName = 'carModel';
       EntitySetName = 'carModels';
       SubPageLink = "Brand Id" = Field(SystemId);
   }
```

This will change the **NavigationalProperty** in the metadata from a Collection to an object as shown below:

```
<NavigationProperty Name="carModel" Type="Microsoft.NAV.carModel" ContainsTarget="true">
           <ReferentialConstraint Property="id" ReferencedProperty="brandId"/>
       </NavigationProperty>
```

Using carBrand and carModel APIs

Both API pages support create, read, update, and delete operations. If you want to disallow create, update, and delete operations, you can use the **InsertAllowed**, **ModifyAllowed**, and **DeleteAllowed** properties respectively.

Now, we will create a car brand:

```
POST https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/bctech/demo/v1.0/companies(<company
id>)/carBrands

{
    "name": "CARBRAND1",
    "description": "Car Brand 1",
    "country": "Italy"
}
```

We can make a `GET` request to retrieve the car brands:

```
GET https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/bctech/demo/v1.0/companies(<company
id>)/carBrands
```

Which will result in following response:

```
{

"@odata.context":"https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/bctech/demo/v1.0/$meta
data#companies(<company id>)/carBrands",
   "value":[
      {
         "@odata.etag":"W/\"JzQ0O2c4UTNaRHErODdzODZnVlJxN2tNTkt3SHBwajNBaHNSdStNeEFONGUwVkE9MTswMDsn\"",
         "id":"24cafc3a-b1fe-ea11-9306-000d3a482952",
         "name":"CARBRAND1",
         "description":"Car Brand 1",
         "country":"Italy"
      }
   ]
}
```

We can now create a car model that belongs to the car brand that we just created. Since the navigational property is defined in the API page as a part, we can create a car model in one of the following different ways:

Example 1

```
POST https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/bctech/demo/v1.0/companies(<company
id>))/carBrands(24cafc3a-b1fe-ea11-9306-000d3a482952)/carModels
{
    "name": "MODEL1",
    "description": "Model 1",
    "power": 1700,
    "fuelType": "Petrol"
}
```

Example 2

```
POST https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/bctech/demo/v1.0/companies(<company
id>))//carModels
{
    "name": "MODEL1",
    "brandId": "24cafc3a-b1fe-ea11-9306-000d3a482952",
    "description": "Model 1",
    "power": 1700,
    "fuelType": "Petrol"
}
```

Example 3

And the navigational property also allows us to do a deep insert; deep insert is the creation of an entity instance and related entity instances, in a single `POST` request. So you can combine car brand and car model creation in a single request as illustrated below:

```
POST https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/bctech/demo/v1.0/companies(<company
id>))//carBrands
{
    "name": "CARBRAND2",
    "description": "Car Brand 2",
    "country": "Germany",
    "carModels": [{
        "name": "MODELA",
        "description": "Model A",
        "power": 0,
        "fuelType": "Electric"
       },
       {
        "name": "MODELB",
        "description": "Model B",
        "power": 0,
        "fuelType": "Electric"
       }]
}
```

> **NOTE**
>
> The sample code is published to the BCTech repo. For more information, see Introduction to Custom API.

General tips for custom APIs

1. Use SystemId as the OData primary key.

2. Make sure that all the table fields in TableRelations/SubPageLinks are available in the API pages and make sure to define the relationship multiplicity (1-0/1-1 or 1-N).
    - Doing so enables the platform to generate ReferentialConstraints, that OData consumers can use to understand the relations between entities
    - The platform will also create bi-directional relationship if possible, allowing consumers to access to the parent by just adding "/parentEntity" in the URI
3. Use Enumerations.
4. Make sure to localize your custom API pages:
    - Use `EntityCaption` and `EntitySetCaption` properties
    - Use captions for Enums
    - All these localizations can be retrieved through
      ```
      https://api.businesscentral.dynamics.com/v2.0/<environmentName>/api/<API publisher>/<API group>/<API version>/entityDefinitions
      ```

See Also

Getting Started with AL
API Page Type
APIPublisher Property
APIGroup Property
APIVersion Property
EntityName Property
EntitySetName Property
Developing Extensions

Customizing an Integration with Microsoft Dataverse

2/17/2021 • 17 minutes to read • Edit Online

NOTE

Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

This walkthrough describes how to customize an integration between Business Central and Microsoft Dataverse. The walkthrough will guide you through setting up an integration between an employee in Business Central and a worker in Microsoft Dataverse.

TIP

Sample code that shows how to integrate an employee in Business Central and a worker in Microsoft Dataverse is available in the BCTech repository.

About this walkthrough

This walkthrough describes how to integrate new and existing extensions with Microsoft Dataverse. At a high-level, those process involve the following tasks:

1. Develop an AL extension to integrate tables in Microsoft Dataverse and Business Central. For more information, see Developing Extensions in AL.
2. Create an integration table object in Business Central for mapping a Microsoft Dataverse table to a Business Central record type.
3. Use a Microsoft Dataverse integration table as the data source for a page in Business Central that displays data from Microsoft Dataverse table rows.
4. Extend a page in Business Central for coupling and synchronizing table rows in Microsoft Dataverse with table records in Business Central.
5. Use events to create an integration table and a field mapping between a table in Business Central and an integration table for Microsoft Dataverse.
6. Develop another AL extension to extend an existing integration between tables in Microsoft Dataverse and Business Central.
7. Create a table extension for an existing integration table object.
8. Use events to add custom integration field mappings for existing integration table mappings.

NOTE

The customization in this walkthrough is done entirely in Business Central online, and does not describe how to modify your Microsoft Dataverse solution, for example, by adding or modifying tables and forms.

Prerequisites

This walkthrough requires the following:

- Microsoft Dataverse, including the following:
  - Worker table.

    > **NOTE**
    > To get the worker table you must install the Talent Core HR solution. For more information, see Microsoft Dataverse tables.

  - The URL of your Microsoft Dataverse environment.
  - The user name and password of a user account that has full permissions to add and modify tables.
- Business Central, including the following:
  - The CRONUS International Ltd. demonstration data.
  - Integration with Microsoft Dataverse is enabled, including the default synchronization setup and a working connection between Business Central and Microsoft Dataverse.
  - Visual Studio Code with the AL Language extension installed. For more information, see Getting Started with AL and AL Language Extension Configuration. The AL Language extension for Visual Studio is free, and you can download it from Marketplace.

    > **NOTE**
    > Make sure that your integration user has permission to access the Worker table in Microsoft Dataverse.

Create an integration table in Business Central for the Microsoft Dataverse table

To integrate data from a Microsoft Dataverse table into Business Central, you must create a table object in Business Central that is based on the Microsoft Dataverse table, and then import the new table into the Business Central database. For this walkthrough we will create a table object that describes the schema for the **Worker** table in Microsoft Dataverse in the Business Central database.

> **NOTE**
> The table can contain some or all of the fields from the Microsoft Dataverse table. However, if you want to set up bi-directional synchronization you must include all fields in the table.

To create the integration table for the worker table in Business Central

1. Create a new AL extension. For more information, see Developing Extensions in AL.
2. Export the **AL Table Proxy Generator** tool called **altpgen.exe** from the Visual Studio Code AL extension. This executable tool allows you to create integration tables. When you have installed the AL Language extension, go to the equivalent of this folder: `C:\Users\<myname>\.vscode\extensions\microsoft.al-4.0.209721\bin` and find the `altpgen.exe` file. For more information, see AL Table Proxy Generator.
3. In PowerShell, run the tool with the following arguments:

```
-project:<Your AL project folder>
-packagecachepath:<Your AL project cache folder>
-serviceuri:<Microsoft Dataverse server URL>
-entities:cds_worker
-baseid:50000
```

This starts the process for creating the table. When completed, the output path contains the `.al` file that contains the description of the **CDS Worker** integration table with ID 50000. This table is set to the table type **CDS**.

Create a page for displaying Microsoft Dataverse data

For scenarios where we want to view Microsoft Dataverse data for a specific table, we can create a page object that uses the integration table for the Microsoft Dataverse table as its data source. For example, we might want to have a list page that displays the current records in a Microsoft Dataverse table, such as all workers. In this walkthrough we will create a list page that uses the generated integration table **CDS Worker** with ID 50000 as its data source.

To create a list page to display Microsoft Dataverse workers

1. Create a new page. For more information, see Pages Overview.
2. Name the page **CDS Worker List**, and specify **50001** as the page ID.
3. Specify the **CDS Worker** integration table as the source table as shown below:

```
page 50001 "CDS Worker List"
{
    PageType = List;
    SourceTable = "CDS Worker";
    Editable = false;
    ApplicationArea = All;
    UsageCategory = Lists;
    ...

    layout
    {
        // add fields to display on the page
    }

    actions
    {
        area(processing)
        {
            action(CreateFromCDS)
            {
                ApplicationArea = All;
                Caption = 'Create in Business Central';
                Promoted = true;
                PromotedCategory = Process;
                ToolTip = 'Generate the table from the coupled Microsoft Dataverse worker.';

                trigger OnAction()
                var
                    CDSWorker: Record "CDS Worker";
                    CRMIntegrationManagement: Codeunit "CRM Integration Management";
                begin
                    CurrPage.SetSelectionFilter(CDSWorker);
                    CRMIntegrationManagement.CreateNewRecordsFromCRM(CDSWorker);
                end;
            }
        }
    }

    var
        CurrentlyCoupledCDSWorker: Record "CDS Worker";

    trigger OnInit()
    begin
        Codeunit.Run(Codeunit::"CRM Integration Management");
    end;

    procedure SetCurrentlyCoupledCDSWorker(CDSWorker: Record "CDS Worker")
    begin
        CurrentlyCoupledCDSWorker := CDSWorker;
    end;
}
```

4. Add the fields from the integration table to display on the page in the `layout` section.

Enable coupling and synchronization between Worker in Microsoft Dataverse and in Business Central

To connect a Business Central table record with a Microsoft Dataverse table row, you create a coupling. A coupling consists of the primary ID, which is typically a GUID, from a Microsoft Dataverse row and the integration ID, also often a GUID, from Business Central.

1. Create a new codeunit.
2. In codeunit **CRM Setup Defaults** (ID 5334), subscribe to the **OnGetCDSTableNo** event, as follows:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"CRM Setup Defaults", 'OnGetCDSTableNo', '', false, false)]
local procedure HandleOnGetCDSTableNo(BCTableNo: Integer; var CDSTableNo: Integer; var handled: Boolean)
begin
    if BCTableNo = DATABASE::Employee then begin
        CDSTableNo := DATABASE::"CDS Worker";
        handled := true;
    end;
end;
```

You can now use the table to create a page for coupling Business Central records with Microsoft Dataverse rows.

3. In codeunit **Lookup CRM Tables** (ID 5332), subscribe to the **OnLookupCRMTables** event, as follows:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Lookup CRM Tables", 'OnLookupCRMTables', '', true, true)]
local procedure HandleOnLookupCRMTables(CRMTableID: Integer; NAVTableId: Integer; SavedCRMId: Guid; var
CRMId: Guid; IntTableFilter: Text; var Handled: Boolean)
begin
    if CRMTableID = Database::"CDS Worker" then
        Handled := LookupCDSWorker(SavedCRMId, CRMId, IntTableFilter);
end;

local procedure LookupCDSWorker(SavedCRMId: Guid; var CRMId: Guid; IntTableFilter: Text): Boolean
var
    CDSWorker: Record "CDS Worker";
    OriginalCDSWorker: Record "CDS Worker";
    CDSWorkerList: Page "CDS Worker List";
begin
    if not IsNullGuid(CRMId) then begin
        if CDSWorker.Get(CRMId) then
            CDSWorkerList.SetRecord(CDSWorker);
        if not IsNullGuid(SavedCRMId) then
            if OriginalCDSWorker.Get(SavedCRMId) then
                CDSWorkerList.SetCurrentlyCoupledCDSWorker(OriginalCDSWorker);
    end;

    CDSWorker.SetView(IntTableFilter);
    CDSWorkerList.SetTableView(CDSWorker);
    CDSWorkerList.LookupMode(true);
    if CDSWorkerList.RunModal = ACTION::LookupOK then begin
        CDSWorkerList.GetRecord(CDSWorker);
        CRMId := CDSWorker.WorkerId;
        exit(true);
    end;
    exit(false);
end;
```

To create actions on the employee page for managing coupling and synchronization

To enable users to create couplings between records in the two systems, we will extend the **Employee Card** page with actions for creating and deleting couplings, and for synchronizing. The following code example adds those actions to **Employee Card**.

```
pageextension 50101 "Employee Synch Extension" extends "Employee Card"
{
    actions
    {
        addlast(navigation)
        {
            group(ActionGroupCDS)
            {
                Caption = 'CDS';
                Visible = CDSIntegrationEnabled;

                action(CDSSynchronizeNow)
```

```
                {
                    Caption = 'Synchronize';
                    ApplicationArea = All;
                    Visible = true;
                    Image = Refresh;
                    Enabled = CDSIsCoupledToRecord;
                    ToolTip = 'Send or get updated data to or from Microsoft Dataverse.';

                    trigger OnAction()
                    var
                        CRMIntegrationManagement: Codeunit "CRM Integration Management";
                    begin
                        CRMIntegrationManagement.UpdateOneNow(RecordId);
                    end;
                }
                action(ShowLog)
                {
                    Caption = 'Synchronization Log';
                    ApplicationArea = All;
                    Visible = true;
                    Image = Log;
                    ToolTip = 'View integration synchronization jobs for the customer table.';

                    trigger OnAction()
                    var
                        CRMIntegrationManagement: Codeunit "CRM Integration Management";
                    begin
                        CRMIntegrationManagement.ShowLog(RecordId);
                    end;
                }
                group(Coupling)
                {
                    Caption = 'Coupling';
                    Image = LinkAccount;
                    ToolTip = 'Create, change, or delete a coupling between the Business Central record and
a Microsoft Dataverse row.';

                    action(ManageCDSCoupling)
                    {
                        Caption = 'Set Up Coupling';
                        ApplicationArea = All;
                        Visible = true;
                        Image = LinkAccount;
                        ToolTip = 'Create or modify the coupling to a Microsoft Dataverse Worker.';

                        trigger OnAction()
                        var
                            CRMIntegrationManagement: Codeunit "CRM Integration Management";
                        begin
                            CRMIntegrationManagement.DefineCoupling(RecordId);
                        end;
                    }
                    action(DeleteCDSCoupling)
                    {
                        Caption = 'Delete Coupling';
                        ApplicationArea = All;
                        Visible = true;
                        Image = UnLinkAccount;
                        Enabled = CDSIsCoupledToRecord;
                        ToolTip = 'Delete the coupling to a Microsoft Dataverse Worker.';

                        trigger OnAction()
                        var
                            CRMCouplingManagement: Codeunit "CRM Coupling Management";
                        begin
                            CRMCouplingManagement.RemoveCoupling(RecordId);
                        end;
                    }
                }
```

```
            }
        }
    }

    trigger OnOpenPage()
    begin
        CDSIntegrationEnabled := CRMIntegrationManagement.IsCDSIntegrationEnabled();
    end;

    trigger OnAfterGetCurrRecord()
    begin
        if CDSIntegrationEnabled then
            CDSIsCoupledToRecord := CRMCouplingManagement.IsRecordCoupledToCRM(RecordId);
    end;

    var
        CRMIntegrationManagement: Codeunit "CRM Integration Management";
        CRMCouplingManagement: Codeunit "CRM Coupling Management";
        CDSIntegrationEnabled: Boolean;
        CDSIsCoupledToRecord: Boolean;

}
```

Customizing Uncoupling

Tables might require custom code to remove couplings, for example, to change tables before or after uncoupling. To enable custom uncoupling, specify the uncoupling codeunit when you create an integration table mapping. To do this, adjust the function **InsertIntegrationTableMapping** in your codeunit, as follows:

```
local procedure InsertIntegrationTableMapping(var IntegrationTableMapping: Record "Integration Table
Mapping"; MappingName: Code[20]; TableNo: Integer; IntegrationTableNo: Integer; IntegrationTableUIDFieldNo:
Integer; IntegrationTableModifiedFieldNo: Integer; TableConfigTemplateCode: Code[10];
IntegrationTableConfigTemplateCode: Code[10]; SynchOnlyCoupledRecords: Boolean)
begin
    IntegrationTableMapping.CreateRecord(MappingName, TableNo, IntegrationTableNo,
IntegrationTableUIDFieldNo, IntegrationTableModifiedFieldNo, TableConfigTemplateCode,
IntegrationTableConfigTemplateCode, SynchOnlyCoupledRecords,
IntegrationTableMapping.Direction::Bidirectional, 'CDS', Codeunit::"CRM Integration Table Synch.",
Codeunit::"CDS Int. Table Uncouple");
end;
```

During the custom uncoupling process, codeunit Int. Rec. Uncouple Invoke (ID 5357) raises and publishes events. You can add code that subscribes to these events so that you can add custom logic at different stages of the uncoupling process. The following table describes the events that are published by codeunit Int. Rec. Uncouple Invoke.

- **OnBeforeUncoupleRecord** - Occurs before remove coupling, and can be used to change data before uncoupling. For an example, see codeunit CDS Int. Table. Subscriber, which includes the event subscriber function HandleOnBeforeUncoupleRecord. The event resets the company ID on the uncoupled tables in Microsoft Dataverse.
- **OnAfterUncoupleRecord** - Occurs after coupling is removed, and can be used to change data after uncoupling. For an example, see codeunit CDS Int. Table. Subscriber, which includes the event subscriber function HandleOnAfterUncoupleRecord. The event removes couplings to the contacts linked to the uncoupled customers and vendors.

For more information about how to subscribe to events, see Subscribing to Events.

Be aware that custom uncoupling is running in background as it could modify Microsoft Dataverse tables and it might take significant time.

TIP

To reset Company ID on uncoupling custom tables just like the base Microsoft Dataverse tables, users can subscribe to the OnHasCompanyIdField event in codeunit CDS Integration Mgt. (ID 7200), as follows:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"CDS Integration Mgt.", 'OnHasCompanyIdField', '',
false, false)]
local procedure HandleOnHasCompanyIdField(TableId: Integer; var HasField: Boolean)
begin
  if TableId = Database::"CDS Worker" then
    HasField := true;
end;
```

Create default integration table mappings and field mappings

For synchronization to work, mappings must exist to associate the table ID and fields of the integration table (in this case, **CDS Worker**) with the table in Business Central (in this case table **Employee**). There are two types of mapping:

- **Integration table mapping** - Integration table mapping links the Business Central table to the integration table for the Microsoft Dataverse table.
- **Integration field mapping** - Field mapping links a field in a table row in Microsoft Dataverse with a field in a record in Business Central. This determines which field in Business Central corresponds to which field in Microsoft Dataverse. Typically, there are multiple field mappings for a table.

In this scenario, we will create integration table and field mappings so that we can synchronize data for a worker in Microsoft Dataverse with an employee in Business Central.

To create an integration table mapping

We can create the integration table mapping by subscribing to the **OnAfterResetConfiguration** event in codeunit **CDS Setup Defaults** (ID 7204).

1. Create a codeunit.
2. Add a local procedure called **InsertIntegrationTableMapping**, as follows:

```
local procedure InsertIntegrationTableMapping(var IntegrationTableMapping: Record "Integration Table
Mapping"; MappingName: Code[20]; TableNo: Integer; IntegrationTableNo: Integer;
IntegrationTableUIDFieldNo: Integer; IntegrationTableModifiedFieldNo: Integer;
TableConfigTemplateCode: Code[10]; IntegrationTableConfigTemplateCode: Code[10];
SynchOnlyCoupledRecords: Boolean)
begin
    IntegrationTableMapping.CreateRecord(MappingName, TableNo, IntegrationTableNo,
IntegrationTableUIDFieldNo, IntegrationTableModifiedFieldNo, TableConfigTemplateCode,
IntegrationTableConfigTemplateCode, SynchOnlyCoupledRecords,
IntegrationTableMapping.Direction::Bidirectional, 'CDS');
end;
```

3. In codeunit **CDS Setup Defaults**, subscribe to the **OnAfterResetConfiguration** event, as follows:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"CDS Setup Defaults", 'OnAfterResetConfiguration',
'', true, true)]
local procedure HandleOnAfterResetConfiguration(CDSConnectionSetup: Record "CDS Connection Setup")
var
    IntegrationTableMapping: Record "Integration Table Mapping";
    IntegrationFieldMapping: Record "Integration Field Mapping";
    CDSWorker: Record "CDS Worker";
    Employee: Record Employee;
begin
    InsertIntegrationTableMapping(
        IntegrationTableMapping, 'EMPLOYEE-WORKER',
        DATABASE::Employee, DATABASE::"CDS Worker",
        CDSWorker.FieldNo(cdm_workerId), CDSWorker.FieldNo(ModifiedOn),
        '', '', true);

    ...
end;
```

For each integration table mapping entry, there must be integration field mapping entries to map the fields of the records in the table and the integration table. The next step is to add integration field mappings for each field in the **Employee** table in Business Central that we want to map to the **Worker** table in Microsoft Dataverse.

To create integration fields mappings

To create an integration field mapping, follow these steps:

1. Add a local procedure called **InsertIntegrationFieldMapping** to the codeunit that you created in step 1 of the previous process, as follows:

```
procedure InsertIntegrationFieldMapping(IntegrationTableMappingName: Code[20]; TableFieldNo: Integer;
IntegrationTableFieldNo: Integer; SynchDirection: Option; ConstValue: Text; ValidateField: Boolean;
ValidateIntegrationTableField: Boolean)
var
    IntegrationFieldMapping: Record "Integration Field Mapping";
begin
    IntegrationFieldMapping.CreateRecord(IntegrationTableMappingName, TableFieldNo,
IntegrationTableFieldNo, SynchDirection,
        ConstValue, ValidateField, ValidateIntegrationTableField);
end;
```

2. In the event subscriber that we created for our integration table mapping (in step 3 in the previous process), after we insert the integration table mapping we will add field mappings, as follows:

```
InsertIntegrationFieldMapping('EMPLOYEE-WORKER', Employee.FieldNo("First Name"),
CDSWorker.FieldNo(cdm_FirstName), IntegrationFieldMapping.Direction::Bidirectional, '', true, false);
```

3. Now repeat these steps for each field that we want to map.

> **TIP**
> If a field in one of the tables does not have a corresponding field in the other table, we can use a constant value.

4. After publishing the extension, we can update the default mappings to include our new integration table mapping by opening the **CDS Connection Setup** page in Business Central and choosing **Use Default Synchronization Setup**.

Users can now manually synchronize employee records in Business Central with Worker table rows in Microsoft

Dataverse from the Business Central client.

> **TIP**
> To learn how to schedule the synchronization by using a job queue entry, examine the code on the **RecreateJobQueueEntry** function in codeunit **CRM Integration Management** (ID 5330) and see how it is called by the integration code for other Microsoft Dataverse tables in the codeunit. For more information, see Scheduling a Synchronization.

Enable customers to reset selected integration table mappings to the default settings

Customers might make changes to the integration table mappings that they later regret. To enable them to reset selected custom integration table mappings to the default, rather than all custom table mappings, follow these steps:

- In the same codeunit created for this section, add an event subscriber to **OnBeforeHandleCustomIntegrationTableMapping** and point to the default behavior, as follows:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"CRM Integration Management",
'OnBeforeHandleCustomIntegrationTableMapping', '', false, false)]
local procedure HandleCustomIntegrationTableMappingReset(var IsHandled: Boolean;
IntegrationTableMappingName: Code[20])
var
    IntegrationTableMapping: Record "Integration Table Mapping";
begin
    case IntegrationTableMappingName of
        'EMPLOYEE-WORKER':
            begin
                InsertIntegrationTableMapping(
                    IntegrationTableMapping, 'EMPLOYEE-WORKER',
                    DATABASE::Employee, DATABASE::"CDS Worker", C
                    DSWorker.FieldNo(cdm_workerId), CDSWorker.FieldNo(ModifiedOn),
                    '', '', true);
                InsertIntegrationFieldMapping('EMPLOYEE-WORKER',
                    Employee.FieldNo("First Name"), CDSWorker.FieldNo(cdm_FirstName),
                    IntegrationFieldMapping.Direction::Bidirectional, '', true, false);
                ...
                IsHandled := true;
            end;
            ...
    end;
end;
```

> **IMPORTANT**
> You must set the the IsHandled property to `true` to avoid triggering the default implementation. Otherwise, all custom table mappings will be reset to default, regardless of the user's selection.

Customizing Synchronization

When synchronizing data, some tables may require custom code to successfully synchronize data. Other tables might require the initialization of fields, the validation of relationships, or the transformation of data.

You can either use the standard transformation rules on page **Integration Field Mapping List** (ID 5361) or you can transform data programmatically. For more information, see Transformation Rules.

During the synchronization process, certain events are published and raised by codeunit **Integration Table Synch.** (ID 5335). We can add code that subscribes to these events so that we can add custom logic at different stages of the synchronization process. The following table describes the events that are published by codeunit

Integration Table Synch..

EVENT	DESCRIPTION
OnFindUnCoupledDestinationRecord	Occurs when the process tries to synchronize an uncoupled record (new record). Use this event to implement custom resolution algorithms for automatic mapping between records. For example, use this event to automatically map records by fields. For an example, see codeunit **CRM Int. Table. Subscriber**, which includes the event subscriber function **CRMTransactionCurrencyFindUncoupledDestinationRecord**. The event resolves Business Central currency codes with ISO currency codes in Microsoft Dataverse.
OnBeforeApplyRecordTemplate	Occurs before applying configuration templates to new records, and can be used to implement algorithms for determining which configuration template to use.
OnAfterApplyRecordTemplate	Occurs after configuration templates are applied to new records, and can be used to change data after configuration templates have been applied.
OnBeforeTransferRecordFields	Occurs before transferring data in modified fields (which are defined in the **Integration Field Mapping** table) from the source table to the destination table. It can be used to validate the source or destination before the data is moved.
OnAfterTransferRecordFields	Occurs after transferring modified field data (which are defined in the Integration Field Mapping table) from the source table to the destination table. It can be used to transfer additional data, validate lookups, and so on. Setting the **AdditionalFieldsWereModified** parameter will cause a destination record modification even though no fields were modified.
OnBeforeInsertRecord	Occurs before inserting a new destination record, and can be used to initialize fields, such as primary keys.
OnAfterInsertRecord	Occurs after a new destination record is inserted, and can be used to perform post-insert operations such as updating related data.
OnBeforeModifyRecord	Occurs before modifying an existing destination record, and can be used to validate or change data before modification.
OnAfterModifyRecord	Occurs after an existing destination record is modified, and can be used to perform post-modify operations such as updating related data.
OnTransferFieldData	Occurs before an existing destination field value is transferred to a source field, and can be used to perform specific transformations of data when the data types of the source and the destination field are different but can be mapped.

For more information about how to subscribe to events, see Subscribing to Events.

TIP

In order to have Company ID mapping for custom tables just like the base Microsoft Dataverse tables , users can subscribe to the **OnBeforeInsertRecord** event in codeunit **Integration Rec. Synch. Invoke** (ID 5345), as follows:

```
  [EventSubscriber(ObjectType::Codeunit, Codeunit::"Integration Rec. Synch. Invoke",
'OnBeforeInsertRecord', '', false, false)]
  local procedure HandleOnBeforeInsertRecord(SourceRecordRef: RecordRef; DestinationRecordRef:
RecordRef)
  var
      CDSIntegrationMgt: Codeunit "CDS Integration Mgt.";
  begin
  if DestinationRecordRef.Number() = Database::"CDS Worker" then
      CDSIntegrationMgt.SetCompanyId(DestinationRecordRef);
  end;
```

For more information on base Microsoft Dataverse tables, see Data Ownership Models.

Create a table extension for an integration table in Business Central

Let us explore another scenario. If we added an **Industry** field to the **Contact** table in Microsoft Dataverse, and now want to include the field in our integration with Microsoft Dataverse.

TIP

Sample code that customizes an integration between a contact in Business Central and a contact in Microsoft Dataverse by adding a new field is available in the BCTech repository.

To create the integration table extension for table "CRM Contact" (ID 5342)

1. Create a new AL extension.
2. Locate the **AL Table Proxy Generator** tool. See the previous section.
3. In PowerShell, run the tool with the following arguments:

   ```
   -project:<Your AL project folder>
   -packagecachepath:<Your AL project cache folder>
   -serviceuri:<CDS server URL>
   -entities:contact
   -baseid:60000
   ```

   The process for creating the table starts. The AL Table Proxy Generator tool finds that an integration table for the **Contact** table already exists, so it creates a table extension with only new fields; in this case **Industry**. When the process is completed, the output path contains the `WorkerExt.al` file.

Extend the contact table and page with the Industry field

To synchronize the **Industry** field we need to add the field in Business Central. The following code example extends table **Contact** and page **Contact Card** with new the field. For example:

```
tableextension 60001 ContactExtension extends Contact
{
    fields
    {
        field(70116; "Industry"; Text[100])
        {
        }
    }
}

pageextension 60001 ContactCardExtension extends "Contact Card"
{
    layout
    {
        addlast(General)
        {
            field("Industry"; "Industry")
            {
                ApplicationArea = All;
                Caption = 'Industry';
            }
        }
    }
}
```

Add new integration field mapping for Industry

Now that we have the field in both Business Central and Microsoft Dataverse, we can add a new integration field mapping for it. To do that we will subscribe to the **OnAfterResetContactContactMapping** event in codeunit **CDS Setup Defaults** (ID 7204), as follows:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"CDS Setup Defaults", 'OnAfterResetContactContactMapping',
'', true, true)]
local procedure HandleOnAfterResetContactContactMapping(IntegrationTableMappingName: Code[20])
var
    CDSContact: Record "CRM Contact";
    Contact: Record Contact;
    IntegrationFieldMapping: Record "Integration Field Mapping";
begin
    InsertIntegrationFieldMapping(
        IntegrationTableMappingName,
        Contact.FieldNo("Industry"),
        CDSContact.FieldNo(cr07b_Industry),
        IntegrationFieldMapping.Direction::Bidirectional,
        '', true, false);
end;
```

After we publish the extension we can update the mappings by running the **CDS Connection Setup** page and choosing **Use Default Synchronization Setup**.

See Also

Overview
Setting Up User Accounts for Integrating with Microsoft Dataverse
Set Up a Connection to Microsoft Dataverse Synchronizing Business Central and Microsoft Dataverse
Mapping the Tables and Fields to Synchronize
Manually Synchronize Table Mappings
Schedule a Synchronization

AL Table Proxy Generator

2/17/2021 • 2 minutes to read • Edit Online

NOTE

Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

The **AL Table Proxy Generator** tool can be used to generate one or more tables for integration with Microsoft Dataverse. When one or more tables are present in Microsoft Dataverse, but not in Dynamics 365 Business Central, the tool can be run to generate integration or proxy tables for the specified table or tables.

An integration or proxy table is a table that represents a table in Microsoft Dataverse. The integration table includes fields that correspond to columns in the Microsoft Dataverse table. The integration table acts as a link or connector between the Business Central table and the Microsoft Dataverse table.

The **AL Table Proxy Generator** tool is available with the **AL Language** extension. Look for the **altpgen.exe** tool in the equivalent folder of `c:\users\<username>\.vscode\extensions\<al extension version>\bin`.

Generating proxy tables

1. Start Windows PowerShell as an administrator.
2. From the command prompt, write `.\altpgen.exe` followed by the parameters as described below.

```
-Project
-PackageCachePath
-ServiceURI
-Entities
-BaseId
-[TableType]
```

3. The table or tables are generated in the folder of the specified AL project.

Parameters

PARAMETER	DESCRIPTION
Project	The AL project folder to create the table(s) in.
PackageCachePath	The AL project cache folder for symbols. **Note:** It is important that the latest symbols have been downloaded because these are used for comparison when the tool runs.
ServiceURI	The server URL for Microsoft Dataverse. For example, `https://tenant.crm.dynamics.com`.

PARAMETER	DESCRIPTION
Entities	The table(s) to create in AL. If multiple, this must be specified as a comma-separated list. **Note:** It is important that all related tables are specified too. Related tables are, for example, used for lookups and if the related tables are not found, a lookup will no longer be working. For more information, see the section Specifying tables.
BaseId	The assigned starting ID for the generated new table(s) in AL.
TableType	The table type for the table(s) in AL. The options are `CDS` and `CRM`. **Note:** If unspecified, the system looks both for `CDS` and `CRM` tables .

Specifying tables

The `Entitites` parameter specifies the logical names of the table(s) to create in AL. To know which ones to specify you need to check the *main* table relationships in Microsoft Dataverse. For more information, see Table relationships overview. You specify all tables that you want created, including the related tables, in the `Entities` parameter separated by commas.

Related tables

An example could be, that you want to generate an AL proxy table for the **CDS Worker Address** (cdm_workeraddress).

If you run the altpgen tool and only specify `cdm_workeraddress`, the tool will not generate the `Worker` lookup field, because no related table `Worker` is specified.

If you, in the `Entities` parameter specify `cdm_workeraddress, cdm_worker`, the `Worker` lookup field will be generated. Furthermore, if your *symbols contain* the `cdm_worker` table definition, the `Worker` table will not be created as it's already in your symbols. If your *symbols do not contain* the `cdm_worker` table, the `Worker` table will be created together with the `Worker Address` table.

Creating a new integration table

The following example starts the process for creating a new integration table in the specified AL project. When complete, the output path contains the **Worker.al** file that contains the description of the **50000 CDS Worker** integration table. This table is set to the table type **CDS**.

```
.\altpgen -project:"C:\myprojectpath" -packagecachepath:"C:\mypackagepath" -
serviceuri:"https://tenant.crm.dynamics.com" -entities:cdm_worker,cdm_workeraddress -baseid:50000 -
tabletype:CDS
```

See Also

Custom Integration with Microsoft Dataverse

Microsoft Power Platform Integration with Business Central

2/17/2021 • 2 minutes to read • Edit Online

PREVIEW: This feature is in preview in Business Central 2020 release wave 2

> **NOTE**
>
> Effective November 2020:
>
> - Common Data Service has been renamed to Microsoft Dataverse. Learn more
> - Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

> **NOTE**
>
> The **Business Central Virtual Entity (Preview)** app available on AppSource is being updated to reflect new terminology with an upcoming release. This also applies to that terminology used in Business Central.

> **IMPORTANT**
>
> This functionality requires version 17 for Business Central, while service update 189 is required for Microsoft Dataverse. The release information for Microsoft Dataverse is published on the latest version availability page.

Business Central Virtual table, which is published on AppSource, is a virtual data source in Microsoft Dataverse, and enables create, read, update, delete (CRUD) operations from Microsoft Dataverse and Microsoft Power Platform. By definition, the data for virtual tables does not reside in Microsoft Dataverse. Instead, it continues to reside in Business Central. To enable operations on Business Central tables in Microsoft Dataverse, tables must be made available as virtual tables in Microsoft Dataverse.

Prerequisite reading

To work with Business Central virtual tables, you must understand how Microsoft Dataverse and virtual tables work. Therefore, the following documentation is a prerequisite:

- What is Microsoft Dataverse?
- Table overview
- Table relationships overview
- Create and edit virtual tables that contain data from an external data source
- Overview of creating apps in Power Apps

Virtual tables for Business Central

Open Data Protocol (OData) APIs exposed through API Pages in Business Central can be consumed in Microsoft Dataverse and virtual tables can be generated. Virtual tables in Microsoft Dataverse acts as regular tables and therefore also in Power Platform. Makers can now build experiences in customer engagement apps with data directly from Business Central with full CRUD capability and without copying to Microsoft Dataverse, and

leverage all the logic already residing in Business Central.

Using Custom APIs as basis for virtual tables

Since the virtual tables depend on APIs exposed on Business Central, custom APIs can also used for generating virtual tables. For more information, see Developing a Custom API.

Known limitations

There are known limitations with Business Central virtual tables including:

- Flows are not triggered for virtual tables. Currently, Business Central has no way to signal Microsoft Dataverse about data change events.
- Virtual tables cannot be used in Charts. Microsoft Dataverses does not support virtual tables being using in Charts.
- Relations between native and virtual tables. This is currently a limitation of the **Preview** version of Business Central virtual tables solution.
- Virtual tables cannot be customized on Microsoft Dataverse, for example, adding new columns. All modifications to virtual tables must happen in the API exposed on Business Central. But custom APIs can be developed and consumed as virtual tables.
- Attachment and Images/Pictures are not supported for virtual tables.
- BLOB to multiline support is not supported in the preview.
- Advanced search has some limitations. Each query designed translates to an OData query against Business Central.
  - The following predicates are not supported: **Does Not Equal**, **Does Not Contain**, **Does Not Begin With**, **Does Not End With**, **Does Not Contain Data**, and **Contains Data**.
  - Combining **And** and **Or** groups across columns.
  - Filtering on related tables.
- PowerApp Portals are not supported in current preview.

See Also

Table Modeling
Application Lifecycle Management for Solutions that use Virtual tables
Business Central and Microsoft Dataverse Admin Reference
FAQ
Developing a Custom API

Table Modeling

2/17/2021 • 11 minutes to read • Edit Online

PREVIEW: This feature is in preview in Business Central 2020 release wave 2

> **NOTE**
>
> Effective November 2020:
>
> - Common Data Service has been renamed to Microsoft Dataverse. Learn more
> - Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

> **NOTE**
>
> The **Business Central Virtual Entity (Preview)** app available on AppSource is being updated to reflect new terminology with an upcoming release. This also applies to that terminology used in Business Central.

> **IMPORTANT**
>
> This functionality requires version 17 for Business Central, while service update 189 is required for Microsoft Dataverse. The release information for Microsoft Dataverse is published on the latest version availability page.

Building an app requires capabilities to perform relational modeling between tables that are being used in the app. In the context of virtual tables, there will be scenarios where virtual tables and native tables in Microsoft Dataverse must work together to enable the desired user experience. This topic explains concepts of relational modeling that can be implemented using virtual tables for Business Central.

Generating virtual tables

By default, virtual tables for Business Central do not exist in Microsoft Dataverse. A user must query the catalog table to view the tables that are available in the linked instance of Business Central. From the catalog, the user can select one or more tables, and then request that Microsoft Dataverse generates the virtual tables. This procedure is explained in later sections.

Table fields

When a virtual table is generated for a Business Central table, the system tries to create each field in the Business Central table in the corresponding virtual table in Microsoft Dataverse. In an ideal case, the total number of fields will be the same in both tables, unless there is a mismatch in supported data types between Business Central and Microsoft Dataverse. For data types that are supported, the column properties in Microsoft Dataverse are set based on the properties in Business Central.

The rest of this section describes supported and unsupported data types. For more information about columns in Microsoft Dataverse, see Columns overview.

DATA TYPE IN BUSINESS CENTRAL	MODELED DATA TYPE IN MICROSOFT DATAVERSE
Real	Decimal For information about the possible mismatch, see the next table.
Long	Decimal, where the precision equals 0 (zero)
Int	Integer
String (non-memo), String (memo)	String – single line of text, String – multiple lines of text
UtcDateTime	DateTime (DateTimeFormat.DateAndTime, DateTimeBehavior.TimeZoneIndependent) An empty date (January 1, 1900) in Business Central is surfaced as a null value in Microsoft Dataverse.
Date	DateTime - (DateTimeFormat.DateOnly, DateTimeBehavior.TimeZoneIndependent) An empty date (January 1, 1900) in Business Central is surfaced as an empty value in Microsoft Dataverse.
Enum	Picklist Business Central enumerations (enums) are generated as global OptionSets in Microsoft Dataverse. Matching between the systems is done by using the **External Name** property of values. Enum integer values in Microsoft Dataverse are not guaranteed to be stable between the systems. Therefore, you should not rely on them, especially in the case of extensible enums in Business Central, because these enums do not have a stable ID either. OptionSet metadata is updated when an table that uses the OptionSet is updated.

Fields of the *real* and *long* data types in Business Central are modeled as the *decimal* data type in Microsoft Dataverse. Because of the mismatch in precision and scale between the two data types, the following behavior must be considered.

USE CASE	RESULTING BEHAVIOR
Microsoft Dataverse has higher precision.	This use case should never occur unless the metadata is out of sync.
Business Central has higher precision.	During a read operation, the value is rounded to the closest precision value in Microsoft Dataverse. If the value is edited in Microsoft Dataverse, it is rounded to the closest precision value in Business Central. During a write operation, the value that is specified in Microsoft Dataverse is written, because Business Central supports higher precision.
Microsoft Dataverse has higher scale.	Not applicable.

USE CASE	RESULTING BEHAVIOR
Business Central has higher scale.	Microsoft Dataverse shows the Business Central value, even if it exceeds 100 billion. However, there will be a loss of precision. For example, 987,654,100,000,000,000 is shown in Microsoft Dataverse as "987,654,099,999,999,900". If the value of this column is edited in Microsoft Dataverse, Microsoft Dataverse validation throws an error that the value exceeds the maximum value before that value is sent to Business Central.

The following data types in Business Central are not supported in Microsoft Dataverse. Fields of these data types in Business Central tables will not be made available in the corresponding virtual tables in Microsoft Dataverse. If fields of these data types are used as parameters in Open Data Protocol (OData) actions, those actions will not be available for use in the corresponding virtual tables. For more information about OData actions, see the OData actions section later in this topic.

Table key - primary key

In Business Central, tables use the SystemId (GUID) as the primary key, which uniquely identifies a record in a Business Central. In Microsoft Dataverse, the SystemId exposed by the table is used as the primary key.

Primary column/field

In Microsoft Dataverse, each table must have a primary column. This column must be a single column of the string type. The primary column is used in Microsoft Dataverse in the following scenarios:

- The default views that are created for an table include the primary column.
- The quick view form for an table includes the primary column.
- A lookup to another table is added to a page and shows the data from the primary column.

The primary field for a virtual table for Business Central is designed to use **displayname** field on table if present. If this field is not present the first string field is chosen as the primary field.

Relations

> **IMPORTANT**
> A write transaction that spans a virtual table and a native table is not supported. Using this form of transaction is not recommended, as there is no way to ensure consistency.

Relations in Business Central tables are modeled as one-to-many (1:n) or many-to-one (n:1) relations. These relations are modeled as relationships in the virtual table in Microsoft Dataverse. Note that many-to-many (n:n) relations are not supported in Business Central.

For example, in Business Central, if table A has a foreign key to table B, this relation will be modeled as an n:1 relationship in virtual table table A in Microsoft Dataverse. The schema name of this relationship in Microsoft Dataverse uses the naming convention **dyn365bc_<source table name>_<target_table name>**. This naming convention has a maximum string length of 120 characters. Any relation where the schema name will produce a name that exceeds 120 characters will not be generated in the virtual table in Microsoft Dataverse. It is required that the foreign key is a SystemId (GUID). If the foreign key is of a different type then the relation will not be generated.

The external name of this relationship uses the naming convention **<relation name>**. The external name is used to determine the relation in Business Central when the query that is sent to Business Central is built.

When a relationship is generated for a virtual table in Microsoft Dataverse, a new column of the lookup type is also added to the source table. In the preceding example, when the relationship is created, a new lookup column that uses the naming convention **dyn365bc_<target_table>_id** is added to source table table A. Because there can be several relations in an table in Business Central, the same number of lookup fields (one per related table) will be created in the source virtual table. When this lookup field is added to a page or a view, it will show the primary field value from the related table.

A relationship in the virtual table in Microsoft Dataverse will be generated only if the related table in the relation already exists as a virtual table in Microsoft Dataverse. In the preceding example, if table B does not exist as a virtual table in Microsoft Dataverse, the relation to table B will not be created in table A when table A is generated as a virtual table. This relation will be added to table A only when table B is generated as a virtual table. Therefore, when a virtual table is generated for Business Central, validations are done to ensure that only relationships that can be complete and functional are generated in the virtual table that is being generated.

In summary, a relationship to another Business Central virtual table might not exist in the virtual table for either of the following reasons:

- The Business Central table that is participating in the relationship does not exist as a virtual table.
- The foreign key is not SystemId (GUID).
- The length of the name of the relationship exceeds 120 characters.

> **NOTE**
> If an error is encountered when any part of a Business Central virtual table is generated in Microsoft Dataverse, the virtual table will not be created at all. If relationships do not exist for either of the preceding reasons, the situation is not considered an error.

Native table–to–native table relationships

Native table–to–native table relationships are the standard Microsoft Dataverse functionality, where relationships are resolved by using the GUID of the related table. (This GUID is the table key.) The GUID identifies the unique table record in the related table.

Virtual table–to–virtual table relationships

The relationships between two Business Central virtual tables are driven by the relation metadata in the Business Central tables. As was explained earlier, these relations are generated as relationships in Microsoft Dataverse when the virtual table is generated. As in the behavior for native tables in Microsoft Dataverse, these relationships use the GUID to identify the unique record of the table in Business Central. Semantically, the GUID on the Business Central virtual table behaves like the GUID on the native Microsoft Dataverse table. For information about the implementation of the GUID in Business Central virtual tables, see the table key/primary key section earlier in this topic.

In the preceding example, the GUID of the related table is the table key of table B and will be used to build queries to identify a record in Business Central. The relation that table A has to table B will be used.

Therefore, in effect, the table name is the only information that is used in a relation that comes from Business Central. The table name gives access to the primary field in the related table, so that it can be shown in the lookup. It also gives access to the GUID of the related table, so that it can be used in other queries, as was explained earlier. The actual field that the relation is built on in the Business Central table is not used at all.

Virtual table–to–native table relationship

The relationship between Business Central virtual table and native table is not supported in the preview version of Business Central virtual table solution.

Enums

Enums in Business Central are modeled as OptionSets in Microsoft Dataverse. When a virtual table for Business Central is generated, the required enums are generated as OptionSets. If an OptionSet already exists, it is used instead.

Company

If Business Central has multiple companies, the default company must be selected. This can be done either on a Business Central virtual table configuration page or on a User table page.

Furthermore, every virtual table for a Business Central table have a relationship to the cdm_company table in Microsoft Dataverse. The cdm_company table is a native table in Microsoft Dataverse and is part of the Dynamics365Company solution. As always, when a relationship is created, a lookup column is also created in the virtual table for the related table (cdm_company in this case). This lookup column is named **Company**, and it must be used to provide an optimal user experience where users can select a value in a list or go to the details of the related record. A column that is named **Company Code** is also added in the virtual table. This column must be used in programming. A Virtual Table can only interact with one Company at a time. Connection to other Companies can be made in either the connection setup page or overridden on the individual user.

OData actions

OData actions in the Business Central tables are made available as custom actions in Microsoft Dataverse. For more information about custom actions and what they enable in Microsoft Dataverse, see Custom actions.

OData actions generated for Business Central have only one parameter which is the table. There is no output parameter.

Labels and localization

Labels that are defined on metadata, such as table names and field names in Business Central, are retrieved when virtual tables are generated in Microsoft Dataverse. The labels are retrieved using an API on Business Central called **tableDefinitions**. This API is available on every API route, and will serve translations and other table metadata, not suited for OData `$metadata`. But with both the tableDefinition and `$metadata` Microsoft Dataverse has all it needs to generate localized virtual tables.

Any runtime labels are returned in the language of the current user context. In other words, they are returned in the language that is specified on that user's UserInfo record in Business Central. This behavior also applies to error messages.

Error handling

Business Central create, read, update, and delete (CRUD) business logic on tables and backing tables is run when it is called through the virtual table in Microsoft Dataverse. If any exception is thrown on the Business Central side, the last message in the error log is returned to Microsoft Dataverse and is thrown as an InvalidPluginExecutionException exception that contains the message from Business Central. Because the Business Central code runs in the context of the user, the language of the error message is based on the language that is specified on the UserInfo record in Business Central. If any messages that are written to the info log in Business Central do not result in an exception, they are not shown in Microsoft Dataverse.

Calculated/unmapped fields

Calculated and unmapped fields in Business Central tables are also available in the corresponding virtual tables in Microsoft Dataverse.

See Also

Microsoft Power Platform Integration with Business Central
Application Lifecycle Management for Solutions that use Virtual tables
Business Central and Microsoft Dataverse Admin Reference
FAQ

Application Lifecycle Management for Solutions that use Virtual Tables

2/17/2021 • 5 minutes to read • Edit Online

PREVIEW: This feature is in preview in Business Central 2020 release wave 2

NOTE

Effective November 2020:

- Common Data Service has been renamed to Microsoft Dataverse. Learn more
- Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

NOTE

The **Business Central Virtual Entity (Preview)** app available on AppSource is being updated to reflect new terminology with an upcoming release. This also applies to that terminology used in Business Central.

IMPORTANT

This functionality requires version17 for Business Central, while service update 189 is required for Microsoft Dataverse. The release information for Microsoft Dataverse is published on the latest version availability page.

The application lifecycle for an end-to-end solution using Business Central virtual tables will encompass both Business Central as well as Microsoft Dataverse.

Solution management

Virtual tables for Business Central do not exist in Microsoft Dataverse until they are created. Virtual tables must be created inside a solution. The **MicrosoftBusinessCentralERPVE** solution is used for this purpose. This solution will contain all the virtual tables that are created from an instance of Business Central.

MicrosoftBusinessCentralERPVE is a managed solution. By definition, a managed solution cannot be modified after it has been generated. However, **MicrosoftBusinessCentralERPVE** is a managed solution that grants privileges to update the components (that is, virtual tables) that are inside it. Therefore, new virtual tables can be added to the solution as they are created, and existing virtual tables can be updated as required. Nevertheless, the privileges to modify the managed solution are available only to the platform itself. Users cannot make changes directly to the solution.

Because **MicrosoftBusinessCentralERPVE** is a managed solution, solutions from customers, partners, and independent software vendors (ISVs) can take a dependency on it. This capability allows for consistent application lifecycle management (ALM) for solutions that use and depend on the virtual tables for Business Central.

When a solution that depends on **MicrosoftBusinessCentralERPVE** is exported, placeholders for the virtual tables that are used in the solution are added in the exported solution. When that solution is imported into another Microsoft Dataverse environment, the import process also generates the dependent Business Central virtual tables in the **MicrosoftBusinessCentralERPVE** solution for the Business Central instance that is

connected to the Microsoft Dataverse environment. Therefore, **MicrosoftBusinessCentralERPVE** must already exist before a solution that depends on it is imported. Otherwise, an error message is shown. Additionally, if a dependent table is not available in the Business Central instance, the virtual table for that table will not be generated. Virtual tables are generated only for tables that are available.

The following list describes other solutions that Business Central virtual tables require to work, and that must be available in the Microsoft Dataverse environment:

- **MicrosoftBusinessCentralERPVE** - Solution that contains the generated virtual tables.
- **MicrosoftBusinessCentralERPCatalog** – This solution provides a catalog of the available tables in a Business Central instance. It also provides the connection that is used to set up a configuration. For more information, see the later sections of this topic.
- **MicrosoftBusinessCentralVESupport** – This solution provides the virtual table provider for Business Central. The provider can communicate with Business Central and Microsoft Dataverse. For more information, see the next section.
- **Dynamics365Company** – This solution adds the Company table, which is referenced by all Business Central tables that have a **PrimaryCompanyContext** metadata value.

All these solutions must be present in an environment. Otherwise, virtual tables will not work with Business Central apps. These solutions are packaged together to allow for easier portability across environments.

Managing tables from multiple environments

The **MicrosoftBusinessCentralVESupport** solution consists of the **msdyn_ businesscentralvirtualtable** table. This table represents the virtual table data source for Business Central that captures connection setup information. Each record in this table represents a connection to a Business Central instance.

A catalog is used to list all the tables in a Business Central instance that are available for virtualization in Microsoft Dataverse (in other words, all the tables in Business Central that are enabled for Open Data Protocol [OData]). The catalog is part of the default **MicrosoftBusinessCentralERPCatalog** solution and is applicable to a Business Central instance.

Note that each Microsoft Dataverse environment must point to only one Business Central instance at any time, and each Business Central environment must point to only one Microsoft Dataverse environment. Therefore, there should be only one record in the **msdyn_businesscentralvirtualtable** table.

The **mserp_businesscentralvirtualtable** table that represents the catalog can be queried to list the tables in a Business Central instance. Because this table is a virtual table, the catalog is never persisted in Microsoft Dataverse.

Notice that the name of the catalog table has the "mserp_" prefix. This prefix identifies the tables in the catalog as Business Central tables. The same prefix is also added to the system names of the virtual tables that are generated for Business Central in the **MicrosoftBusinessCentralERPVE** solution. Therefore, the maker can distinguish Business Central virtual tables from other tables. The prefix is set in the managed solution and cannot be changed.

Managing tables from multiple ISV solutions

One or more ISV solutions will take a dependency on the **MicrosoftBusinessCentralERPCatalog** solution to use virtual tables for Business Central. Because custom tables in Business Central use the same catalog as out-of-box tables in Business Central, the virtual tables for custom Business Central tables will also be generated in the **MicrosoftBusinessCentralERPVE** solution.

The established guidelines and ALM for table development in Business Central ensure that there are no conflicting table names across ISV solutions. Therefore, no conflicts of this type can occur when virtual tables are generated in Microsoft Dataverse for custom Business Central tables from multiple ISV solutions. All virtual

tables for Business Central tables, including custom tables, will have the "mserp_" prefix that was mentioned earlier.

Managing a Business Central instance in a Microsoft Dataverse environment for virtual tables

One Business Central instance must be linked to a Microsoft Dataverse environment for virtual tables. The connection setup information that is required is captured in a virtual table data source for Business Central. This data source is included in the **MicrosoftBusinessCentralERPCatalog** solution.

See Also

Microsoft Power Platform Integration with Business Central
Table Modeling
Business Central and Microsoft Dataverse Admin Reference
FAQ

Business Central Virtual Table for Microsoft Dataverse Admin Reference

2/17/2021 • 4 minutes to read • Edit Online

PREVIEW: This feature is in preview in Business Central 2020 release wave 2

> **NOTE**
>
> Effective November 2020:
>
> - Common Data Service has been renamed to Microsoft Dataverse. Learn more
> - Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

> **NOTE**
>
> The **Business Central Virtual Entity (Preview)** app available on AppSource is being updated to reflect new terminology with an upcoming release. This also applies to that terminology used in Business Central.

> **IMPORTANT**
>
> This functionality requires version 17 of Business Central and service update 189 for Microsoft Dataverse. The release information for Microsoft Dataverse is published on the latest version availability page.

This topic provides step-by-step instructions on how to set up and configure virtual tables for Business Central in Microsoft Dataverse.

Getting the solution

First get the Business Central Virtual Entity solution from AppSource.

The following solutions are installed in Microsoft Dataverse once the Business Central virtual tables is installed from AppSource.

- **Dynamics365Company** - This adds the **cdm_company** table, which is referenced by all Business Central virtual tables. All communication to Business Central requires the company ID in the request.
- **MicrosoftBusinessCentralVESupport** - This provides the core support for the Business Central virtual table feature.
- **MicrosoftBusinessCentralERPCatalog** - This provides a list of available Business Central.
- **MicrosoftBusinessCentralVEAnchor** - This serves as a container, holding information needed for AppSource.
- **MicrosoftBusinessCentralERPVE** - Virtual tables generated for Business Central will be contained in this solution. tables are added at runtime once they are made visible.

Authentication and authorization

After the solutions are installed in the Microsoft Dataverse environment, connection can be set up to a Business Central environment. *Both environments have to be in the same tenant.*

The next step in the process is to provide Microsoft Dataverse with the Business Central environment and company to connect to. The following steps walk through this part of the process.

0. In Business Central, go to the page 'AAD Applications' and toggle the app 'Dynamics 365 Business Central for Virtual tables' to **Enabled**. This will allow Microsoft Dataverse to communicate with Business Central.

1. In Microsoft Dataverse, go to the table **Business Central Virtual Data Source Configuration**.
2. Select and edit the data source named "Business Central".
3. On the Business Central Virtual Data Source Configuration, set the environment name. Unless changed, Business Central tenants will have an default environment called 'production'.
4. Save changes before setting **Default Company**. Else, company cannot be set in the next step.
5. Set the **Default Company** value.
6. Save the changes.

Making virtual tables visible

Due to the large number of OData enabled tables available in Business Central, by default, the tables are not available as virtual tables in Microsoft Dataverse. The following steps allow for enabling tables to be virtual, as needed.

1. In Microsoft Dataverse, go to **Data -> tables** and search for *Available Business Central table*. Make sure to search for All and not just Default.
2. Choose **Data** in the horizontal menu to view the available data.
3. Locate and edit the table that you want to enable.
4. Set **Visible** to **Checked** and save. This will generate the virtual table, so that it will appear in all of the appropriate menus, and in advanced find dialog box.

Refreshing virtual table metadata

The virtual table metadata can be force-refreshed when it is expected for the table metadata in Business Central to have changed. This can be done by setting **Refresh** to **Checked** and saving. This will sync the latest table definition from Business Central to Microsoft Dataverse and update the virtual table.

Referencing virtual tables

The virtual tables are all generated in the **MicrosoftBusinessCentralERPVE** solution. That means the items in the solution change as you make tables visible/hidden, but it is still a managed solution that you can take dependency on. The standard ALM flow would be to just take a standard reference to a virtual table from this solution with the **Add existing** option in the ISV solution. It will then show as a missing dependency of the solution and be checked at solution import time. During import if a specified virtual table does not yet exist, it would automatically be made visible without needing additional work.

To consume virtual tables:

1. Create a separate solution as usual in Microsoft Dataverse, which will contain the consuming logic.
2. Select **tables > Add Existing**. Select the virtual table that you want to reference from the list.
3. When prompted to select assets to add, select any forms, views, or other elements that you want to customize, then select **Finish**.

From the development tooling, existing elements such as forms can be modified for the virtual table. Additionally, new forms, views, and other elements can also be added.

When the solution is exported, it will contain hard dependencies on the virtual table generated in the **MicrosoftBusinessCentralVE** solution.

See Also

Microsoft Power Platform Integration with Business Central
Table Modeling
Application Lifecycle Management for Solutions that use Virtual tables
FAQ

Business Central Virtual Tables FAQ

2/17/2021 • 2 minutes to read • Edit Online

PREVIEW: This feature is in preview in Business Central 2020 release wave 2

> **NOTE**
>
> Effective November 2020:
>
> - Common Data Service has been renamed to Microsoft Dataverse. Learn more
> - Some terminology in Microsoft Dataverse has been updated. For example, *entity* is now *table* and *field* is now *column*. Learn more

> **NOTE**
>
> The **Business Central Virtual Entity (Preview)** app available on AppSource is being updated to reflect new terminology with an upcoming release. This also applies to that terminology used in Business Central.

> **IMPORTANT**
>
> This functionality requires version 17 of Business Central, while service update 189 is required for Microsoft Dataverse. The release information for Microsoft Dataverse is published on the latest version availability page.

This topic is a collection of frequently asked questions about Business Central virtual tables.

What version of Business Central do I need?

Version 17.0 of Business Central is needed. Version 17 contains v2.0 of the standard APIs and improvements to the OData stack that enable APIs to be exposed and consumed as virtual tables.

Can a solution from an independent software vendor (ISV) take a dependency on virtual tables? What does the application lifecycle management (ALM) look like?

Yes. The virtual tables are all generated in the **MicrosoftBusinessCentralERPVE** solution, which is API-managed. In other words, the items in the solution change as you make tables visible or hidden, but the solution is still a managed solution that you can take dependency on. The standard ALM flow just takes a standard reference to a virtual table from this solution with the **Add existing** option in the ISV solution. Missing dependency of the solution will be checked when the solution is imported and during import, if a specified virtual table does not yet exist, the virtual table is automatically made visible.

Which tables from Business Central do users see in the catalog in Microsoft Dataverse?

Generally, users see all tables exposed as API pages with the exception of Microsoft legacy API pages: beta and v1.0.

Do all Microsoft Power Platform users have to be users in Business Central?

Any user of Microsoft Power Platform who tries to access Business Central data through a virtual table must also exist as a user in Business Central. Therefore, technically, not *all* users have to be users in Business Central. Only those users who access Business Central data through virtual tables must be users in Business Central.

Where do I find the catalog table?

In the **Advanced find** window, the table is named **Available Business Central tables**.

Can I change the prefix for the virtual tables?

No. All Business Central virtual tables should be generated in the **MicrosoftBusinessCentralERPVE** solution, and they all have the "dyn365bc_" prefix. This prefix will not be changed.

How can I show, in the same grid, data from multiple virtual tables that are joined to a physical table record in Microsoft Dataverse?

This approach is not currently possible in Microsoft Dataverse.

If I want a default value to be entered in a column during pre-create, will an initValue on the data table then work?

Yes. Here is the order of calls:

1. Microsoft Dataverse sends a create or update message.
2. All the existing logic on the Business Central table and backing tables is invoked. This logic includes default value entry that might change values.
3. Microsoft Dataverse sends another Retrieve (single) message to get the latest copy of the data, including any columns that default values were entered for.

See Also

Microsoft Power Platform Integration with Business Central
Table Modeling
Business Central and Microsoft Dataverse Admin Reference
Application Lifecycle Management for Solutions that use Virtual tables

Developing Pages and Tables for Microsoft Teams Integration

2/17/2021 • 3 minutes to read • Edit Online

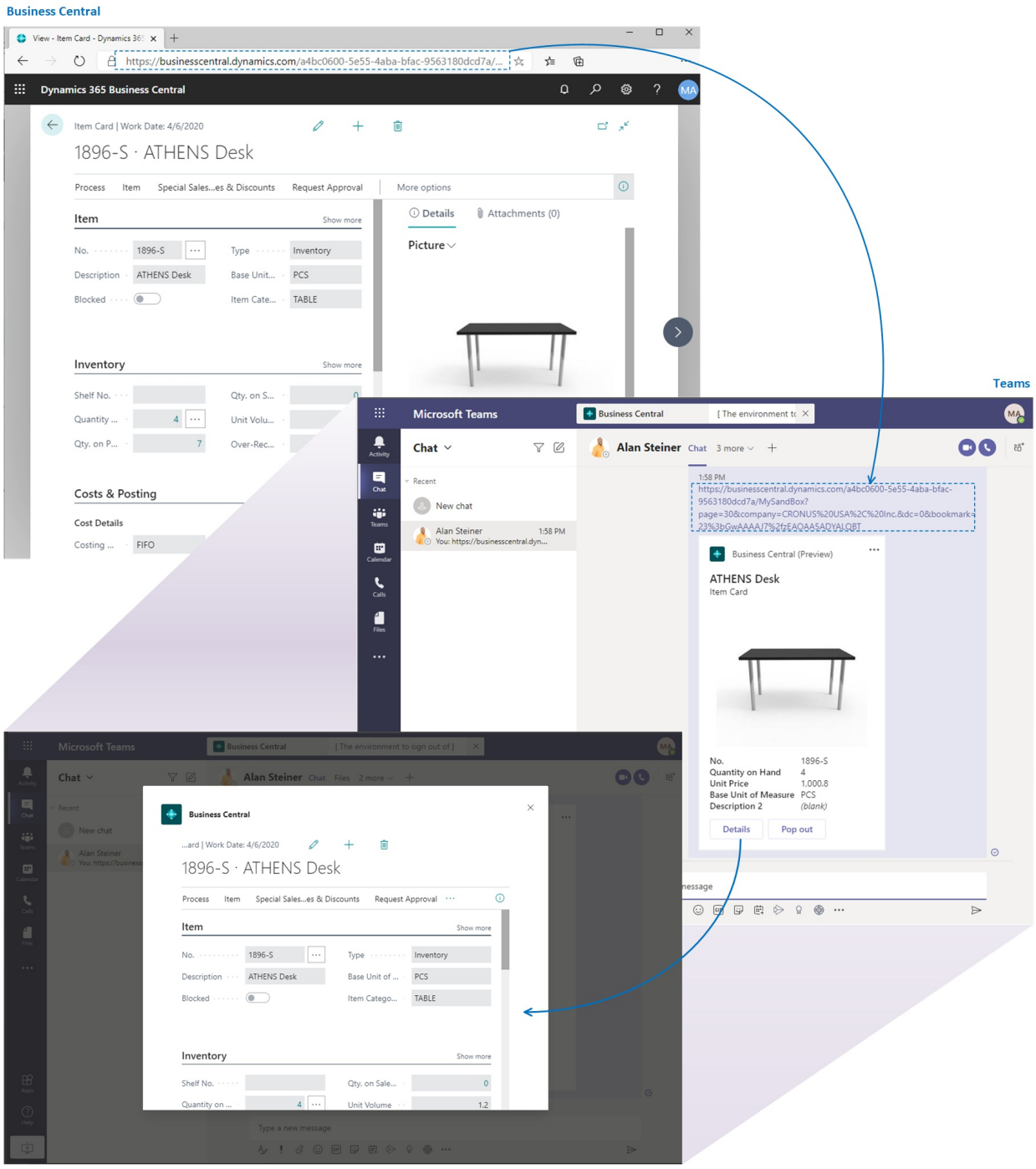
APPLIES TO: Business Central online

Microsoft Teams is a Microsoft 365 product that lets you connect with others, collaborate seamlessly and simplify work. Business Central offers an app that connects Teams to your business data in Business Central so users can quickly share details across team members and respond faster to inquiries. In this article, you'll learn how you can develop pages and tables so that data appears in Teams as you want.

Introduction to the Business Central app for Teams

In short, the app lets users:

- Copy a link to any Business Central record and paste it into Teams conversation to share with your coworkers. In Teams, the link will expand that into a compact card that displays a defined set of fields about the record.
- From the card in teams, users can select a button to view more details about the record. The card opens into window inside Teams that includes FactBoxes and other embedded content, such as charts. With the proper permissions, anyone in the conversation can edit fields, start workflows, and take action from the window - without having to switch from Teams.



Managing, Installing, and Using the Business Central App for Teams

For more details about installing and using the Business Central app in Teams from an admin and user perspective, see Business Central and Microsoft Teams Integration.

Development overview

Users can choose which links to records are pasted into Teams. The pasted links are then displayed as cards. As a developer, you can choose which fields are shown on the card, and the look and behavior of the card details.

About cards

The Business Central app for Teams is designed to work with entity pages that represent a single record. More specifically, it's designed for Card, Document, or ListPlus pages. Teams is ready to work with links to these page types and the underlying data in the source tables. In most cases, displaying a card in Teams requires no additional development effort. But if a card doesn't display the fields and data you want, you have a couple options in AL code for making changes:

- Use a field group control on the source table

  This method primarily involves using the `Brick` field group on the page's source table.

- Use Teams-specific events

  There are currently two events that you can code against to change the fields that display in the card at runtime. We recommend that you use these events as an additive measure.

In most cases, we recommend you set the `Brick` field group instead of using events to define card content. Using the `Brick` field group ensures a consistent experience across the Business Central Web client, mobile devices, and Teams. If a table doesn't include a `Brick` field group, you should add one, then use the events to change the fields as needed.

For more information about these two development options, see Extending Teams Cards.

About card details

Users can select the **Details** button on a Teams card to drill into the details of the card. This button opens the Business Central Web client in a window that's embedded inside Teams. The window displays the target page object. The page includes any extensions to the page and its source table, role customizations, and user personalization. The experience is nearly identical to that of displaying the same record in the Business Central Web client, except with some minor client differences.

You can modify or extend the layout and behavior of the page, like you would any page. The changes you make will affect the page in all Business Central clients, not just in Teams. You can't implement functionality that is available only in Teams but not other Business Central clients. For more information about implementing pages, see Pages Overview.

Preparing your environment for development and testing

Developing for Teams integration is similar to other development activities in Business Central. It requires Visual Studio Code, installed with the AL language extension, and knowledge of the AL language. However, because the Business Central app for Teams only supports Business Central online, you must deploy your extensions to an online sandbox to test the implementation. For more information about sandboxes, see Sandbox Environments for Dynamics 365 Business Central Development.

See Also

FAQ for Teams Integration
Field Groups
Designing List Pages
Working With Media on Records

Extending Teams Cards

2/17/2021 • 10 minutes to read • Edit Online

APPLIES TO: Business Central online

In this article, you'll learn how to customize the fields that display on a Teams card. You customize the fields by using a field groups control on the source table or using events. You can also combine these two methods to achieve the results that you want.

Extend Teams cards by using field groups

This section explains how to customize the fields that display on Teams card by setting metadata in Business Central page and table objects.

Understanding the card layout in Teams

The main metadata elements for displaying fields in Teams are the `Brick` field group, together with the page and field captions. However, with the lack of a `Brick` field group on a table, the `Dropdown` field group and primary keys will play a role in determining the fields that display. This section describes how the app selects and arranges fields on the card in Teams. Use this information to help define your `Brick` field group.

What fields are displayed

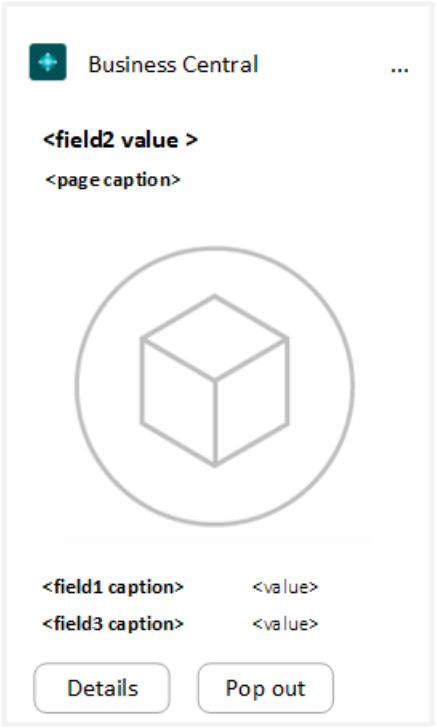
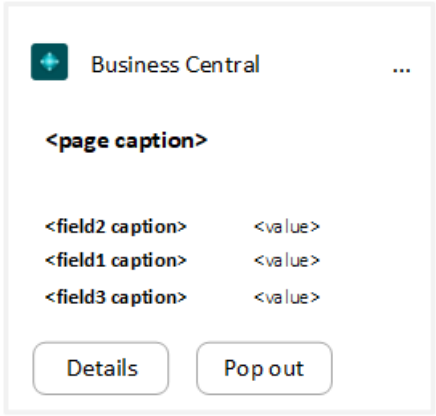
Although it's recommended, a `Brick` field group isn't required to display a card in Teams. Lacking a `Brick` field group, the app will fall back to use either `DropDown` field group or the primary key fields of on the source table, according to the following flow:

1. If a `Brick` field group exists, the app uses the fields specified by the `Brick` field group.
2. If a `Brick` field group doesn't exist, the app uses fields that are defined in `Dropdown` field group.
3. If `Dropdown` field group doesn't exist, then the primary key fields and fields called Name or Description in the table are used.

How fields are arranged

The app will arrange fields on cards in Teams based on one of two templates, depending on whether a media field is included in the `Brick` field group. Except media data type fields, the order of fields on the card is determined by the field order in the `Brick` field group. You can't reposition fields, add new controls like actionable buttons, or change the overall style of a card.

The two templates and how they order fields are explained in the following table.

WITH MEDIA	WITHOUT MEDIA
	
The card is laid out from top to bottom as follows: 1. The second field defined in `Brick` field group appears first in a large font. 2. The page's caption property value 3. The media thumbnail 4. The remaining brick fields in the order they're defined. Guidelines: • For the second field, it's a good idea to use a field that best identifies the record, like a name or description field. • Don't place a media field as the second field in the `Brick` field group. It doesn't matter where you place the field otherwise - it will appear after the page caption. • If there's more than one media field in the `Brick` field group, only the first one listed is used. • BLOB data type fields aren't supported.	The card is laid out from top to bottom as follows: 1. The page's caption property value 2. The second field defined in `Brick` field group. 3. The remaining brick fields in the order they're defined. Guidelines: • BLOB data type fields aren't supported.

Using the Brick field group

By default, the fields that display on a card in Teams are based on the `Brick` field group that's defined on a page's source table. The `Brick` field group is the same element that determines which fields display with records in list pages viewed as tiles in the client. So you'll have to keep it in mind when designing for Teams.

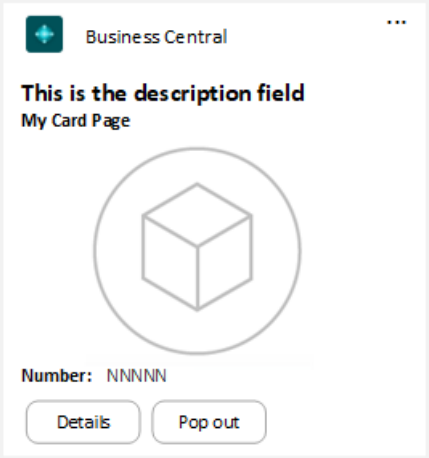
Use the `Brick` field group to identify a subset of table fields that best summarize a record. Typically, you include fields that allow users to quickly identify the record and give insight about the record at a glance. The following code is snippet of `Brick` field group on a table:

```
fieldgroups
{
    fieldgroup(Brick; field[, field][, field])
    {
    }
}
```

The order that you specify fields is important. For an explanation, see Understanding the card layout in Teams in this article.

Example

This code example adds a simple table and card page. The `Brick` field group specifies which fields of the table will display in a Teams card, as illustrated by the following figure:



```
Table 50100 MyTable
{

    fields
    {
        field(1; Number; Integer)
        {
        }
        field(2; Description; Text[50])
        {
        }
        field(3; Inventory; Integer)
        {
        }
        field(4; Image; Media)
        {
        }
    }

    keys
    {
        key(PK; Number)
        {
        }
    }

    fieldgroups
    {
        fieldgroup(Brick; Number, Description, Image)
        {
        }
    }
}

page 50100 MyPage
{
    PageType = Card;
    ApplicationArea = All;
    UsageCategory = Administration;
    SourceTable = MyTable;
    Caption = 'My Card Page';

    layout
    {
        area(Content)
        {
            group(GroupName)
            {
                field(Number; Number)
                {
                    ApplicationArea = All;
                }
                field(Description; Description)
                {
                    ApplicationArea = All;
                }
                field(Inventory; Inventory)
                {
                    ApplicationArea = All;
                }
            }
        }
    }
}
```

For more information about how to specify the `Brick` field group. see Displaying Data as Tiles.

Extensibility limitations on the Brick field group

The following list provides information to keep in mind when designing table and table extension objects in your extensions:

- You can add a `Brick` field group to a new table object.
- You can add a `Brick` field group to a table extension object as long as the base table it extends doesn't already have one.
- In a table extension object, you can't add to or overwrite a `Brick` field group that's already specified for a base table.

Extending Teams cards by using events

Business Central offers the following events for customizing the fields and data that appear in a Teams card:

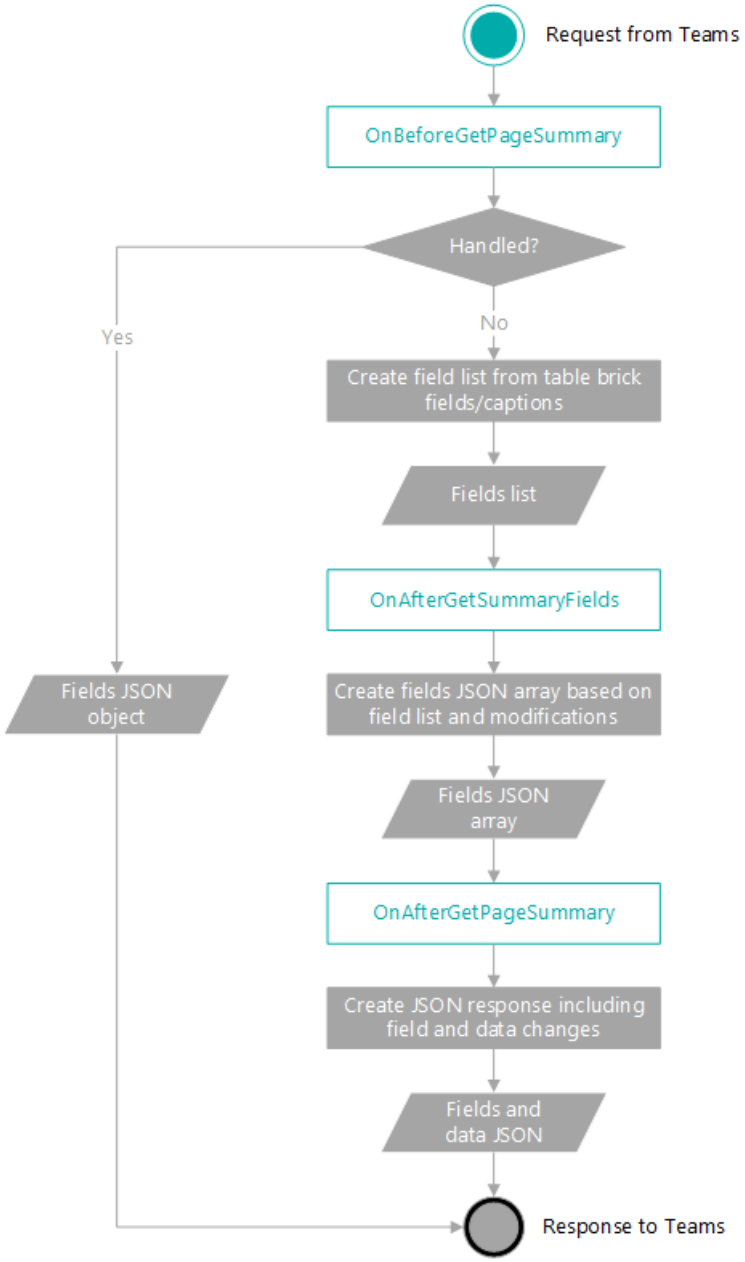
EVENT	DESCRIPTION
OnBeforeGetPageSummary	Use this event to override **all data** for a Card. By handling the event, you can specify a custom set of key-value pairs that will be displayed. The platform won't attempt to select any list of fields or their corresponding values if you subscribe to this event.
OnAfterGetSummaryFields	Use this event to specify a custom set of fields from the same table as the record. Using this method you can add more fields than are those fields specified by the `Brick` field group on the source table. For example, you could add fields that are added to the source table by a table extension.
OnAfterGetPageSummary	Use this event to modify fields and their values that are already selected for the card by the platform. For example, you could change the captions or value of a field.

The events are part of the Page Summary Provider module of the Microsoft System Application.

> **NOTE**
> In most cases, we recommend you set the `Brick` field group instead of using events to define card content. This ensures a consistent experience across the Business Central Web client, mobile devices, and Teams.

Event flow

The following figure illustrates the sequence of the events and operations involved in building the card in Teams. The flow has been simplified for illustration purposes.



OnBeforeGetPageSummary event

The OnBeforeGetPageSummary event gives you the most control over the fields and data that appears in the card. With this event, you specify a complete set of fields as key-value pairs using JSON Array. Only these fields will appear on the card, even if the source table has a `Brick` field group or other extensions include code that modifies the card.

Syntax

The OnBeforeGetPageSummary event subscription has the following syntax:

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Page Summary Provider", 'OnBeforeGetPageSummary', '',
false, false)]
local procedure OnBeforeGetPageSummary(PageId: Integer; RecId: RecordId; var FieldsJsonArray: JsonArray; var
Handled: Boolean);
```

Parameters

PARAMETER	DESCRIPTION
`PageId`	The ID of the page for which to retrieve page summary.

PARAMETER	DESCRIPTION
`RecId`	The underlying record ID of the source table for the page being retrieved, based on the bookmark.
`FieldsJsonArray`	A JSON array that includes the fields and values to display on the card, if the event is handled.
`Handled`	Specifies whether the event has been handled and no further execution should occur.

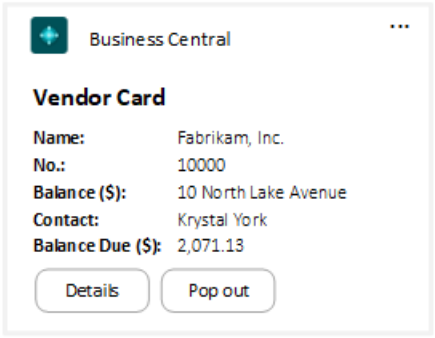
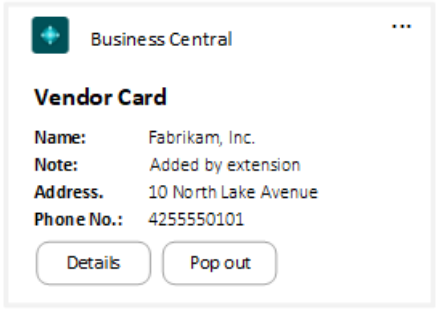
Behavior and recommended use

This event is raised before platform tries to get summary fields for the card from the table, for example, from the `Brick` field group. So when you subscribe to this event, the fields in the `Brick` field group aren't used. Subscribing to this event prohibits other extensions from adding, modifying, or removing summary fields on the card by the other event.

For these reasons, use this event only if you don't want to use any fields of the source table in the card.

Example

The following code example uses the OnBeforeGetPageSummary event to change the fields on a Teams card, as shown in the following table:

BEFORE	AFTER
	

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Page Summary Provider", 'OnBeforeGetPageSummary', '',
false, false)]
local procedure OnBeforeGetPageSummary(PageId: Integer; RecId: RecordId; FieldsJsonArray: JsonArray; var
Handled: Boolean)
var
    Vendor: Record Vendor;
begin
    if PageId <> Page::"Vendor Card" then
        exit;

    if not Vendor.Get(RecId) then
        exit;

    AddField(FieldsJsonArray, 'Name', Vendor.Name, 'Text');
    AddField(FieldsJsonArray, 'Note', 'Added by extension', 'Text');
    AddField(FieldsJsonArray, 'Address', Vendor.Address, 'Text');
    AddField(FieldsJsonArray, 'Phone No.', Vendor."Phone No.", 'Text');

    Handled := true;
end;

local procedure AddField(var FieldsJsonArray: JsonArray; Caption: Text; FieldValue: Text; FieldType: Text)
var
    FieldsJsonObject: JsonObject;
begin
    FieldsJsonObject.Add('caption', Caption);
    FieldsJsonObject.Add('fieldValue', FieldValue);
    FieldsJsonObject.Add('fieldType', FieldType);
    FieldsJsonArray.Add(FieldsJsonObject);
end;
```

OnAfterGetSummaryFields event

The OnAfterGetSummaryFields event lets you add or remove from the set of fields selected by the platform. For example, if the platform automatically selects fields based on its `Brick` field group, use this event to add fields from the same table, which aren't part of the `Brick` field group.

Syntax

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Page Summary Provider", 'OnAfterGetSummaryFields', '',
false, false)]
local procedure OnAfterGetSummaryFields(PageId: Integer; RecId: RecordId; var FieldList: List of [Integer]);
```

Parameters

PARAMETER	DESCRIPTION
`PageId`	The ID of the page object in Business Central that the card is generated for.
`RecId`	The underlying record ID in the source table for the page being retrieved, based on the bookmark.
`FieldList`	The list of fields that will be returned.

Behavior and recommended use

This event is raised after the platform gets the summary fields from the table.

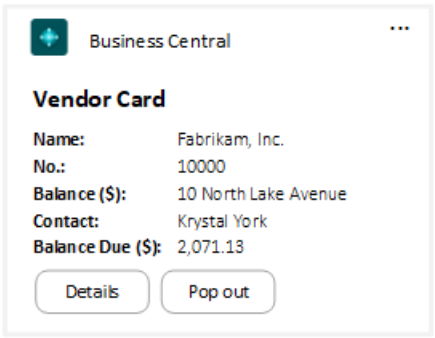
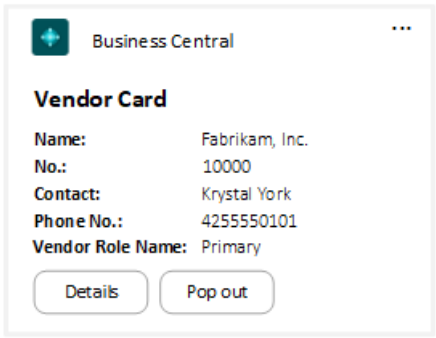
The event the ideal for specifying a custom set of fields from the same table. It enables the platform to fetch the corresponding values for the current record. If you extended a table with fields, use this event to add the fields to the card.

Here are some considerations, when using this event:

- Use the FieldList to add or remove individual fields you want to show on the card in Teams.
- Fields on the card in Teams are shown in the order specified the `FieldList`.
- Don't remove all fields from the list, because you'll be removing fields that are added by other partner extensions.

Example

The following code example uses the OnAfterGetSummaryFields event to change the fields on a Teams card, as shown in the following table:

BEFORE	AFTER
	


```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Page Summary Provider", 'OnAfterGetSummaryFields', '',
false, false)]
local procedure OnAfterGetSummaryFields(PageId: Integer; RecId: RecordId; var FieldList: List of [Integer])
var
    Vendor: Record Vendor;
begin
    if PageId <> Page::"Vendor Card" then
        exit;

    // Remove Balance Due  details
    FieldList.Remove(Vendor.FieldNo("Balance Due (LCY)"));
    FieldList.Remove(Vendor.FieldNo("Balance (LCY)"));

    // Add contact number
    FieldList.Add(Vendor.FieldNo("Phone No."));

    // Add new field "Vendor Role Name" from extension
    FieldList.Add(Vendor.FieldNo("Vendor Role Name"));
end;
```

In the code, the **Vendor Role Name** field is defined in a table extension object that extends the **Vendor** table.

OnAfterGetPageSummary event

This event allows you to modify, add, or remove fields included in the card through the direct access to the FieldsJsonArray.

Syntax

```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Page Summary Provider", 'OnAfterGetPageSummary', '',
false, false)]
local procedure OnAfterGetPageSummary(PageId: Integer; RecId: RecordId; var FieldsJsonArray: JsonArray);
```

Parameters

PARAMETER	DESCRIPTION
`PageId`	The ID of the page for which to retrieve page summary.
`RecId`	The underlying record ID of the source table for the page being retrieved, based on the bookmark.
`FieldsJsonArray`	A Json array that lists the fields to include in the card, overriding the current fields and values set for the card.

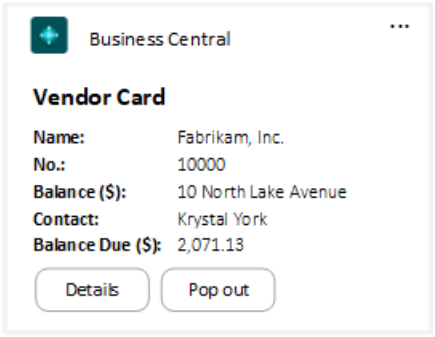
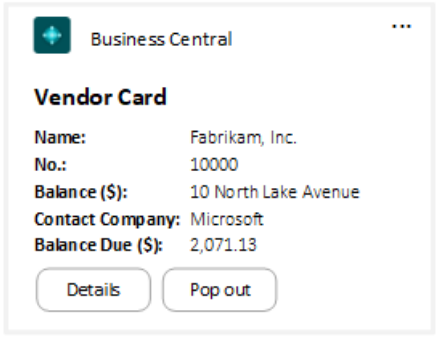
Behavior and recommended use

This event is ideal for modifying the list of fields and their values that are selected by the platform.

Use the event to change the value of existing fields or to add fields that don't already exist on the record.

Example

The following code example uses the OnAfterGetPageSummary event to change the fields on a Teams card, as shown in the following table:

BEFORE	AFTER
	


```
[EventSubscriber(ObjectType::Codeunit, Codeunit::"Page Summary Provider", 'OnAfterGetPageSummary', '',
false, false)]
local procedure OnAfterGetPageSummary(PageId: Integer; RecId: RecordId; var FieldsJsonArray: JsonArray)
var
    FieldJsonToken: JsonToken;
    CaptionToken: JsonToken;
    fieldNo: Integer;
begin
    if PageId <> Page::"Vendor Card" then
        exit;

    // Change value of field caption
    for fieldNo := 0 to FieldsJsonArray.Count() - 1 do begin
        FieldsJsonArray.Get(fieldNo, FieldJsonToken);
        FieldJsonToken.AsObject().Get('caption', CaptionToken);
        if CaptionToken.AsValue().AsText() = 'Contact' then begin
            // FieldJsonToken.AsObject().Replace('caption', 'Contact Company');
            // FieldJsonToken.AsObject().Replace('fieldValue', 'Microsoft');
            FieldJsonToken.AsObject().Replace('caption', 'Note');
            FieldJsonToken.AsObject().Replace('fieldValue', 'Added by extension');
            FieldsJsonArray.Set(fieldNo, FieldJsonToken);
            exit;
        end;
    end;
end;
```

In the code, the **Vendor Role Name** field is defined in a table extension object that extends the **Vendor** table.

See Also

FAQ for Teams Integration
Field Groups
Designing List Pages
Working With Media on Records

FAQ for Microsoft Teams Integration with Business Central

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central online

What's the technology behind the card in Teams?

The ability to translate a URL into a card is known as link *unfurling* and is one of many extensibility points offered by Teams. When the Business Central link is pasted into message box of a Teams conversation, Teams does the following operations:

1. Matches the domain of the URL to Business Central, which is associated with the Business Central app for Teams.
2. Connects to a micro-service that forms part of the Business Central Online services
3. Returns an adaptive card.

Adaptive cards are platform-agnostic way to display compact blocks of information. To learn more, see Link unfurling and Cards in the Teams documentation.

What's the technology behind the card details?

Within the portfolio of extensibility points, Teams offers *task modules*. A task module is a mechanism to display a task-focused window within Teams. Applying the strengths of Business Central's modern client stack, Teams displays an adaptation of the Business Central Web client in this window.

To learn more, see Task Modules in the Teams documentation.

What licenses are required for these features to work?

This table gives you an overview of the licenses needed for these features in a Teams conversation.

WHAT	TEAMS LICENSE	BUSINESS CENTRAL LICENSE
Send a card	✅	✅
View a card	✅	
View card details	✅	✅

What permissions in Business Central are required for these features to work?

A user who pastes a link into a conversation must have at least read permission to the page and data that is targeted by the link. Otherwise, the link won't expand into card. This requirement is identical to what is required in Business Central.

Once a card has been submitted in a conversation, any user in that conversation can view that card without having permission to Business Central. However, to view the card details, users have to have at least read

permission to the target page and data. To make changes to data, they'll need modify permissions.

Does the Business Central app for Teams allow me to create tabs in Teams for displaying Business Central?

No. This scenario isn't supported. The Business Central platform doesn't provide any extensibility or standard features to display a page inside a Teams tab.

Does the Business Central app for Teams allow me to connect to Teams APIs?

The Business Central platform doesn't provide any specific ways to call Team APIs from AL code, such as programmatically creating a new team with channels and users. To extend Business Central using Teams APIs, known as "connectors", see Teams extensibility documentation at https://developer.microsoft.com/en-us/microsoft-teams.

See Also

Field Groups
Designing List Pages
Working With Media on Records

Deprecated Tables

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

In the latest version of Business Central, a number of tables have been deprecated. The following tables have been marked with an `ObsoleteState:Removed`. Code that uses the deprecated table or tables, must be rewritten to use the new table or tables. The deprecated tables are mapped as follows:

DEPRECATED TABLE NAME	NEW TABLE NAME
NAV App	Published Application
NAV App Object Metadata	Application Object Metadata
NAV App Resource	Application Resource
NAV App Dependencies	Application Dependency
NAV App Tenant App	Installed Application

See Also

Technical Upgrade

Deprecated Features in W1

2/17/2021 • 7 minutes to read • Edit Online

This article describes the features that have been moved, removed, or replaced in the W1 version of Business Central. This information will change with future releases, and might not include each deprecated feature.

Deprecated features won't be available in future versions of Business Central, which can happen for different kinds of reasons. For example, a feature may no longer be relevant, or something better may have become available. If you use a feature that is listed, either the feature itself or an extension of it, you should look for or develop an alternative.

The next sections give a brief description of the deprecated features, state what happened to the feature, and explain why. The following table gives a few examples of what we mean by "moved, removed, or replaced."

STATE	EXAMPLES
Moved	The capability has been moved from local functionality to W1 because it was no longer specific to one or more country versions. The capability was combined with other related functionality to eliminate redundancy.
Removed	The capability will be removed from Business Central in a coming release.
Replaced	Something better has become available, and will be used instead.

Changes in 2022 release wave 1

Web Service Access Keys (Basic Auth) for Business Central Online (SaaS)

The following feature will be **Removed** with Business Central 2022 release wave 1.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed (for SaaS only)	The capability to access web services in Business Central using Web Service Access Key (Basic Auth) is deprecated for SaaS. OAuth2 will be the authentication option for SaaS. OAuth samples are published in the BCTech repo. For on-premises, Web Service Access Key (Basic Auth) will remain an option for the time being. This change has no impact on how Business Central connects to external services.

Changes in 2021 release wave 2

Standard APIs, Beta version

The following feature will be **Removed** with Business Central 2021 release wave 2.

MOVED, REMOVED, OR REPLACED?	WHY?

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	Beta version of the standard APIs will be removed by 2021 release wave 2. At this point, Beta APIs will not be available in new releases of Business Central. There are many improvements to v1.0 and v2.0 of the standard APIs. Improvements include more APIs, better performance and improved OData capabilities. It's recommended that integrations move to v2.0 of the standard APIs.

Automation APIs, Beta version

The following feature will be **Removed** with Business Central 2021 release wave 2.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	Beta version of the Automation APIs will be removed by 2021 release wave 2. At this point, Automation Beta APIs will not be available in new releases of Business Central. It's recommended that integrations move to v2.0 of the Automation APIs.

Changes in 2021 release wave 1

Expose UI pages as SOAP endpoints (Warning)

In Business Central 2021 release wave 1, a warning will be shown if you expose UI pages as SOAP endpoints. The capability of exposing UI pages as SOAP endpoints will be removed in a later release.

MOVED, REMOVED, OR REPLACED?	WHY?
Replaced	SOAP has been superseded by OData V4. SOAP endpoints will be deprecated as of Business Central 2021 release wave 1, but the feature won't be removed in this release. It's recommended that integrations are migrated to OData V4 as soon as possible.

OData V3

The following feature will be **Removed** with Business Central 2021 release wave 1.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	OData V3 has been superseded by OData v4. OData V3 is deprecated, and will be removed as of Business Central 2021 release wave 1. It's recommended that integrations are migrated to OData v4 as soon as possible.

Deprecated Features in 2020 release wave 1

The following feature was marked as obsolete:pending in 2020 release wave 1.

Best Price Calculations

When you have recorded special prices and line discounts for sales and purchases, Business Central ensures that your profit on item trade is always optimal by automatically calculating the best price on sales and purchase documents and on job and item journal lines.

MOVED, REMOVED, OR REPLACED?	WHY?
Replaced	The functionality is replaced with new calculations that you can extend to include additional sources or calculation methods. The current capabilities will be available, and can be used in parallel with the new, until 2021 release wave 1. For more information, see Extending Best Price Calculations.

Deprecated Features in 2019 release wave 2

The following sections describe the features that were deprecated in 2019 release wave 2.

The Bank Data Conversion Service

You can use the bank data conversion service from AMC to convert bank data from an XML file that you export from Business Central into a format that your bank can accept.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The functionality has been moved to an extension. It now ships as the AMC Banking 365 Fundamentals extension, which can convert bank data to formats that are used by more than 600 banks worldwide. For more information, see Using the AMC Banking 365 Fundamentals extension.

The Windows Client

You can use Business Central in the Windows client that is installed on your computer.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	Business Central continues to evolve the modern client experiences where users work with Business Central in the browser, Windows 10 desktop app, or mobile apps on Android and iOS. The legacy Dynamics NAV Windows client is no longer available for deployment. Instead, users can switch to the modern experience in the browser, the Android/iOS mobile apps, or the Windows 10 desktop app (available through the respective stores).

Reports 204-207

You can generate external-facing documents, such as sales invoices and order confirmations, that you send to customers as PDF files.

The reports in the 204-207 range are replaced by the following updated reports in the 1304 to 1307 range:

- 204, Sales-Quote -> 1304, Standard Sales-Quote
- 205, Order-Confirmation -> 1305, Standard Sales-Order Conf.
- 206, Sales-Invoice -> 1306, Standard Sales-Invoice
- 207, Credit-Memo -> 1307, Standard Sales-Credit Memo

> **NOTE**
>
> The **No. of Copies** field is removed from the new reports because of performance reasons and because selection of the quantity to print works differently on modern printers. To print a report more than once, users can list the same report multiple times on the **Report Selection** page for the document type in question.

MOVED, REMOVED, OR REPLACED?	WHY?
Replaced	The reports in the 204-207 range are replaced by the reports in the 1304 to 1307 range. To avoid duplicated features, the reports in the 204-207 range will be removed.

User Personalizations and Profile Configurations

You can personalize pages and configure profiles by adding or removing fields, and Business Central will save your changes.

MOVED, REMOVED, OR REPLACED?	WHY?
Replaced	The shift to AL caused the legacy personalization and profile configuration features to become outdated, so we have introduced new tooling. In this release, existing personalizations and configurations are discarded, and you must use the new tools to recreate them. Your new changes will be kept in future releases.

Excel COM Add-In

You can export data to an Excel workbook.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	The Excel COM add-in was installed along with the Windows client. Now that the Windows Client is no longer available, neither is the add-in. To export data to Excel, use the **Edit in Excel** action.

Printing Programmatically

You can print documents such as invoices automatically, without prompting the user or without the user choosing to do so.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	This feature was tied to the Windows Client, which is no longer available.

Objects that have been marked as obsolete

Part of deprecating features is marking the objects that comprise them as "obsolete." Before we deprecate an object, we tag it as "obsolete:pending" to alert our partners of it's deprecation. The object will have the tag for one year before we remove it from Business Central.

Breaking Changes

When we move, remove, or replace an object, breaking changes can result in other apps or extensions that use the object. To help our partners identify and resolve breaking changes, we have created a Breaking Changes document that lists known issues and suggestions for what to do about them.

Features that are available only in the online version

Some features are available only under very specific circumstances, or not at all intended for use in on-premises versions of Business Central. For a list and descriptions of those features, see Features not implemented in on-

premises deployments.

See Also

AIAppExtensions repository

Deprecated Fields, and Fields Marked as Obsolete

2/17/2021 • 42 minutes to read • Edit Online

In the latest version of Business Central, a number of fields have been deprecated in the current release or marked to be obsolete in a later release.

Definitions

Deprecated fields fall into one of the following groups:

1. Fields moved to an extension by Microsoft

   Partner impact: Remember to install the extension when you upgrade an existing solution from an earlier version of Business Central.

   Specifically for the extensions that are required for connecting on-premises solutions with Business Central online for intelligent insights, you must install the **Intelligent Cloud Base Extension** extension first, and then the product-specific extension or extensions.

2. Fields marked as **Obsolete:Pending**

   Partner impact: None in the current release, this is just a heads-up that a change is coming.

3. Fields no longer in use in Microsoft code

   Partner impact: Refactor your code as soon as possible.

Fields marked as ObsoleteState:Pending in Business Central

A number of fields are marked as ObsoleteState:Pending. There is no impact on code in this release.

Austria

The following fields are marked as ObsoleteState:Pending in the AT version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
242	Source Code Setup	5005350	Phys. Invt. Order	Will be removed in a later release.
257	VAT Statement Name	11000	Sales VAT Adv. Notification	Will be removed in a later release.
313	Inventory Setup	5005350	Phys. Inv. Order Nos.	Will be removed in a later release.
313	Inventory Setup	5005352	Posted Phys. Inv. Order Nos.	Will be removed in a later release.
11011	Sales VAT Advance Notification	All	All	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
11012	Transmission Log Entry	All	All	Will be removed in a later release.
11013	Electronic VAT Decl. Setup	All	All	Will be removed in a later release.
11014	Certificate	All	All	Will be removed in a later release.
5005350	Phys. Inventory Order Header	All	All	Will be removed in a later release.
5005351	Phys. Inventory Order Line	All	All	Will be removed in a later release.
5005352	Phys. Invt. Recording Header	All	All	Will be removed in a later release.
5005353	Phys. Invt. Recording Line	All	All	Will be removed in a later release.
5005354	Post. Phys. Invt. Order Header	All	All	Will be removed in a later release.
5005355	Posted Phys. Invt. Order Line	All	All	Will be removed in a later release.
5005356	Posted Phys. Invt. Rec. Header	All	All	Will be removed in a later release.
5005357	Posted Phys. Invt. Rec. Line	All	All	Will be removed in a later release.
5005358	Phys. Inventory Comment Line	All	All	Will be removed in a later release.
5005359	Posted Phys. Invt. Track. Line	All	All	Will be removed in a later release.
5005360	Phys. Invt. Tracking Buffer	All	All	Will be removed in a later release.
5005361	Expect. Phys. Inv. Track. Line	All	All	Will be removed in a later release.
5005362	Post. Exp. Ph. In. Track. Line	All	All	Will be removed in a later release.

Belgium

The following fields are marked as ObsoleteState:Pending in the BE version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
4	Currency	2000000	ISO Currency Code	Will be removed in a later release.
9	Country/Region	2000000	ISO Country/Region Code	Will be removed in a later release.
11306	Electronic Banking Setup	22	Notification E-mail address	Will be removed in a later release.
11306	Electronic Banking Setup	23	Language	Will be removed in a later release.
11306	Electronic Banking Setup	31	IBS Service Version	Will be removed in a later release.
11306	Electronic Banking Setup	21	IBS Version	Will be removed in a later release.
11306	Electronic Banking Setup	24	Upload Integration Mode	Will be removed in a later release.
11306	Electronic Banking Setup	29	IBS Log Download Nos.	Will be removed in a later release.
11306	Electronic Banking Setup	40	Test Environment	Will be removed in a later release.
11306	Electronic Banking Setup	30	IBS Request ID	Will be removed in a later release.
11306	Electronic Banking Setup	28	IBS Log Upload Nos.	Will be removed in a later release.
11306	Electronic Banking Setup	25	Upload Path	Will be removed in a later release.
11306	Electronic Banking Setup	26	Download Integration Mode	Will be removed in a later release.
11306	Electronic Banking Setup	27	Download Path	Will be removed in a later release.
2000010	IBS Log	All	All	Will be removed in a later release.
2000011	IBS Contract	All	All	Will be removed in a later release.
2000012	IBS Account	All	All	Will be removed in a later release.
2000013	IBS Account Conflict	All	All	Will be removed in a later release.

Canada

The following fields are marked as ObsoleteState:Pending in the CA version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	10004	SAT Certificate Thumbprint	Will be removed in a later release.
10000	PAC Web Service	22	Certificate Thumbprint	Will be removed in a later release.

Czech Republic

The following fields are marked as ObsoleteState:Pending in the CZ version.

Different types of disposal and maintenance:

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5603	FA Setup	31043	FA Maintenance By Maint. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5603	FA Setup	31045	FA Disposal By Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5606	FA Posting Group	31042	Use Standard Disposal	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Posting Description:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
18	Customer	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
23	Vendor	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
36	Sales Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
38	Purchase Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
302	Finance Charge Memo Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11779	Fin. Charge Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
312	Purchases & Payables Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
313	Inventory Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5109	Purchase Header Archive	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5900	Service Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5911	Service Mgt. Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11785	Posting Description	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11786	Posting Desc. Parameter	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

General Journal Reconciliation:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
269	G/L Account Net Change	11760	Type	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Industry Classification:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
18	Customer	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
23	Vendor	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
38	Purchase Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
79	Company Information	11793	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
110	Sales Shipment Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
120	Purch. Rcpt. Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
122	Purch. Inv. Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
124	Purch. Cr. Memo Hdr.	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5990	Service Shipment Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6650	Return Shipment Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6660	Return Receipt Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11791	Industry Code	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

VAT Rounding Improvements:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
98	General Ledger Setup	11765	Round VAT Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11766	VAT Coeff. Rounding Precision	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Postponing VAT on Sales Credit Memo:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
36	Sales Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
91	Gen. Journal Line	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	11764	Postponed VAT Realized	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11777	Credit Memo Confirmation	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
325	VAT Posting Setup	11764	Sales VAT Postponed Account	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5911	Service Mgt. Setup	11777	Credit Memo Confirmation	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5994	Service Cr.Memo Header	11764	Postponed VAT Realized	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Non-deductible VAT:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
39	Purchase Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
39	Purchase Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
39	Purchase Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
49	Invoice Post. Buffer	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
49	Invoice Post. Buffer	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
49	Invoice Post. Buffer	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
81	Gen. Journal Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11772	VAT Base LCY (Non Deduct.)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11773	VAT Amount LCY (Non Deduct.)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11762	Statement Templ. Name Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11763	Statement Name Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11764	Statement Line No. Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
123	Purch. Inv. Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
123	Purch. Inv. Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
123	Purch. Inv. Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
125	Purch. Cr. Memo Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
125	Purch. Cr. Memo Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
125	Purch. Cr. Memo Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
242	Source Code Setup	11762	VAT Coefficient	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11770	Primary Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
290	VAT Amount Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
290	VAT Amount Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
290	VAT Amount Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
325	VAT Posting Setup	11763	Non Deduct. VAT Corr. Account	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
325	VAT Posting Setup	11766	Allow Non Deductible VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11731	Cash Document Line	602	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11731	Cash Document Line	603	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11731	Cash Document Line	604	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11736	Posted Cash Document Line	602	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11736	Posted Cash Document Line	603	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
11736	Posted Cash Document Line	604	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	1	VAT Bus. Posting Group	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	2	VAT Prod. Posting Group	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	3	From Date	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	4	Non Deductible VAT %	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

VAT Registration in Other Countries:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
32	Item Ledger Entry	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
38	Purchase Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
38	Purchase Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
83	Item Journal Line	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
110	Sales Shipment Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
120	Purch. Rcpt. Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
122	Purch. Inv. Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
122	Purch. Inv. Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
124	Purch. Cr. Memo Hdr.	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
124	Purch. Cr. Memo Hdr.	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	31061	Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	31062	Currency Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
261	Intrastat Jnl. Template	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
262	Intrastat Jnl. Batch	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5107	Sales Header Archive	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5109	Purchase Header Archive	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5109	Purchase Header Archive	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5990	Service Shipment Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5990	Service Shipment Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5992	Service Invoice Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5992	Service Invoice Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6650	Return Shipment Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6660	Return Receipt Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11762	Registration Country/Region	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11763	Registr. Country/Region Route	All	All fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11764	Perf. Country Curr. Exch. Rate	All	All fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31000	Sales Advance Letter Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31000	Sales Advance Letter Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31020	Purch. Advance Letter Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31020	Purch. Advance Letter Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31066	VIES Declaration Header	80	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31100	VAT Control Report Header	15	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

VAT Identifier:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
98	General Ledger Setup	11771	Check VAT Identifier	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11767	VAT Identifier	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11768	VAT Identifier Translate	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Reverse Charge Statement:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31065	Stat. Reporting Setup	31090	Reverse Charge Nos.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31065	Stat. Reporting Setup	31095	Reverse Charge Auth. Emp. No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31065	Stat. Reporting Setup	31096	Rvrs. Chrg. Filled by Emp. No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31093	Reverse Charge Header	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31094	Reverse Charge Line	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Fixed Assets Classification - SKP codes:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5600	Fixed Asset	31044	SKP Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31043	SKP Code	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Fixed Assets – Depreciation FA Appreciation From:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5612	FA Depreciation Book	31040	Depr. FA Appreciation From	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Fixed Assets – item consumption for maintenance:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
32	Item Ledger Entry	31043	FA No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
32	Item Ledger Entry	31044	Maintenance Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
32	Item Ledger Entry	31045	Maintenance Amount	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	31070	Item Ledger Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
83	Item Journal Line	31043	FA No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
83	Item Journal Line	31044	Maintenance Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
210	Job Journal Line	31043	FA No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
210	Job Journal Line	31044	Maintenance Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
242	Source Code Setup	31041	Maintenance Adjustment	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
313	Inventory Setup	31042	Automatic Maintenance Posting	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5600	Fixed Asset	31042	Deprec. Book Code (Mainten.)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5621	FA Journal Line	31040	Item Ledger Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5625	Maintenance Ledger Entry	31040	Item Ledger Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Tax corrective documents for VAT:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
36	Sales Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
37	Sales Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
113	Sales Invoice Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
115	Sales Cr.Memo Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
254	VAT Entry	11780	Pmt.Disc. Tax Corr.Doc. No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11772	Reas.Cd. on Tax Corr.Doc.Mand.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11773	Pmt.Disc.Tax Corr.Doc. Nos.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11774	Copy As Tax Corr. Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11775	Reason Code For Payment Disc.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5108	Sales Line Archive	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5902	Service Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5911	Service Mgt. Setup	11772	Reas.Cd. on Tax Corr.Doc.Mand.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5911	Service Mgt. Setup	11775	Reason Code For Payment Disc.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5992	Service Invoice Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5993	Service Invoice Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5995	Service Cr.Memo Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6661	Return Receipt Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Vendor Templates:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
11794	Vendor Template	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Item Charges Enhancements:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
313	Inventory Setup	31076	Check Item Charge Pst.Group	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5800	Item Charge	31070	Use Ledger Entry Dimensions	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31071	Sales Only	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31072	Purchase Only	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31073	Disable Receipt Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31074	Disable Transfer Receipt Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31075	Disable Return Shipment Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31076	Disable Sales Shipment Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31077	Disable Return Receipt Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31078	Assignment on Receive/Shipment	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

No. Series Enhancements:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
308	No. Series	11790	Mask	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
308	No. Series	11791	No. Series Link Exists	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11730	Receiving Wh. No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11797	Shipping No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11798	Receiving No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11799	Shipping Wh. No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11799	No. Series Link	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Packaging Tax Calculation:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31070	Package Material	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31071	Item Package Material	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Dimension for VAT Entry:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
254	VAT Entry	11771	Global Dimension 1 Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11772	Global Dimension 2 Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11773	Dimension Set ID	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Disable Cards Deleting:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
98	General Ledger Setup	11792	Delete Card with Entries	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Field for Full Description:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
15	G/L Account	11792	Full Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
18	Customer	11792	Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
23	Vendor	11792	Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
27	Item	11792	Full Description	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
260	Tariff Number	11792	Full Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
260	Tariff Number	11793	Full Name ENG	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5050	Contact	11792	Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5079	Marketing Setup	11792	Inherit Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5600	Fixed Asset	11792	Full Description	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5718	Nonstock Item	11792	Full Description	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Customer Template extension:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5105	Customer Template	11790	No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5105	Customer Template	11791	Language Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Indivisible Unit of Measure:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
204	Unit of Measure	31070	Indivisible Unit	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5404	Item Unit of Measure	31070	Indivisible Unit	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Check Balance in General Ledger Journal:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
80	Gen. Journal Template	11761	Not Check Correction	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Denmark

The following fields are marked as ObsoleteState:Pending in the Danish version.

TABLE ID	FIELD ID	COMMENTS
3	13600	Will be removed in a later release.
4	13600	Will be removed in a later release.
9	13600	Will be removed in a later release.
18	13605	Will be removed in a later release.
36	13600	Will be removed in a later release.
36	13601	Will be removed in a later release.
36	13602	Will be removed in a later release.
36	13604	Will be removed in a later release.
36	13605	Will be removed in a later release.
36	13606	Will be removed in a later release.
36	13607	Will be removed in a later release.
36	13620	Will be removed in a later release.
37	13600	Will be removed in a later release.

TABLE ID	FIELD ID	COMMENTS
295	13600	Will be removed in a later release.
295	13602	Will be removed in a later release.
295	13605	Will be removed in a later release.
295	13606	Will be removed in a later release.
295	13607	Will be removed in a later release.
295	13608	Will be removed in a later release.
295	13620	Will be removed in a later release.
296	13600	Will be removed in a later release.
297	13600	Will be removed in a later release.
297	13601	Will be removed in a later release.
297	13602	Will be removed in a later release.
297	13605	Will be removed in a later release.
297	13606	Will be removed in a later release.
297	13607	Will be removed in a later release.
297	13608	Will be removed in a later release.
297	13620	Will be removed in a later release.
298	13600	Will be removed in a later release.
302	13600	Will be removed in a later release.
302	13601	Will be removed in a later release.
302	13602	Will be removed in a later release.
302	13605	Will be removed in a later release.
302	13606	Will be removed in a later release.
302	13607	Will be removed in a later release.
302	13608	Will be removed in a later release.
302	13620	Will be removed in a later release.

TABLE ID	FIELD ID	COMMENTS
303	13600	Will be removed in a later release.
304	13600	Will be removed in a later release.
304	13601	Will be removed in a later release.
304	13602	Will be removed in a later release.
304	13605	Will be removed in a later release.
304	13606	Will be removed in a later release.
304	13607	Will be removed in a later release.
304	13608	Will be removed in a later release.
304	13620	Will be removed in a later release.
305	13600	Will be removed in a later release.
311	13600	Will be removed in a later release.
311	13601	Will be removed in a later release.
311	13602	Will be removed in a later release.
311	13603	Will be removed in a later release.
311	13604	Will be removed in a later release.
5107	13600	Will be removed in a later release.
5107	13601	Will be removed in a later release.
5107	13602	Will be removed in a later release.
5107	13605	Will be removed in a later release.
5107	13606	Will be removed in a later release.
5107	13607	Will be removed in a later release.
5107	13608	Will be removed in a later release.
5107	13620	Will be removed in a later release.
5108	13602	Will be removed in a later release.
5900	13600	Will be removed in a later release.

TABLE ID	FIELD ID	COMMENTS
5900	13601	Will be removed in a later release.
5900	13604	Will be removed in a later release.
5900	13608	Will be removed in a later release.
5900	13620	Will be removed in a later release.
5902	13600	Will be removed in a later release.
5911	13600	Will be removed in a later release.
5911	13601	Will be removed in a later release.
5992	13600	Will be removed in a later release.
5992	13601	Will be removed in a later release.
5992	13602	Will be removed in a later release.
5992	13604	Will be removed in a later release.
5992	13608	Will be removed in a later release.
5992	13620	Will be removed in a later release.
5993	13600	Will be removed in a later release.
5994	13600	Will be removed in a later release.
5994	13601	Will be removed in a later release.
5994	13602	Will be removed in a later release.
5994	13604	Will be removed in a later release.
5994	13608	Will be removed in a later release.
5994	13620	Will be removed in a later release.
5995	13600	Will be removed in a later release.

Germany

The following fields are marked as ObsoleteState:Pending in the DE version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
242	Source Code Setup	5005350	Phys. Invt. Order	Will be removed in a later release.
257	VAT Statement Name	11000	Sales VAT Adv. Notification	Will be removed in a later release.
313	Inventory Setup	5005352	Posted Phys. Inv. Order Nos.	Will be removed in a later release.
313	Inventory Setup	5005350	Phys. Inv. Order Nos.	Will be removed in a later release.
11011	Sales VAT Advance Notification	All	All	Will be removed in a later release.
11012	Transmission Log Entry	All	All	Will be removed in a later release.
11013	Electronic VAT Decl. Setup	All	All	Will be removed in a later release.
11014	Certificate	All	All	Will be removed in a later release.
5005350	Phys. Inventory Order Header	All	All	Will be removed in a later release.
5005351	Phys. Inventory Order Line	All	All	Will be removed in a later release.
5005352	Phys. Invt. Recording Header	All	All	Will be removed in a later release.
5005353	Phys. Invt. Recording Line	All	All	Will be removed in a later release.
5005354	Post. Phys. Invt. Order Header	All	All	Will be removed in a later release.
5005355	Posted Phys. Invt. Order Line	All	All	Will be removed in a later release.
5005356	Posted Phys. Invt. Rec. Header	All	All	Will be removed in a later release.
5005357	Posted Phys. Invt. Rec. Line	All	All	Will be removed in a later release.
5005358	Phys. Inventory Comment Line	All	All	Will be removed in a later release.
5005359	Posted Phys. Invt. Track. Line	All	All	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5005360	Phys. Invt. Tracking Buffer	All	All	Will be removed in a later release.
5005361	Expect. Phys. Inv. Track. Line	All	All	Will be removed in a later release.
5005362	Post. Exp. Ph. In. Track. Line	All	All	Will be removed in a later release.

Iceland

The following fields are marked as ObsoleteState:Pending in the IS version.

TABLE ID	FIELD ID	COMMENTS
21	10900	Will be removed in a later release.
311	10900	Will be removed in a later release.

Mexico

The following fields are marked as ObsoleteState:Pending in the MX version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	10004	SAT Certificate Thumbprint	Will be removed in a later release.
10000	PAC Web Service	22	Certificate Thumbprint	Will be removed in a later release.

Netherlands

The following fields are marked as ObsoleteState:Pending in the NL version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
312	Purchases & Payables Setup	11312	Show Totals on Purch. Inv./CM.	Will be removed in a later release.

Spain

The following fields are marked as ObsoleteState:Pending in the ES version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
10751	SII Setup	9	IntracommunityEndpointUrl	Will be removed in a later release.

Switzerland

The following fields are marked as ObsoleteState:Pending in the CH version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
4	Currency	3010541	ISO Currency Code	Will be removed in a later release.
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
98	General Ledger Setup	11004	Post Pmt.Disc Tol. to Pmt.Disc	Will be removed in a later release.
242	Source Code Setup	5005350	Phys. Invt. Order	Will be removed in a later release.
257	VAT Statement Name	11000	Sales VAT Adv. Notification	Will be removed in a later release.
311	Sales & Receivables Setup	11501	Line Amt. Round LCY	Will be removed in a later release.
313	Inventory Setup	5005350	Phys. Inv. Order Nos.	Will be removed in a later release.
313	Inventory Setup	5005352	Posted Phys. Inv. Order Nos.	Will be removed in a later release.
11011	Sales VAT Advance Notification	All	All	Will be removed in a later release.
11012	Transmission Log Entry	All	All	Will be removed in a later release.
11013	Electronic VAT Decl. Setup	All	All	Will be removed in a later release.
11014	Certificate	All	All	Will be removed in a later release.
5005350	Phys. Inventory Order Header	All	All	Will be removed in a later release.
5005351	Phys. Inventory Order Line	All	All	Will be removed in a later release.
5005352	Phys. Invt. Recording Header	All	All	Will be removed in a later release.
5005353	Phys. Invt. Recording Line	All	All	Will be removed in a later release.
5005354	Post. Phys. Invt. Order Header	All	All	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5005355	Posted Phys. Invt. Order Line	All	All	Will be removed in a later release.
5005356	Posted Phys. Invt. Rec. Header	All	All	Will be removed in a later release.
5005357	Posted Phys. Invt. Rec. Line	All	All	Will be removed in a later release.
5005358	Phys. Inventory Comment Line	All	All	Will be removed in a later release.
5005359	Posted Phys. Invt. Track. Line	All	All	Will be removed in a later release.
5005360	Phys. Invt. Tracking Buffer	All	All	Will be removed in a later release.
5005361	Expect. Phys. Inv. Track. Line	All	All	Will be removed in a later release.
5005362	Post. Exp. Ph. In. Track. Line	All	All	Will be removed in a later release.

United Kingdom

The following fields are marked as ObsoleteState:Pending in the UK version.

TABLE ID	FIELD ID	COMMENTS
18	10500	Will be removed in a later release.
23	10500	Will be removed in a later release.
36	10501	Will be removed in a later release.
38	10501	Will be removed in a later release.
112	10501	Will be removed in a later release.
114	10501	Will be removed in a later release.
122	10501	Will be removed in a later release.
124	10501	Will be removed in a later release.
5107	10501	Will be removed in a later release.
5109	10501	Will be removed in a later release.

The following tables are marked as ObsoleteState:Pending in the UK version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
10533	MTD-Liability	All	All	Will be removed in a later release.
10534	MTD-Payment	All	All	Will be removed in a later release.
10535	MTD-Return Details	All	All	Will be removed in a later release.

United States

The following fields are marked as ObsoleteState:Pending in the US version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	10004	SAT Certificate Thumbprint	Will be removed in a later release.
10000	PAC Web Service	22	Certificate Thumbprint	Will be removed in a later release.

Base version

A number of fields that are related to pictures are no longer in use, because the pictures are now stored in Image fields that are of type *Media*. The fields are marked as ObsoleteState:Pending.

TABLE ID	FIELD ID	COMMENTS		
18	Customer	89	Picture	Will be removed in a later release.
23	Vendor	89	Picture	Will be removed in a later release.
130	Incoming Document	20	URL2	Will be removed in a later release.
130	Incoming Document	21	URL3	Will be removed in a later release.
130	Incoming Document	22	URL4	Will be removed in a later release.
156	Resource	52	Picture	Will be removed in a later release.
167	Job	57	Picture	Will be removed in a later release.
270	Bank Account	89	Picture	Will be removed in a later release.
5050	Contact	89	Picture	Will be removed in a later release.

TABLE ID	FIELD ID	COMMENTS		
5200	Employee	19	Picture	Will be removed in a later release.
5600	Fixed Asset	22	Picture	Will be removed in a later release.

For more information about Media Data Type, see Media Data Type reference documentation.

A field that is related to VAT Registration Number validation is no longer in use, because the feature was replaced EU VAT Registration No. Validation Service Setup Business Central. The field is marked as ObsoleteState:Pending.

TABLE ID	FIELD ID	COMMENTS		
98	General Ledger Setup	161	VAT Reg. No. Validation URL	Will be removed in a later release.

For more information about validation of VAT Registration Numbers, see Setting Up Calculations and Posting Methods for Value-Added Tax documentation.

A flow field that was used to calculate Balance in My Account page is no longer in use in Business Central and has been replaced with Account Balance field to improve performance. The field is marked as ObsoleteState:Pending.

TABLE ID	FIELD ID	COMMENTS		
9153	My Account	4	Balance	Will be removed in a later release.

A number of fields that are related to product groups are no longer in use, because the feature was replaced by item categories in Microsoft Dynamics NAV 2017. The fields are marked as ObsoleteState:Pending.

TABLE ID	FIELD ID	COMMENTS
5723	All	Deprecated. Do not use.
27	5704	Will be removed in a later release.
32	5707	Will be removed in a later release.
37	5712	Will be removed in a later release.
83	5707	Will be removed in a later release.
111	5712	Will be removed in a later release.
113	5712	Will be removed in a later release.
115	5712	Will be removed in a later release.
123	5712	Will be removed in a later release.

TABLE ID	FIELD ID	COMMENTS
125	5712	Will be removed in a later release.
246	5705	Will be removed in a later release.
753	5707	Will be removed in a later release.
5108	5712	Will be removed in a later release.
5110	5712	Will be removed in a later release.
5741	5712	Will be removed in a later release.
5745	5707	Will be removed in a later release.
5747	5707	Will be removed in a later release.
5902	5712	Will be removed in a later release.
5991	5712	Will be removed in a later release.
5993	5712	Will be removed in a later release.
6651	5712	Will be removed in a later release.
6661	5712	Will be removed in a later release.

For more information about the impact, see The new Item Categories feature replaced the Product Group feature in Dynamics NAV 2017 on the Dynamics NAV team blog. For more information about item categories, see How to: Categorize Items in the Dynamics 365 Business Central documentation.

Fields moved to an extension by Microsoft in Business Central

A number of fields have been moved from the base application to an extension.

Czech Republic

The functionality for Different types of disposal and maintenance in the Czech version has been moved to the Different types of disposal and maintenance (CZ) extension. For more information, see Different types of disposal and maintenance (CZ) Extension in the Dynamics 365 Business Central documentation.

OLD TABLE ID	OLD TABLE NAME	OLD FIELD ID	OLD FIELD NAME	NEW TABLE ID	NEW TABLE NAME	NEW FIELD ID	NEW FIELD NAME
5603	FA Setup	31043	FA Maintenance By Maint. Code	N/A	Not needed after refactoring.	N/A	
5603	FA Setup	31045	FA Disposal By Reason Code	N/A	Not needed after refactoring.	N/A	

OLD TABLE ID	OLD TABLE NAME	OLD FIELD ID	OLD FIELD NAME	NEW TABLE ID	NEW TABLE NAME	NEW FIELD ID	NEW FIELD NAME
5606	FA Posting Group	31042	Use Standard Disposal	N/A	Not needed after refactoring.	N/A	
5615	FA Allocation	11760	Reason/Maintenance Code	5615	FA Allocation	xxxxx	ReasonMaintenanceCode
31042	FA Extended Posting Group	1	FA Posting Group Code	xxxxx	FA Extended Posting Group	1	FA Posting Group Code
31042	FA Extended Posting Group	2	FA Posting Type	xxxxx	FA Extended Posting Group	2	FA Posting Type
31042	FA Extended Posting Group	3	Code	xxxxx	FA Extended Posting Group	3	Code
31042	FA Extended Posting Group	4	Book Val. Acc. on Disp. (Gain)	xxxxx	FA Extended Posting Group	4	Book Val. Acc. on Disp. (Gain)
31042	FA Extended Posting Group	5	Book Val. Acc. on Disp. (Loss)	xxxxx	FA Extended Posting Group	5	Book Val. Acc. on Disp. (Loss)
31042	FA Extended Posting Group	6	Maintenance Expense Account	xxxxx	FA Extended Posting Group	6	Maintenance Expense Account
31042	FA Extended Posting Group	7	Maintenance Bal. Acc.	xxxxx	FA Extended Posting Group	7	Maintenance Bal. Acc.
31042	FA Extended Posting Group	8	Allocated Book Value % (Gain)	xxxxx	FA Extended Posting Group	8	Allocated Book Value % (Gain)
31042	FA Extended Posting Group	9	Allocated Book Value % (Loss)	xxxxx	FA Extended Posting Group	9	Allocated Book Value % (Loss)

OLD TABLE ID	OLD TABLE NAME	OLD FIELD ID	OLD FIELD NAME	NEW TABLE ID	NEW TABLE NAME	NEW FIELD ID	NEW FIELD NAME
31042	FA Extended Posting Group	10	Allocated Maintenanc e %	xxxxx	FA Extended Posting Group	10	Allocated Maintenanc e %
31042	FA Extended Posting Group	31040	Sales Acc. On Disp. (Gain)	xxxxx	FA Extended Posting Group	31040	Sales Acc. On Disp. (Gain)
31042	FA Extended Posting Group	31042	Sales Acc. On Disp. (Loss)	xxxxx	FA Extended Posting Group	31042	Sales Acc. On Disp. (Loss)

Denmark

The functionality for payments and reconciliation in the Danish version (FIK) has been moved to the Payments and Reconciliations (DK) extension. For more information, see The Payments and Reconciliations (DK) Extension in the Dynamics 365 Business Central documentation.

TABLE ID	TABLE NAME	OLD FIELD ID	NEW FIELD ID	OLD FIELD NAME	NEW FIELD NAME
23	Vendor	13650	13651	Giro Acc No.	GiroAccNo
25	Vendor Ledger Entry	13650	13651	Giro Acc No.	GiroAccNo
38	Purchase Header	13650	13651	Giro Acc No.	GiroAccNo
79	Company Information	13600	13651	Bank Creditor No.	BankCreditorNo
81	General Journal Line	13650	13651	Giro Acc No.	GiroAccNo
122	Purchase Invoice Header	13650	13651	Giro Acc No.	GiroAccNo
273	Bank Acc. Reconciliation	13600	13601	FIK Payment Reconciliation	FIKPaymentReco nciliation
274	Bank Acc. Reconciliation Line	13600	13601	Payment Reference	PaymentReferenc e
289	Payment Method	13601	13652	Payment Type Validation	PaymentTypeVali dation
372	Payment Buffer	13650	13651	Giro Acc No.	GiroAccNo
1226	Payment Export Data	13650	13651	Recipient Giro Acc No.	RecipientGiroAcc No

TABLE ID	TABLE NAME	OLD FIELD ID	NEW FIELD ID	OLD FIELD NAME	NEW FIELD NAME
1250	Bank Statement Matching Buffer	13601	13652	Match Status	MatchStatus
1250	Bank Statement Matching Buffer	13600	13653	Description	DescriptionBankStatement

See Also

Upgrading to Business Central
Upgrading the Application Code
Important Information and Considerations for Before Upgrading to Dynamics 365 Business Central

Deprecated Features in the Austrian Version of Dynamics 365 Business Central

2/17/2021 • 3 minutes to read • Edit Online

This topic lists and describes the local functionality for Austria that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Batch Print Sales and Purchase Documents When Posting

When you batch post sales or purchase documents, you can select to also print the related reports.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Batch Print when Posting feature is no longer specific to Austria, so we have made it generally available in the standard product.

Copy Existing Items to Create New Items

When you add a new item, to save time, you can use the **Copy Item** function to copy an existing item to use as a template for a new item.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Copy Item feature is no longer specific to Austria, so we have made it generally available in the standard product.

Physical Inventory Order

You can take inventory of your items by using the **Physical Inventory Order** and **Physical Inventory Recording** pages. The physical inventory order contains data for planning, realizing, and analyzing physical inventory. The physical inventory recording contains the items and quantities to be counted and forms the basis of the print-out to be used in the warehouse.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Physical Inventory Order feature is no longer specific to Austria, so we have made it generally available in the standard product.

Blanket Order Archiving and Document Line Tracking

You can archive and delete blanket sales and purchase orders. You can view documents that are related to sales order lines and purchase order lines, including from archived order lines. Related documents that you can track include quotes, shipments, receipts, and blanket orders. This helps you to identify documents used to process orders.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	Blanket Order Archiving and Document Line Tracking features are no longer specific to Austria, so we have made them generally available in the standard product.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the AT version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
242	Source Code Setup	5005350	Phys. Invt. Order	Will be removed in a later release.
257	VAT Statement Name	11000	Sales VAT Adv. Notification	Will be removed in a later release.
313	Inventory Setup	5005350	Phys. Inv. Order Nos.	Will be removed in a later release.
313	Inventory Setup	5005352	Posted Phys. Inv. Order Nos.	Will be removed in a later release.
11011	Sales VAT Advance Notification	All	All	Will be removed in a later release.
11012	Transmission Log Entry	All	All	Will be removed in a later release.
11013	Electronic VAT Decl. Setup	All	All	Will be removed in a later release.
11014	Certificate	All	All	Will be removed in a later release.
5005350	Phys. Inventory Order Header	All	All	Will be removed in a later release.
5005351	Phys. Inventory Order Line	All	All	Will be removed in a later release.
5005352	Phys. Invt. Recording Header	All	All	Will be removed in a later release.
5005353	Phys. Invt. Recording Line	All	All	Will be removed in a later release.
5005354	Post. Phys. Invt. Order Header	All	All	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5005355	Posted Phys. Invt. Order Line	All	All	Will be removed in a later release.
5005356	Posted Phys. Invt. Rec. Header	All	All	Will be removed in a later release.
5005357	Posted Phys. Invt. Rec. Line	All	All	Will be removed in a later release.
5005358	Phys. Inventory Comment Line	All	All	Will be removed in a later release.
5005359	Posted Phys. Invt. Track. Line	All	All	Will be removed in a later release.
5005360	Phys. Invt. Tracking Buffer	All	All	Will be removed in a later release.
5005361	Expect. Phys. Inv. Track. Line	All	All	Will be removed in a later release.
5005362	Post. Exp. Ph. In. Track. Line	All	All	Will be removed in a later release.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Austria Local Functionality

Deprecated Features in the Belgian Version of Microsoft Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Belgium that has been removed from Dynamics 365 Business Central, made available from a new page or report, or replaced by a new feature.

Isabel integration

You can integrate with Isabel to make it easy to upload and download bank files.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	This feature uses an automation that is not supported by the Dynamics 365 Business Central compiler and blocks the C/AL to AL conversion. In an earlier release we made this functionality unavailable, and now we have removed it. Microsoft partners can provide their solutions for customers who need to integrate with Isabel.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the BE version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
4	Currency	2000000	ISO Currency Code	Will be removed in a later release.
9	Country/Region	2000000	ISO Country/Region Code	Will be removed in a later release.
11306	Electronic Banking Setup	22	Notification E-mail address	Will be removed in a later release.
11306	Electronic Banking Setup	23	Language	Will be removed in a later release.
11306	Electronic Banking Setup	31	IBS Service Version	Will be removed in a later release.
11306	Electronic Banking Setup	21	IBS Version	Will be removed in a later release.
11306	Electronic Banking Setup	24	Upload Integration Mode	Will be removed in a later release.
11306	Electronic Banking Setup	29	IBS Log Download Nos.	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
11306	Electronic Banking Setup	40	Test Environment	Will be removed in a later release.
11306	Electronic Banking Setup	30	IBS Request ID	Will be removed in a later release.
11306	Electronic Banking Setup	28	IBS Log Upload Nos.	Will be removed in a later release.
11306	Electronic Banking Setup	25	Upload Path	Will be removed in a later release.
11306	Electronic Banking Setup	26	Download Integration Mode	Will be removed in a later release.
11306	Electronic Banking Setup	27	Download Path	Will be removed in a later release.
2000010	IBS Log	All	All	Will be removed in a later release.
2000011	IBS Contract	All	All	Will be removed in a later release.
2000012	IBS Account	All	All	Will be removed in a later release.
2000013	IBS Account Conflict	All	All	Will be removed in a later release.

See Also

Upgrading to Microsoft Dynamics 365 Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Belgium Local Functionality

Deprecated Features in the Canadian Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Canada that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Customer Statement Report

Shows a list of financial transactions for a selected customer statements for a given period of time. For example, use the report as part of your payment collection process.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Standard Statement report (report 1316) provides the same capabilities as the Customer Statement report (report 10072), plus the ability to customize layouts in Word, and it is available for all countries.	2019 release wave 2

Aged Accounts Payable Report

Shows a list of aged remaining balances for each vendor for a given period of time.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Aged Accounts Payable report (report 322) provides the same capabilities as the (local) Aged Accounts Payable (report 10085) and it is available for all countries.	2019 release wave 2

Aged Accounts Receivable Report

Shows a list of aged remaining balances for each customer for a given period of time.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Aged Accounts Receivable report (report 120) provides the same capabilities as the (local) Aged Accounts Receivable (report 10040) and it is available for all countries.	2019 release wave 2

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the CA version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	10004	SAT Certificate Thumbprint	Will be removed in a later release.
10000	PAC Web Service	22	Certificate Thumbprint	Will be removed in a later release.

See Also

Upgrading Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Canadian Local Functionality in Business Central

Deprecated Features in the Czech Version of Dynamics 365 Business Central

2/17/2021 • 39 minutes to read • Edit Online

This topic lists and describes the local functionality for Czech that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Statutory Company Information

With this feature, users can:

- define company officials and designate them as General Manager, Accounting Managers, and Finance Managers for usage in internal and external documents.
- define document footers in different languages. Such footers can be used in different reports and documents.
- additional company registration numbers and other registration information can be stored in Company Information, for customers and vendors, and used in documents.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Moved	The Statutory Company Information is no longer specific to Czech, so we have made it generally available in the standard product.	2020 release wave 2

Different types of disposal and maintenance

In the Czech accounting, it is necessary to post different types of disposal and different types of maintenance of Fixed Assets to specific G/L Accounts. A standard way offers only one method of disposal and maintenance posting.

A new setup was added for this feature – **FA Extended Posting Group** table. This table allows to set up for each **FA Posting group** different G/L accounts for:

- posting disposal in combination with **Reason Code** used in disposal transaction.
- posting maintenance in combination with **Maintenance Code** used in maintenance transaction.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Moved	The functionality for Different types of disposal and maintenance in the Czech version will be moved to the Fixed Asset Localization for Czech extension.	2021 release wave 1

Posting Description

This feature enables users to customize the way Dynamics NAV is storing Posting Descriptions during posting to General Ledger Entries.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature is deprecated and no longer needed and as not mandatory it will be deleted.	CU for 01.2021 release 2020 wave 2

General Entry Description

This feature transfers Sales and Purchase document line descriptions to G/L Entry description (instead of document header defined Posting Description). This functionality can be set separately for each account type in the document line.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Moved	The General Entry Description is no longer specific to Czech, so we have made it generally available in the standard product. In standard product is available only for G/L Accounts.	2019 release wave 2

General Journal Reconciliation

This functionality has been extended. Net Change in Jnl. and Balance after Posting are not designated for G/L Accounts marked as Reconciliation Account only, but for all G/L Accounts used in the general journal, and also for other account types - Bank Account, Customer, Vendor, Fixed Asset and IC Partner. The functionality works with amounts in general journal lines and does not calculate VAT amounts for VAT Accounts.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature is deprecated and no longer needed and as not mandatory it will be deleted.	CU for 01.2021 release 2020 wave 2

Industry Classification

This feature adds **Industry Code** classification field to Customer/Vendor tables and Sales/Purchase documents.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	Lack of use. The business need that this functionality was introduced to cover is no longer relevant. Functionality will be deleted.	CU for 01.2021 release 2020 wave 2

VAT Rounding Improvements

Necessary standard functionality modification to round the VAT amount according to the Czech Republic law requirements. The VAT Rounding is subject of VAT Law 235/2004 (§37). The rounding procedure uses rounding precision in order to calculate the result. Further additions include an ability to select special rounding precisions (special rounding is identified by a new field **Round VAT Coeff.** on the **General Ledger Setup** form) during VAT calculation for the following entities in G/L posting, Sales and Purchase.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature is no longer mandatory by law from 2019 and will be deleted.	CU for 01.2021 release 2020 wave 2

Postponing VAT on Sales Credit Memo

Special function for the Sales Credit memo documents can be used. It is important to know (by §42 part 3b) of VAT Law 235/2004) the delivery date of the Sales Credit memo document. User can fill in the delivery date in the Posted Sales Credit Memo as the new VAT Date.
Sales VAT Postponed Account field has been added in the **VAT Posting Setup**. The **Postponed VAT** field was filled in the **VAT Entry**.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature is no longer mandatory by law from 2019 and will be deleted.	CU for 01.2021 release 2020 wave 2

Non-deductible VAT

This functionality is important due to §76 of VAT Law 235/2004 – the shortened claim to deduction. The functionality includes:

- extension of **VAT Posting Setup** – allow Non-deductible VAT.
- special Non-deductible VAT Setup – user can set up Non Deductible VAT % and applicability.
- user can see **VAT % (Non-deductible)**, **VAT Base (Non-deductible)**, **VAT Amount (Deductible)** in the **VAT Entry**.
- extension of VAT Statement functionality.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature will be not supported in extension-based Czech online tenants and will be deleted.	CU for 01.2021 release 2020 wave 2

VAT Registration in Other Countries

This functionality extends the possibility to work with VAT and allows the user to:

- set up the company as the VAT payer in the another EU country.
- set up exchange rates for **Fulfillment Country**.
- set up **Registration Country** for Customers and Vendors.
- set up **Registration Country Routes** for changing **VAT Bus. Posting Group**.
- define the **VAT Country** and the **VAT Fulfillment Country** on Documents.
- work with VAT Statement and VAT Settlement for fulfillments in another **VAT Fulfillment Country**.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature will be not supported in extension-based Czech online tenants and will be deleted.	CU for 01.2021 release 2020 wave 2

VAT Identifier

This feature extends **VAT Identifier** within **VAT Posting Setup**. Adds source table for **VAT Identifier** and allow specify additional legal texts for printing in VAT recapitulation on tax documents. The **VAT Identifier** field is the part of the **VAT Entry**.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	This feature was replaced by standard functionality of VAT Clauses and is no longer used and it will be deleted.	CU for 01.2021 release 2020 wave 2

Reverse Charge Statement

The amendment of VAT Law 235/2004 (§ 92a – 92e) introduced the VAT reverse charge calculation also for specific domestic fulfillments. A Reverse Charge Statement has to be submitted to the Tax Office electronically.

This feature allows user to:

- select combinations in **VAT Posting Setup** to include into the **Reverse Charge Statement**.
- set the Tariff Number Statement Code and Unit of Measures for **Reverse Charge Statement**.
- input all information on Sales and Purchase documents needed for electronic file submission.
- print **Reverse Charge Statement** and export statement data into the file for electronic submission.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature is no longer mandatory by law and will be deleted.	CU for 01.2021 release 2020 wave 2

Fixed Assets Classification - SKP codes

SKP code is a code from the Standard Production Classification register. The Standard Production Classification was established by the Czech Statistical Office. This code is used for Fixed Asset classification and categorizing to the tax depreciation group. New **SKP Code** table and **SKP Code field** on Fixed Asset were added.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature is no longer mandatory by law and will be deleted.	CU for 01.2021 release 2020 wave 2

Fixed Assets - Depreciation FA Appreciation From

This feature allows you calculate depreciation for additional FA appreciation on fully depreciated fixed asset.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature will be not supported in extension based Czech SaaS environment and will be deleted.	CU for 01.2021 release 2020 wave 2

Fixed Assets – item consumption for maintenance

Fixed Asset Maintenance functionality was improved by extending the functionality in Item Journal. New fields

FA No. and **Maintenance Code** added to **Item Journal Line** table. These fields are used if purpose of item consumption is Fixed Asset Maintenance.

When the Item Negative Adjustment is posted in Item Journal, the **Maintenance Ledger Entry** will be created and posted automatically, based on **Automatic Maintenance Posting** field in **Inventory Setup**. For manually posting and/or item costs adjusting to maintenance, the report **Adj. Maintenance-Item Entries** was created.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Removed	This feature is deprecated and no longer needed and as not mandatory it will be deleted.	CU for 01.2021 2019 release wave 2

Multiple Interest Rates

When you create finance charge terms and reminder terms, for delayed payment penalty, you can specify multiple interest rates so that the penalty fee is calculated from different interest rates in different periods.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Moved	The Multiple Interest Rates feature is no longer specific to Czech, so we have made it generally available in the standard product.	2020 release wave 2

Deletion of Posted Documents feature

The majority of legislation's requires companies to keep their electronic record indefinitely or for a specific period of time. In order to avoid situation of accidental deletion of Sales and Purchase documents, Business Central prevents users to do this.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	This Deletion of Posted Documents feature is no longer specific to Czech, so we have made it generally available in the standard product.	2019 release wave 2

Tax corrective documents for VAT

According to the VAT law amendment (Act No 235/2004 Coll.; amended with Act No 47/2011 Coll.), companies must correct the VAT base and amount when they are changed due to different reason (financial discount, payment discount and other corrections).

The printout of the correction document is named **Tax Correction Document**.

In the moment of paying the invoice with a defined payment discount, a Sales (or Service) Credit-Memo will be created. The document will reflect payment discount amount and calculated VAT Amount. The document can be printed in an ordinary way. The printout of the created Credit-Memo is a **Tax Corrective Document**. A Tax Corrective Document for payment discounts will be created if a payment discount is calculated based on an amount including VAT (it means the **Adjust for Payment Disc.** field is active in the **General Ledger Setup** form) and for those combinations which are active in the **Adjust for Payment Discount** fields in the **VAT Posting Setup**. In other cases payment discount will be calculated in an ordinary way.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Removed	This feature will be not supported in extension based Czech SaaS environment and will be deleted.	CU for 01.2021 release 2020 wave 2

Vendor Templates

New feature for creating templates for different group of vendors has been added - **Vendor Templates**. This feature copies similar functionality for customers – **Customer templates**. The Vendor Template feature makes creating vendors from contacts easy and decreases amount of mistakes. When the user creates a new Vendor, **Apply Template** function can be used. Some fields will be automatically filled in thanks to that.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Removed	This feature is deprecated and no longer needed and as not mandatory it will be deleted.	CU for 01.2021 2019 release wave 2

Item Charges Enhancements

This feature allows the user to set and check the usage of item charges in Sales and Purchase. You can set the following options:

- assign item charge for the usage only in Sales or only in Purchase.
- disable assigning to different source lines – Receipt Lines, Transfer Receipt Lines, Return Shipment Lines, Sales Shipment Lines, Return Receipt Lines.
- check if item charge is assigned at Receive/Shipment posting.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Removed	Lack of use. The business need that this functionality was introduced to cover is no longer relevant. Functionality will be deleted.	CU for 01.2021 2019 release wave 2

No. Series Enhancements

The majority of companies in the Czech Republic request the following improvements to be implemented in No. Series setup.

No. Series Mask

New field **Mask** for No. Series structure mask added to the **No. Series** table where the user defines a position structure of the number generated in this No. Series. This feature makes creating new No. Series lines for new fiscal year easy and decreases the amount of mistakes.

This **Mask** creates a new No. Series line using the new **No. Series Mask Generator** function.

No. Series Links

New **No. Series Link** table added. The table contains setup of No. Series links for various system functions.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Removed	Lack of use. The business need that this functionality was introduced to cover is no longer relevant. Functionality will be deleted.	CU for 01.2021 2019 release wave 2

Certificate Management

For Electronic Registration of Sales (EET) functionality was introduced Certificate Management functionality to allow storing and assignment of certificates in EET.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Moved	The Certificate Management feature is no longer specific to Czech, so we have made it generally available in the standard product.	2019 release wave 2

Packaging Tax Calculation

Companies selling or consuming goods on local market are liable of declaring package tax to the authorities. To do this Dynamics NAV enables users to:

- set package materials.
- collect data about sales and consumption of items with packages.
- calculate and report package tax for above mentioned transactions and items.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Removed	Lack of use. The business need that this functionality was introduced to cover is no longer relevant. Functionality will be deleted.	CU for 01.2021 2019 release wave 2

Small regulatory features and best practices

Following small regulatory features and local practices of Czech companies will be removed.

Dimension for VAT Entry

New field **Dimension Set ID** added to **VAT Ledger Entry** table.

Disable Cards Deleting

This feature enhance standard functionality to completely disallows the user to delete records in master data with entries (such as G/L accounts, Customers, Vendors, Items). This is driven by **Delete Card with Entries** field in **General Ledger Setup**.

Field for Full Description

In earlier versions, the character limit for **Description** and **Name** fields was 50. Therefore was a new field **Full Description** in all master data tables introduced to fulfill legal requirements for master data naming.

You can now enter up to 100 characters in all **Description** and **Name** fields across Business Central and it covers now the business need that this functionality was introduced.

Posted Warehouse Documents cannot be deleted

The majority of legislation's requires companies to keep their electronic record indefinitely or for a specific period of time. In order to avoid situation of accidental deletion of Warehouse documents, Business Central prevents users to do this.

Indivisible Unit of Measure

New field **Indivisible Unit** in **Unit of Measure** and **Item Unit of Measure** tables is added to avoid entering a decimal numbers in **Quantity** field in documents when Item should be posted in specific unit of measure.

Customer Template extension

New fields added to **Customer Template** to enhance data set populated in new customer card created from contact.

Quote Validity field on the Sales/Purchase Quote

New **Quote Validity** date field added to Sales/Purchase Quote documents to identify quote expiration.

This feature is replaced by Quote Validity feature introduced in standard product in 2019 release wave 1.

Original User ID on Sales/Purchase documents

New **Original User ID** field added to Sales/Purchase documents to identify creator of the document.

Check Balance in General Ledger Journal

Additional feature for disabling balance check by **Correction** field in General Ledger Journal posting.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN
Removed	Lack of use. The business need that this functionality was introduced to cover is no longer relevant. Functionality will be deleted.	CU for 01.2021 2019 release wave 2

Objects and Fields that are deleted in Dynamics 365 Business Central

The following list shows additional fields that are deleted as a result of the features that have been removed.

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETESTATE	OBSOLETEREASON
18	Customer	11793	Default Ship-to Address Code	Removed	Replaced by **Ship-to Code**
32	Item Ledger Entry	31065	Shipment Method Code	Pending	Merge to W1
36	Sales Header	11793	Quote Validity	Removed	Replaced by **Quote Valid Until Date**
49	Invoice Post. Buffer	11761	Description	Pending	Merge to W1
83	Item Journal Line	31065	Shipment Method Code	Pending	Merge to W1

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETESTATE	OBSOLETEREASON
86	Exch. Rate Adjmt. Reg.	11760	Starting Date	Pending	This field is not needed and it is not used anymore.
86	Exch. Rate Adjmt. Reg.	11761	Ending Date	Pending	This field is not needed and it is not used anymore.
86	Exch. Rate Adjmt. Reg.	11762	Running Date	Pending	This field is not needed and it is not used anymore.
98	General Ledger Setup	11793	Reg. No. Validation URL	Removed	This field has been replaced by Table 11757 **Reg. No. Srv Config.**
110	Sales Shipment Header	11700	Bank Account Code	Pending	This field is not needed and it is not used anymore.
110	Sales Shipment Header	11701	Bank Account No.	Pending	This field is not needed and it is not used anymore.
110	Sales Shipment Header	11702	Bank Branch No.	Pending	This field is not needed and it is not used anymore.
110	Sales Shipment Header	11707	IBAN	Pending	This field is not needed and it is not used anymore.
110	Sales Shipment Header	11708	SWIFT Code	Pending	This field is not needed and it is not used anymore.
110	Sales Shipment Header	11709	Bank Name	Pending	This field is not needed and it is not used anymore.
263	Intrastat Jnl. Line	31069	Shipment Method Code	Pending	Merge to W1
270	Bank Account	11703	Specific Symbol	Pending	Duplicated with Field 11701.

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETESTATE	OBSOLETEREASON
270	Bank Account	11735	User ID	Pending	This field is not needed and it is not used anymore.
270	Bank Account	11762	Direct Posting	Pending	This field is not needed and it is not used anymore.
287	Customer Bank Account	11700	Priority	Pending	This field is not needed and it should not be used.
270	Customer Bank Account	11762	Specific Symbol	Pending	This field is not needed and it should not be used.
270	Vendor Bank Account	11700	Priority	Pending	This field is not needed and it should not be used.
270	Vendor Bank Account	11762	Specific Symbol	Pending	This field is not needed and it should not be used.
325	VAT Posting Setup	31102	Insolvency Proceedings (p.44)	Removed	Replaced by **Corrections for Bad Receivable**
1200	Bank Export/Import Setup	11705	Default Folder Path	Removed	Folder path isn't supported to use in DownloadFromStream, UploadToStream, Download and Upload functions for web client
6660	Return Receipt Header	11700	Bank Account Code	Pending	This field is not needed and it is not used anymore.
6660	Return Receipt Header	11701	Bank Account No.	Pending	This field is not needed and it is not used anymore.

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETESTATE	OBSOLETEREASON
6660	Return Receipt Header	11702	Bank Branch No.	Pending	This field is not needed and it is not used anymore.
6660	Return Receipt Header	11707	IBAN	Pending	This field is not needed and it is not used anymore.
6660	Return Receipt Header	11708	SWIFT Code	Pending	This field is not needed and it is not used anymore.
6660	Return Receipt Header	11709	Bank Name	Pending	This field is not needed and it is not used anymore.
11761	Electronically Govern. Setup	30	Proxy Server	Pending	This field is not needed and it should not be used.
11761	Electronically Govern. Setup	31	Proxy User	Pending	This field is not needed and it should not be used.
11761	Electronically Govern. Setup	32	Proxy Password	Removed	Moved to **Service Password**
11761	Electronically Govern. Setup	33	Proxy Password Key	Pending	This field is not needed and it should not be used.
31101	VAT Control Report Line	44	Insolvency Proceedings (p.44)	Removed	Replaced by **Corrections for Bad Receivable**
31103	VAT Control Report Buffer	44	Insolvency Proceedings (p.44)	Removed	Replaced by **Corrections for Bad Receivable**
31101	Certificate CZ	All	All Fields	Removed	Moved to standard table 1262 **Isolated Certificate**

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the CZ version.

Different types of disposal and maintenance:

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5603	FA Setup	31043	FA Maintenance By Maint. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5603	FA Setup	31045	FA Disposal By Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5606	FA Posting Group	31042	Use Standard Disposal	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Posting Description:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
18	Customer	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
23	Vendor	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
38	Purchase Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
302	Finance Charge Memo Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11779	Fin. Charge Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
312	Purchases & Payables Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
313	Inventory Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5109	Purchase Header Archive	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5911	Service Mgt. Setup	11765	Posting Desc. Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
11785	Posting Description	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11786	Posting Desc. Parameter	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

General Journal Reconciliation:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
269	G/L Account Net Change	11760	Type	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Industry Classification:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
18	Customer	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
23	Vendor	31063	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
38	Purchase Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
79	Company Information	11793	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
110	Sales Shipment Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
120	Purch. Rcpt. Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
122	Purch. Inv. Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
124	Purch. Cr. Memo Hdr.	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5990	Service Shipment Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
6650	Return Shipment Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6660	Return Receipt Header	31065	Industry Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11791	Industry Code	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

VAT Rounding Improvements:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
98	General Ledger Setup	11765	Round VAT Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11766	VAT Coeff. Rounding Precision	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Postponing VAT on Sales Credit Memo:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
36	Sales Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
91	Gen. Journal Line	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
114	Sales Cr.Memo Header	11764	Postponed VAT Realized	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11777	Credit Memo Confirmation	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
325	VAT Posting Setup	11764	Sales VAT Postponed Account	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5911	Service Mgt. Setup	11777	Credit Memo Confirmation	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	11763	Postponed VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	11764	Postponed VAT Realized	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Non-deductible VAT:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
39	Purchase Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
39	Purchase Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
39	Purchase Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
49	Invoice Post. Buffer	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
49	Invoice Post. Buffer	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
49	Invoice Post. Buffer	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	11772	VAT Base LCY (Non Deduct.)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
81	Gen. Journal Line	11773	VAT Amount LCY (Non Deduct.)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11762	Statement Templ. Name Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11763	Statement Name Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
98	General Ledger Setup	11764	Statement Line No. Coeff.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
123	Purch. Inv. Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
123	Purch. Inv. Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
123	Purch. Inv. Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
125	Purch. Cr. Memo Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
125	Purch. Cr. Memo Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
125	Purch. Cr. Memo Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
242	Source Code Setup	11762	VAT Coefficient	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11770	Primary Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
290	VAT Amount Line	11765	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
290	VAT Amount Line	11766	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
290	VAT Amount Line	11767	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
325	VAT Posting Setup	11763	Non Deduct. VAT Corr. Account	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
325	VAT Posting Setup	11766	Allow Non Deductible VAT	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11731	Cash Document Line	602	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11731	Cash Document Line	603	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11731	Cash Document Line	604	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11736	Posted Cash Document Line	602	VAT % (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11736	Posted Cash Document Line	603	VAT Base (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11736	Posted Cash Document Line	604	VAT Amount (Non Deductible)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	1	VAT Bus. Posting Group	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
11784	Non Deductible VAT Setup	2	VAT Prod. Posting Group	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	3	From Date	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11784	Non Deductible VAT Setup	4	Non Deductible VAT %	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

VAT Registration in Other Countries:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
32	Item Ledger Entry	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
36	Sales Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
38	Purchase Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
38	Purchase Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
83	Item Journal Line	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
110	Sales Shipment Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
114	Sales Cr.Memo Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
120	Purch. Rcpt. Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
122	Purch. Inv. Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
122	Purch. Inv. Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
124	Purch. Cr. Memo Hdr.	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
124	Purch. Cr. Memo Hdr.	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	31061	Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	31062	Currency Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
261	Intrastat Jnl. Template	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
262	Intrastat Jnl. Batch	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5109	Purchase Header Archive	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5109	Purchase Header Archive	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5990	Service Shipment Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5990	Service Shipment Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5992	Service Invoice Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5992	Service Invoice Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5994	Service Cr.Memo Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5994	Service Cr.Memo Header	31061	Curr. Factor Perf. Country/Reg	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6650	Return Shipment Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6660	Return Receipt Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11762	Registration Country/Region	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11763	Registr. Country/Region Route	All	All fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11764	Perf. Country Curr. Exch. Rate	All	All fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31000	Sales Advance Letter Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31000	Sales Advance Letter Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31020	Purch. Advance Letter Header	31060	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31020	Purch. Advance Letter Header	31061	Perf. Country Currency Factor	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31066	VIES Declaration Header	80	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31100	VAT Control Report Header	15	Perform. Country/Region Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

VAT Identifier:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
98	General Ledger Setup	11771	Check VAT Identifier	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11767	VAT Identifier	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11768	VAT Identifier Translate	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Reverse Charge Statement:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31065	Stat. Reporting Setup	31090	Reverse Charge Nos.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31065	Stat. Reporting Setup	31095	Reverse Charge Auth. Emp. No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31065	Stat. Reporting Setup	31096	Rvrs. Chrg. Filled by Emp. No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31093	Reverse Charge Header	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31094	Reverse Charge Line	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Fixed Assets Classification - SKP codes:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5600	Fixed Asset	31044	SKP Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31043	SKP Code	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Fixed Assets – Depreciation FA Appreciation From:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5612	FA Depreciation Book	31040	Depr. FA Appreciation From	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Fixed Assets – item consumption for maintenance:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
32	Item Ledger Entry	31043	FA No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
32	Item Ledger Entry	31044	Maintenance Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
32	Item Ledger Entry	31045	Maintenance Amount	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
81	Gen. Journal Line	31070	Item Ledger Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
83	Item Journal Line	31043	FA No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
83	Item Journal Line	31044	Maintenance Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
210	Job Journal Line	31043	FA No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
210	Job Journal Line	31044	Maintenance Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
242	Source Code Setup	31041	Maintenance Adjustment	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
313	Inventory Setup	31042	Automatic Maintenance Posting	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5600	Fixed Asset	31042	Deprec. Book Code (Mainten.)	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5621	FA Journal Line	31040	Item Ledger Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5625	Maintenance Ledger Entry	31040	Item Ledger Entry No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Tax corrective documents for VAT:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
36	Sales Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
37	Sales Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
112	Sales Invoice Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
113	Sales Invoice Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
115	Sales Cr.Memo Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11780	Pmt.Disc. Tax Corr.Doc. No.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
311	Sales & Receivables Setup	11772	Reas.Cd. on Tax Corr.Doc.Mand.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11773	Pmt.Disc.Tax Corr.Doc. Nos.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11774	Copy As Tax Corr. Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
311	Sales & Receivables Setup	11775	Reason Code For Payment Disc.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5107	Sales Header Archive	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5108	Sales Line Archive	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5900	Service Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5902	Service Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5911	Service Mgt. Setup	11772	Reas.Cd. on Tax Corr.Doc.Mand.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5911	Service Mgt. Setup	11775	Reason Code For Payment Disc.	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5992	Service Invoice Header	11762	Tax Corrective Document	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5993	Service Invoice Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5995	Service Cr.Memo Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
6661	Return Receipt Line	11762	Reason Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Vendor Templates:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
11794	Vendor Template	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Item Charges Enhancements:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
313	Inventory Setup	31076	Check Item Charge Pst.Group	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31070	Use Ledger Entry Dimensions	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5800	Item Charge	31071	Sales Only	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31072	Purchase Only	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31073	Disable Receipt Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31074	Disable Transfer Receipt Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31075	Disable Return Shipment Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31076	Disable Sales Shipment Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31077	Disable Return Receipt Lines	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5800	Item Charge	31078	Assignment on Receive/Shipment	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

No. Series Enhancements:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
308	No. Series	11790	Mask	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
308	No. Series	11791	No. Series Link Exists	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11730	Receiving Wh. No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11797	Shipping No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11798	Receiving No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5740	Transfer Header	11799	Shipping Wh. No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
11799	No. Series Link	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Packaging Tax Calculation:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
31070	Package Material	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
31071	Item Package Material	All	All Fields	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Dimension for VAT Entry:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
254	VAT Entry	11771	Global Dimension 1 Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11772	Global Dimension 2 Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
254	VAT Entry	11773	Dimension Set ID	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Disable Cards Deleting:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
98	General Ledger Setup	11792	Delete Card with Entries	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Field for Full Description:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
15	G/L Account	11792	Full Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
18	Customer	11792	Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
23	Vendor	11792	Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
27	Item	11792	Full Description	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
260	Tariff Number	11792	Full Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
260	Tariff Number	11793	Full Name ENG	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5050	Contact	11792	Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5079	Marketing Setup	11792	Inherit Registered Name	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5600	Fixed Asset	11792	Full Description	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5718	Nonstock Item	11792	Full Description	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Customer Template extension:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
5105	Customer Template	11790	No. Series	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5105	Customer Template	11791	Language Code	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Indivisible Unit of Measure:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
204	Unit of Measure	31070	Indivisible Unit	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).
5404	Item Unit of Measure	31070	Indivisible Unit	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

Check Balance in General Ledger Journal:

TABLENO	TABLENAME	NO.	FIELDNAME	OBSOLETEREASON
80	Gen. Journal Template	11761	Not Check Correction	Will be removed in a later release (marked as ObsoleteState:Removed in 01.2021).

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Czech Local Functionality

Deprecated Features in the Dutch Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for the Netherlands that has been removed from Business Central, made available from a new page or report, made available to one or more additional countries, or replaced by a new feature.

Checking Totals for Purchase Invoices and Purchase Credit Memos

If the total amount on a purchase document does not match the total amount from the purchase lines, you can find out why by letting Business Central calculate the total amount, total base amount, total VAT amount, and total amount including VAT for the purchase lines. The totals display in fields at the bottom of the **Purchase Invoice** or **Purchase Credit Memo** pages.

By default, Business Central does not show these totals. To display them, on the **Purchases & Payables Setup** page, choose the **Show Totals on Purch. Inv./CM.** check box.

> **NOTE**
> To use this feature, your purchase invoices or purchase credit memos must have at least one purchase line, and a quantity. Additionally, when you turn on this feature Business Central recalculates totals on all purchase invoices and credit memos. Depending on the number of documents, this can take some time.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The feature to check totals for purchase invoices and credit memos is no longer specific to the Netherlands, so we have made it generally available in the standard product.

Standard Sales and Purchase Codes

If you often need to create sales and purchase lines with similar information, you can set up standard codes representing sales and purchase lines that you can then insert on recurring sales and purchase documents, for example, for recurring replenishment orders.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	The feature has been removed from the Dutch version because it is generally available in the standard product.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the NL version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
312	Purchases & Payables Setup	11312	Show Totals on Purch. Inv./CM.	Will be removed in a later release.

See Also

Upgrading to Dynamics 365 Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Netherlands Local Functionality

Deprecated Features in the Finnish Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Finland that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Multiple Interest Rates

When you create finance charge terms and reminder terms, for delayed payment penalty, you can specify multiple interest rates so that the penalty fee is calculated from different interest rates in different periods.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Multiple Interest Rates feature is no longer specific to Finland, so we have made it generally available in the standard product.

Fields marked as ObsoleteState:Pending

Currently, no fields are marked as ObsoleteState:Pending in the FI version.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Finland Local Functionality

Deprecated Features in the German Version of Dynamics 365 Business Central

2/17/2021 • 3 minutes to read • Edit Online

This topic lists and describes the local functionality for Germany that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Batch Print Sales and Purchase Documents When Posting

When you batch post sales or purchase documents, you can select to also print the related reports.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Batch Print when Posting feature is no longer specific to Germany, so we have made it generally available in the standard product.

Copy Existing Items to Create New Items

When you add a new item, to save time, you can use the **Copy Item** function to copy an existing item to use as a template for a new item.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Copy Item feature is no longer specific to Germany, so we have made it generally available in the standard product.

Physical Inventory Order

You can take inventory of your items by using the **Physical Inventory Order** and **Physical Inventory Recording** pages. The physical inventory order contains data for planning, realizing, and analyzing physical inventory. The physical inventory recording contains the items and quantities to be counted and forms the basis of the print-out to be used in the warehouse.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Physical Inventory Order feature is no longer specific to Germany, so we have made it generally available in the standard product.

Blanket Order Archiving and Document Line Tracking

You can archive and delete blanket sales and purchase orders. You can view documents that are related to sales order lines and purchase order lines, including from archived order lines. Related documents that you can track include quotes, shipments, receipts, and blanket orders. This helps you to identify documents used to process orders.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	Blanket Order Archiving and Document Line Tracking features are no longer specific to Germany, so we have made them generally available in the standard product.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the DE version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
242	Source Code Setup	5005350	Phys. Invt. Order	Will be removed in a later release.
257	VAT Statement Name	11000	Sales VAT Adv. Notification	Will be removed in a later release.
313	Inventory Setup	5005352	Posted Phys. Inv. Order Nos.	Will be removed in a later release.
313	Inventory Setup	5005350	Phys. Inv. Order Nos.	Will be removed in a later release.
11011	Sales VAT Advance Notification	All	All	Will be removed in a later release.
11012	Transmission Log Entry	All	All	Will be removed in a later release.
11013	Electronic VAT Decl. Setup	All	All	Will be removed in a later release.
11014	Certificate	All	All	Will be removed in a later release.
5005350	Phys. Inventory Order Header	All	All	Will be removed in a later release.
5005351	Phys. Inventory Order Line	All	All	Will be removed in a later release.
5005352	Phys. Invt. Recording Header	All	All	Will be removed in a later release.
5005353	Phys. Invt. Recording Line	All	All	Will be removed in a later release.
5005354	Post. Phys. Invt. Order Header	All	All	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5005355	Posted Phys. Invt. Order Line	All	All	Will be removed in a later release.
5005356	Posted Phys. Invt. Rec. Header	All	All	Will be removed in a later release.
5005357	Posted Phys. Invt. Rec. Line	All	All	Will be removed in a later release.
5005358	Phys. Inventory Comment Line	All	All	Will be removed in a later release.
5005359	Posted Phys. Invt. Track. Line	All	All	Will be removed in a later release.
5005360	Phys. Invt. Tracking Buffer	All	All	Will be removed in a later release.
5005361	Expect. Phys. Inv. Track. Line	All	All	Will be removed in a later release.
5005362	Post. Exp. Ph. In. Track. Line	All	All	Will be removed in a later release.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Germany Local Functionality

Deprecated Features in the Icelandic Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Iceland that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Icelandic Tax Regulations of Conditional Discounts

The local tax regulation of conditional discounts feature enables you to issue a credit memo if a conditional discount is given to a customer. The payment for a conditional discount must be made within a specified period.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	This feature is no longer specific to Iceland, so we have made it generally available in the standard product. There is a field for number series for credit invoices on the **Sales and Receivables Setup** form, and a field on the **Customer Ledger Entry** table to link the appropriate entries to a credit invoice.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the IS version.

TABLE ID	FIELD ID	COMMENTS
21	10900	Will be removed in a later release.
311	10900	Will be removed in a later release.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Iceland Local Functionality

Deprecated Features in the Italian Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Italy that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Report Trade with Customers and Vendors in Blocked Countries/Regions

You must submit a periodic report of transactions with customers and vendors in certain countries/regions that the Italian government has identified in a block list.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	The functionality for blocked countries/regions has been removed from the Italian version.

Multiple Interest Rates

When you create finance charge terms and reminder terms, for delayed payment penalty, you can specify multiple interest rates so that the penalty fee is calculated from different interest rates in different periods.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Multiple Interest Rates feature is no longer specific to Italy, so we have made it generally available in the standard product.

Fields marked as ObsoleteState:Pending

Currently, no fields are marked as ObsoleteState:Pending in the IT version.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Italy Local Functionality

Deprecated Features in the Mexican Version of Microsoft Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for the Mexico that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Customer Statement Report

Shows a list of financial transactions for a selected customer statements for a given period of time. For example, use the report as part of your payment collection process.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Standard Statement report (report 1316) provides the same capabilities as the Customer Statement report (report 10072), plus the ability to customize layouts in Word, and it is available for all countries.	2019 release wave 2

Aged Accounts Payable Report

Shows a list of aged remaining balances for each vendor for a given period of time.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Aged Accounts Payable report (report 322) provides the same capabilities as the (local) Aged Accounts Payable (report 10085) and it is available for all countries.	2019 release wave 2

Aged Accounts Receivable Report

Shows a list of aged remaining balances for each customer for a given period of time.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Aged Accounts Receivable report (report 120) provides the same capabilities as the (local) Aged Accounts Receivable (report 10040) and it is available for all countries.	2019 release wave 2

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the MX version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	10004	SAT Certificate Thumbprint	Will be removed in a later release.
10000	PAC Web Service	22	Certificate Thumbprint	Will be removed in a later release.

See Also

Upgrading Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Mexican Local Functionality in Business Central

Deprecated Features in the Norwegian Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Norway that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Multiple Interest Rates

When you create finance charge terms and reminder terms, for delayed payment penalty, you can specify multiple interest rates so that the penalty fee is calculated from different interest rates in different periods.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Multiple Interest Rates feature is no longer specific to Norway, so we have made it generally available in the standard product.

Paper Sources and Tray Numbers

When printing Norwegian sales documents, you can set up different tray numbers and paper sources on the first, last, and other pages.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	The feature is not used.

Fields marked as ObsoleteState:Pending

Currently, no fields are marked as ObsoleteState:Pending in the NO version.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Norway Local Functionality

Deprecated Features in the Swedish Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for Sweden that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Multiple Interest Rates

When you create finance charge terms and reminder terms, for delayed payment penalty, you can specify multiple interest rates so that the penalty fee is calculated from different interest rates in different periods.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Multiple Interest Rates feature is no longer specific to Sweden, so we have made it generally available in the standard product.

Fields marked as ObsoleteState:Pending

Currently, no fields are marked as ObsoleteState:Pending in the SE version.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Sweden Local Functionality

Deprecated Features in the Swiss Version of Dynamics 365 Business Central

2/17/2021 • 3 minutes to read • Edit Online

This topic lists and describes the local functionality for Switzerland that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Batch Print Sales and Purchase Documents When Posting

When you batch post sales or purchase documents, you can select to also print the related reports.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Batch Print when Posting feature is no longer specific to Switzerland, so we have made it generally available in the standard product.

Copy Existing Items to Create New Items

When you add a new item, to save time, you can use the **Copy Item** function to copy an existing item to use as a template for a new item.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Copy Item feature is no longer specific to Switzerland, so we have made it generally available in the standard product.

Physical Inventory Order

You can take inventory of your items by using the **Physical Inventory Order** and **Physical Inventory Recording** pages. The physical inventory order contains data for planning, realizing, and analyzing physical inventory. The physical inventory recording contains the items and quantities to be counted and forms the basis of the print-out to be used in the warehouse.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Physical Inventory Order feature is no longer specific to Switzerland, so we have made it generally available in the standard product.

Blanket Order Archiving and Document Line Tracking

You can archive and delete blanket sales and purchase orders. You can view documents that are related to sales order lines and purchase order lines, including from archived order lines. Related documents that you can track include quotes, shipments, receipts, and blanket orders. This helps you to identify documents used to process orders.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	Blanket Order Archiving and Document Line Tracking features are no longer specific to Switzerland, so we have made them generally available in the standard product.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the CH version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
4	Currency	3010541	ISO Currency Code	Will be removed in a later release.
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
98	General Ledger Setup	11011	Sales VAT Advance Notif. Nos.	Will be removed in a later release.
98	General Ledger Setup	11004	Post Pmt.Disc Tol. to Pmt.Disc	Will be removed in a later release.
242	Source Code Setup	5005350	Phys. Invt. Order	Will be removed in a later release.
257	VAT Statement Name	11000	Sales VAT Adv. Notification	Will be removed in a later release.
311	Sales & Receivables Setup	11501	Line Amt. Round LCY	Will be removed in a later release.
313	Inventory Setup	5005350	Phys. Inv. Order Nos.	Will be removed in a later release.
313	Inventory Setup	5005352	Posted Phys. Inv. Order Nos.	Will be removed in a later release.
11011	Sales VAT Advance Notification	All	All	Will be removed in a later release.
11012	Transmission Log Entry	All	All	Will be removed in a later release.
11013	Electronic VAT Decl. Setup	All	All	Will be removed in a later release.
11014	Certificate	All	All	Will be removed in a later release.
5005350	Phys. Inventory Order Header	All	All	Will be removed in a later release.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
5005351	Phys. Inventory Order Line	All	All	Will be removed in a later release.
5005352	Phys. Invt. Recording Header	All	All	Will be removed in a later release.
5005353	Phys. Invt. Recording Line	All	All	Will be removed in a later release.
5005354	Post. Phys. Invt. Order Header	All	All	Will be removed in a later release.
5005355	Posted Phys. Invt. Order Line	All	All	Will be removed in a later release.
5005356	Posted Phys. Invt. Rec. Header	All	All	Will be removed in a later release.
5005357	Posted Phys. Invt. Rec. Line	All	All	Will be removed in a later release.
5005358	Phys. Inventory Comment Line	All	All	Will be removed in a later release.
5005359	Posted Phys. Invt. Track. Line	All	All	Will be removed in a later release.
5005360	Phys. Invt. Tracking Buffer	All	All	Will be removed in a later release.
5005361	Expect. Phys. Inv. Track. Line	All	All	Will be removed in a later release.
5005362	Post. Exp. Ph. In. Track. Line	All	All	Will be removed in a later release.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
Switzerland Local Functionality

Deprecated Features in the UK Version of Dynamics 365 Business Central

2/17/2021 • 4 minutes to read • Edit Online

This topic lists and describes the local functionality for the United Kingdom that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Accounting periods and system calendar

If your fiscal year is different than the calendar, you can measure your fiscal period in other units of time, such as months or quarters. To do this, you set up system calendars and accounting periods.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	Lack of use. Additionally, there are standard features for accounting periods that provide most of the same functionality as the UK accounting periods.

Create and export a Bankers' Automated Clearing Service file

You can use Bankers' Automated Clearing Service (BACS) to process financial transactions electronically. To do so, you must export vendor payments to a BACS file using the Export BACS option.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	This banking format standard is no longer used. This functionality is now covered by extensions such as the Envestnet Yodlee Bank Feeds, AMC Banking, and various other formats.

Non-invoiced stock reports

For month-end reconciliation and auditing, you can use the **Stock Received Not Invoiced** report to view stock that is received but not yet invoiced, and the **Stock Shipped Not Invoiced** report to see stock that has been shipped but not yet invoiced.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The functionality for the Shipped, Non-Invoiced Sales Orders and the Received, Not Invoiced Purchase Order reports are no longer specific to the UK, so we have made them generally available as views for sales orders and purchase orders. The views are available in the Navigation Pane as Shipped Not Invoiced and Shipped Not Received options under Sales Orders and Purchase Orders, respectively.

Print Unposted Sales and Unposted Purchase reports

The Unposted Sales and Unposted Purchase reports let you print a list of sales and purchase documents that are not yet posted.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Unposted Sales and Unposted Purchase reports are now available from the Navigation Pane as views under Sales Orders and Purchase Orders.

Other VAT reports

You can use the following reports for VAT reporting:

- **Day Book VAT Entry** - Displays the daily total for VAT entries for a specific period.
- **Day Book Cust. Ledger Entry** - Displays the daily total for customer ledger entries for a specific period.
- **Day Book Vendor Ledger Entry** - Displays the daily total for vendor ledger entries for a specific period.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	These VAT-related reports are no longer specific to the UK, so we have made them generally available in the standard product.

Specify the Supply Type on documents

You can specify supply types such as sales, loan, exchange, hire, lease, rental, sales on commission, on tax invoices. To do this, you must update the codes and names of the supply types in the **Types of Supply** window.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	Lack of use. The business need that this functionality was introduced to cover is no longer relevant.

Multiple interest rates

When you create finance charge terms and reminder terms, for delayed payment penalty, you can specify multiple interest rates so that the penalty fee is calculated from different interest rates in different periods.

MOVED, REMOVED, OR REPLACED?	WHY?
Moved	The Multiple Interest Rates feature is no longer specific to the UK, so we have made it generally available in the standard product.

Direct Sales Details and Direct Purchase Details reports

You can view headers with order numbers and descriptions from sales and purchase documents, and filter or select data for these reports based on the general ledger account number, document number, and posting date.

MOVED, REMOVED, OR REPLACED?	WHY?
Removed	Lack of use.

Objects and Fields that are deleted in Dynamics 365 Business Central

Table 10505 has been deleted. The following list shows additional fields that are deleted as a result of the features that have been removed.

TABLE ID	FIELD ID	COMMENTS
23	10550	Deleted.
81	10550	Deleted.
81	10551	Deleted.
81	10552	Deleted.
81	10553	Deleted.
271	10550	Deleted.
312	10550	Deleted.
312	10551	Deleted.
334	10505	Deleted.
363	10550	Deleted.
7118	10505	Deleted.
7152	10550	Deleted.

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the UK version.

TABLE ID	FIELD ID	COMMENTS
18	10500	Will be removed in a later release.
23	10500	Will be removed in a later release.
36	10501	Will be removed in a later release.
38	10501	Will be removed in a later release.
112	10501	Will be removed in a later release.
114	10501	Will be removed in a later release.
122	10501	Will be removed in a later release.
124	10501	Will be removed in a later release.
5107	10501	Will be removed in a later release.
5109	10501	Will be removed in a later release.

The following tables are marked as ObsoleteState:Pending in the UK version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
10533	MTD-Liability	All	All	Will be removed in a later release.
10534	MTD-Payment	All	All	Will be removed in a later release.
10535	MTD-Return Details	All	All	Will be removed in a later release.

See Also

Upgrading to Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete
United Kingdom Local Functionality

Deprecated Features in the United States Version of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic lists and describes the local functionality for the United States that has been removed from Business Central, made available from a new page or report, or replaced by a new feature.

Customer Statement Report

Shows a list of financial transactions for a selected customer statements for a given period of time. For example, use the report as part of your payment collection process.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Standard Statement report (report 1316) provides the same capabilities as the Customer Statement report (report 10072), plus the ability to customize layouts in Word, and it is available for all countries.	2019 release wave 2

Aged Accounts Payable Report

Shows a list of aged remaining balances for each vendor for a given period of time.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Aged Accounts Payable report (report 322) provides the same capabilities as the (local) Aged Accounts Payable (report 10085) and it is available for all countries.	2019 release wave 2

Aged Accounts Receivable Report

Shows a list of aged remaining balances for each customer for a given period of time.

MOVED, REMOVED, OR REPLACED?	WHY?	WHEN?
Replaced	The Aged Accounts Receivable report (report 120) provides the same capabilities as the (local) Aged Accounts Receivable (report 10040) and it is available for all countries.	2019 release wave 2

Fields marked as ObsoleteState:Pending

The following fields are marked as ObsoleteState:Pending in the US version.

TABLE ID	TABLE NAME	FIELD ID	FIELD NAME	COMMENTS
98	General Ledger Setup	10004	SAT Certificate Thumbprint	Will be removed in a later release.
10000	PAC Web Service	22	Certificate Thumbprint	Will be removed in a later release.

See Also

Upgrading Business Central
Upgrading the Application Code
Deprecated Fields, and Fields Marked as Obsolete

Legal Resources for Business Central On-Premises

2/17/2021 • 2 minutes to read • Edit Online

This page provides links to legal information for Business Central On-Premises.

Software License Terms

- Dynamics 365 Business Central on-premises- <language>.pdf

Third Party Notices

- Dynamics Business Central On Prem Third Party Notice.pdf

AL Language

- AL Language - Terms of Use.pdf

Envestnet Yodlee - Bank Feeds

- Envestnet Yodlee - Bank Feeds Terms and Conditions.pdf

GetAddressIO

- GetAddressIO-MBS - External Components License Agreement Summary.pdf

See Also

Legal Resources for Business Central Online
Microsoft Online Service Terms (OST)
Developer and Administration Help for Microsoft Dynamics 365 Business Central

Deployment of Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

The topics in the Deployment section are intended to help a partner prepare a Business Central solution for a customer online or on-premises.

Take prospects and customers online

You can give prospects a quick introduction to Business Central by asking them to get a free trial, and by showing them the apps in AppSource, for example. You can also create and give powerful demos that will attract prospects in your market space. With Business Central online, data is stored in the Microsoft cloud, removing the need to install SQL Server locally, for example. So for you as a reseller, deployments of Business Central online are taken care of for you so that you can focus on your prospects and customers.

Get set up for selling Business Central online

If your organization has a background as resellers of Dynamics NAV or other on-premises products, you have to get set up in the Microsoft Partner Center. For more information, see Get Started as a Reseller of Business Central Online.

For more information about reseller readiness for Business Central, see Build Your Business on Dynamics 365 Business Central.

Give powerful demos

You can create a trial environment based on the Business Central content pack in cdx.transform.microsoft.com.

For more information, see Preparing Demonstration Environments of Dynamics 365 Business Central.

Managing Business Central online

For more information about administering online tenants, see Administration of Business Central Online.

When to choose on-premises deployment

There can be many reasons to prefer to deploy Business Central on-premises rather than using online deployments.

Key Features of Setup for On-Premises Deployments

With Business Central Setup, you can:

- Install different components on different computers.
- Choose from a selection of predefined installation options, or create your own custom list of components and options to install.
- Preconfigure components before installation.
- Create, save, or load setup configuration files that capture your selection of components and configuration information.

You use setup to install software and to create custom deployments that you can distribute to different users across a company.

Installation Notes

- Before installing Business Central components on a computer, you must remove (uninstall) any previous

versions.

- All components must be from the same version and build of Business Central for the software to run correctly.
- If you have either SQL Server 2000 or Microsoft SQL Server Desktop Engine (MSDE) installed on a computer where you want to install Business Central, then you must remove it before you begin installing. The presence of either of these database products causes a setup error.

Managing Business Central on-premises

For more information about administering on-premises deployments, see Administration of Business Central On-Premises.

Configuring the Help Experience

Part of your configuration is to specify where to look up the Help for the solution. For on-premises deployments, you can choose to install the legacy Help Server, for example. For more information, see Configuring the Help Experience.

See Also

Administration of Business Central Online
Get Started as a Reseller of Business Central Online
Upgrading to Business Central
Product and Architecture Overview
System Requirements 2019 release wave 2
System Requirements April '19

Features not implemented in on-premises deployments of Dynamics 365 Business Central

2/17/2021 • 3 minutes to read • Edit Online

This topic lists features that are available in Business Central but not in on-premises deployments. The topic is divided into two sections:

- The first section lists features that are available under very specific circumstances in on-premises deployments.
- The second section lists features that are not intended for use with on-premises deployments. There are no plans to implement these features.

Features that require specific circumstances

The following features are not available in all on-premises deployments because they require specific circumstances.

FEATURE	DESCRIPTION
Read/write data with Excel add-in	The Excel add-in that enables update of data requires Azure Active Directory as the authentication mechanism.
Excel financial reports	The Excel add-in that is used with the predefined Excel-based financial reports requires Azure Active Directory as the authentication mechanism.
Coversheets for contact management	The integration with Word to create the coversheets requires Azure Active Directory as the authentication mechanism.
Built-in Power BI reports and charts	The integration with Power BI requires Azure Active Directory as the authentication mechanism.
Built-in Power Automate Management	You can use the built-in workflows in on-premises deployments of Business Central, provided that you connect to Power Automate using Azure Active Directory as the authentication mechanism. This can be done using the Azure Active Directory Assisted Setup guide in Business Central. Microsoft Azure and Microsoft Power Automate require Azure Active Directory authentication; however, your Business Central on-premises deployment does not have to use Azure Active Directory as the general authentication mechanism.
Built-in web services	A number pages and queries are exposed as web services. However, the default endpoint must be manually updated before the web services can be consumed.
Outlook add-in	The Outlook add-in requires Dynamics NAV User/Password or Azure Active Directory as the authentication mechanism.

FEATURE	DESCRIPTION
Standard REST API	Business Central contains new standard REST APIs. However, on-premises deployments cannot be reached through Microsoft Graph or the common endpoint, `https://api.businesscentral.dynamics.com/v1.0/api/beta`. Instead, you must connect directly to the on-premises deployment, just as when you connect to web services.
Sales and Inventory Forecast	This functionality requires an Azure Machine Learning subscription.
Image Analyzer	This functionality requires an Computer Vision service.
Cortana Intelligence in Cash Flow Forecast	This functionality requires an Azure Machine Learning subscription.

Features not intended for use in on-premises deployments

The following features are not intended for use in on-premises deployments. There are no plans to implement these features in on-premises deployments.

FEATURE	DESCRIPTION
Default Power BI reports	Automatic deployment and configuration of Power BI reports is not supported in on-premises deployments of Business Central.
Bulk Invoicing from Microsoft Bookings	Integration with the Bookings app that is available in certain Microsoft 365 subscriptions is not supported.
Create workflow from Power Automate	Power Automate does not integrate with on-premises workflow functionality. You cannot create new workflows based on existing Power Automate templates in on-premises deployments of Business Central.
Sandbox environments	The sandbox environment that you can use to develop extensions against for the new developer experience cannot connect to an on-premises deployment. For more information, see Get started with the Container Sandbox Development Environment.
In-product search	In online deployments of Business Central, Tell Me, the in-product search, also searches in content on the docs.microsoft.com site. For on-premises deployments, this is not supported.
Late Payment Prediction	The Late Payment Prediction functionality is not supported in on-premises deployments of Business Central.
Use the company hub to manage work across multiple companies.	Integration with the company hub is not supported in on-premises deployments of Business Central.
Inviting the external accountant	Integration with the now deprecated Dynamics 365 — Accountant Hub and the existing company hub is not supported in on-premises deployments of Business Central.

See Also

System Requirements
How to: Create a Sandbox Environment

System Requirements for Dynamics 365 Business Central 2020 Release Wave 2

2/17/2021 • 8 minutes to read • Edit Online

The following sections list the minimum hardware and software requirements to install and run Business Central on-premises (version 17). **Minimum** means that later versions (such as SP1, SP2, or R2 versions) of a required software product are also supported.

> **NOTE**
> Business Central Setup installs some software if it's not already present in the target computer. For more information, see the "Additional Information" section for each component.

CLIENT COMPONENTS

Web Client

The following table shows the minimum system requirements for the Business Central Web client on-premises.

SPECIFICATION	REQUIREMENT
Supported browsers	Recommended browsers: • New Microsoft Edge, latest version • Google Chrome for Windows, latest version • Mozilla Firefox for Windows, latest version • Safari for macOS, latest version Other supported browsers: • Internet Explorer 11, latest version • Microsoft Edge Legacy, latest version Cookies and JavaScript must be enabled in the browser.

Business Central Mobile App

The following table shows the minimum system requirements for the Business Central Mobile App.

For the latest information, see the app in the Windows Store, App Store, or Google Play.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 S, Home, Pro, Enterprise, or Education (32-bit and 64-bit editions). • Android 6.0 or higher (tablet and phone). • iOS 10.0 or higher (iPad and iPhone).

SPECIFICATION	REQUIREMENT
Additional hardware	• 1-GB RAM for Android and Windows.
Additional information	• Device diagonal screen size 7" for tablets. • Screen resolution 960 × 510 for tablets. • Device diagonal screen size 4" for phones. • Screen resolution 854 x 480 for phones.

Microsoft Office Applications

Business Central on-premises offers various features that require Office apps to be available on client devices. The following table shows the minimum system requirements for the features.

SPECIFICATION	REQUIREMENT
Excel	• Sending data to Excel requires Microsoft Office 2019, Excel for web, or Excel mobile app for iOS or Android™. • Editing in Excel using the Excel Add-In requires Microsoft Office 2019 or Excel for web.
Word	• Microsoft Office 2019, Word for web, or Word mobile app for iOS or Android™.
Outlook	Please see Business Inbox in Microsoft Outlook.
Additional software	• A third-party telephony or VoIP app such as Skype or Microsoft Teams is required for placing calls from Business Central.

AL Development

The following table shows the minimum system requirements for customizing or extending Business Central using the AL language in Visual Studio Code.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows Server 2019 • Windows Server, version 1809 or later • Windows 10 For information about the supported versions and their lifecycles, see Windows lifecycle fact sheet.
Required software	• Visual Studio Code • AL language extension

SPECIFICATION	REQUIREMENT
Hardware resources	• Hard disk space: 500 MB. • CPU: four cores minimum • Memory: 16 GB for development only. 16 GB for developing and locally deploying small extensions (<1000 objects>). 32-64 GB for developing and locally deploying large extensions (>1000 objects).
Reports	• For creating and editing RDL report layouts: ◦ Report Builder for SQL Server 2016, or ◦ Visual Studio 2017 with Microsoft Rdlc Report Designer for Visual Studio installed. • For creating and editing Word report layouts: ◦ Word 2016 or later

For more information, see Getting Started with AL.

SERVER COMPONENTS

Business Central Server

The following table shows the minimum system requirements for Business Central Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition) • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 (Datacenter, Standard) For information about the supported versions and their lifecycles, see Windows lifecycle fact sheet.
Hardware resources	• Hard disk space: 500 MB. • Memory: 2 GB.
Dynamics 365 Sales integration	• Windows Identity Framework. For a list of supported Dynamics 365 Sales versions, see Microsoft Dynamics 365 for Sales Integration Requirements.
Additional software	• Microsoft .NET Framework 4.8 • Windows PowerShell 4.0.

SPECIFICATION	REQUIREMENT

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer:
	  ◦ Microsoft .NET Framework 4.8
	  ◦ Windows Identity Framework.

Business Central Web Server Components

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition) • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 (Datacenter, Standard) For information about the supported versions and their lifecycles, see Windows lifecycle fact sheet.
Web server	• Internet Information Services 10
Additional software	• Microsoft .NET Framework 4.8 • Windows PowerShell 4.0.

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer.
	  ◦ Microsoft .NET Core 1.0 Windows Server Hosting. This software is installed by Business Central Setup if not already present.
	  ◦ Microsoft .NET Framework 4.8
	  ◦ Internet Information Services 10 is installed with the required features enabled.
	• For more information about configuring IIS, see Configuring IIS

Business Central Database Components for SQL Server

The following table shows the minimum system requirements for Business Central database components for SQL Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition) • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 (Datacenter, Standard) For information about the supported versions and their lifecycles, see Windows lifecycle fact sheet.

SPECIFICATION	REQUIREMENT
Hardware resources	For more information, see Hardware and Software Requirements for Installing SQL Server. From this page, you can also access requirements for other versions of SQL Server.
SQL Server	• Microsoft SQL Server 2019 Express, Standard, or Enterprise. • Microsoft SQL Server 2017 Express, Standard, or Enterprise. • Microsoft SQL Server 2016 Express, Standard, or Enterprise (Service Pack 2 or later). • Azure SQL Database Managed Instance, Elastic Pool, or Single Database.
Service Packs and Cumulative Updates	Unless explicitly stated, all released Service Packs and Cumulative Updates of the above Microsoft SQL Server versions are supported. it's recommended to always be on the latest released Service Pack and Cumulative Update.
Additional information	Business Central Setup installs the following software if it's not already present on the target computer: • SQL Server 2016 Express (64-bit edition). If the operating system on the target computer doesn't support SQL Server 2016 Express, Setup displays a pre-requisite warning. In this case, you should exit Setup. Then, update the operating system on the computer to one that does support SQL Server 2016 Express and run Setup again.

Business Central Help Server Requirements

The following table shows the minimum system requirements for the Business Central Help Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit editions) • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 (Datacenter, Standard) For information about the supported versions and their lifecycles, see Windows lifecycle fact sheet.
Hardware resource	• Hard disk space: 500 MB. • Memory: 2 GB.
Web server	• Internet Information Services 10.
Additional software	• Microsoft .NET Framework 4.8

SPECIFICATION	REQUIREMENT

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer.
	  ◦ Microsoft .NET Framework 4.8
	  ◦ Internet Information Services 10.
	• Windows Search must be enabled on the computer that you install the Business Central Help Server on.

ADDITIONAL COMPONENTS AND FEATURES

Automated Data Capture System

The following table shows the minimum system requirements for Automated Data Capture System (ADCS) for Business Central.

SPECIFICATION	REQUIREMENT
Additional software	• MSXML version 6.0. • Telnet or Microsoft Windows HyperTerminal. • VT100 Plug-in for each computer on which you install ADCS. • Microsoft Loopback Adapter.
Additional information	• VT100 Plug-in acts as a virtual Telnet server. • Starting with Business Central 2020 release wave 1, the VT100 Plug-in for ADCS is no longer included on installation media. You can use VT100 Plug-in from previous releases. However, remember to test that it works properly. • HyperTerminal is no longer included with Windows.

Business Inbox in Microsoft Outlook

The following table shows the minimum system requirements for using Business Central as your business inbox in Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2019 or later • Outlook on the web • Outlook Web App for iPad • Outlook Web App for iPhone • Outlook Web App for Android™.
Supported Exchange Servers	• Exchange Online • Exchange Server 2019 In deployments that use Exchange Server, the Exchange PowerShell endpoint must be accessible by Business Central Server.

SPECIFICATION	REQUIREMENT
Supported Authentication	• The Business Central Server must be configured to run with NavUserPassword, ACS, or AAD Credentials Type. Also, the Business Central Web client must be configured for Secure Sockets Layer (SSL).
Supported Browsers	• When using Outlook on the web, your computer must be running a supported browser listed in the Business Central Web client Requirements.
Supported Operating Systems	• When using Outlook Web App for iPad, iPhone, or Android™, your mobile device must use a supported operating system that's listed in the Business Central Mobile App section.

Microsoft Outlook Legacy Add-In

The following table shows the minimum system requirements for the Business Central legacy Add-In for Outlook for synchronization with Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2019.
Supported Exchange Servers	• Exchange Server 2019 • Exchange Online.

Microsoft Dynamics 365 for Sales Integration

The following table shows the product version requirements for integrating Business Central with Dynamics 365 Sales, and the versions in which users can view the availability of items in Business Central from Dynamics 365 Sales.

SALES/DYNAMICS NAV/BUSINESS CENTRAL	2016/UPDATE 1/ONLINE	SALES ENTERPRISE (V8.X)	SALES ENTERPRISE AND SALES PROFESSIONAL (V9.X)
Dynamics NAV 2016	Supported ***	Supported ***	Supported ***
Dynamics NAV 2017	Supported *	Supported *	Supported *
Dynamics NAV 2018	Supported *	Supported *	Supported *
Business Central (online)	Not supported	Supported *	Supported *
Business Central (on-premises)	Supported *	Supported *	Supported *

Legend:

- "*" item availability capability is supported.
- "**" integration solution can be installed from the Dynamics NAV 2016 DVD, but viewing item availability isn't supported.
- "***" viewing item availability isn't supported

> **NOTE**
> AD, IFD and Claims authentication types are supported for the 2016 on-premises version of Dynamics 365 Sales. OAuth and Office 365 authentication are supported for the 2016 Update 1 and online versions of Dynamics 365 Sales. For more information about authentication types, see Connection strings in XRM tooling to connect to Dynamics 365.

Business Central as an App for SharePoint

The following table shows the minimum system requirements for Business Central as an App for SharePoint.

SPECIFICATION	REQUIREMENT
Supported SharePoint servers	• SharePoint Server 2016 • SharePoint Online.

See Also

Welcome to the Developer and IT-Pro Help for Business Central
Product and Architecture Overview
Deployment

System Requirements for Dynamics 365 Business Central 2020 Release Wave 1

2/17/2021 • 9 minutes to read • Edit Online

The following sections list the minimum hardware and software requirements to use or connect to Business Central online, and to install and run Business Central on-premises (version 16). **Minimum** means that later versions (such as SP1, SP2, or R2 versions) of a required software product are also supported.

> **NOTE**
> Business Central Setup installs some software if it's not already present in the target computer. For more information, see the "Additional Information" section for each component.

Client Components

Web Client Requirements

The following table shows the minimum system requirements for the Business Central Web client on-premises.

SPECIFICATION	REQUIREMENT
Supported browsers	Recommended browsers: • New Microsoft Edge for Windows • Google Chrome for Windows (81.0 or later) • Mozilla Firefox for Windows (75.0 or later) • Safari for macOS (75.0 or later) Other supported browsers: • Internet Explorer 11 • Microsoft Edge Legacy Cookies and JavaScript must be enabled in the browser.
Business inbox in Outlook	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Sending data to Excel	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Editing in Excel using the Excel Add-in	• Excel 2019, Excel 2016, or Excel Online. For more information, see Exporting Your Business Data to Excel.
SharePoint Online links	• Microsoft Office 2019, Microsoft Office 2016, or Microsoft 365.

SPECIFICATION	REQUIREMENT
Printing reports to Excel or Word	• Microsoft Office 2019, Microsoft Office 2016, or Microsoft 365.
Additional information	If you experience problems using the Business Central Web client, you can try to turn off browser tools, such as translator tools that may run in the background.

Business Central Tablet Client and Phone Client (in a Browser) Requirements

The following table shows the minimum system requirements for the Business Central tablet client and Business Central phone client running in a desktop browser when used for development and testing purposes.

SPECIFICATION	REQUIREMENT
Server component	Identical to the Business Central Web client.
Supported browsers	Identical to the Business Central Web client.

Business Central Mobile App Requirements

The following table shows the minimum system requirements for the Business Central Mobile App.

For the latest information, see the app in the Windows Store, App Store, or Google Play.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 S, Home, Pro, Enterprise, or Education (32-bit and 64-bit editions). • Android 6.0 or higher (tablet and phone). • iOS 10.0 or higher (iPad and iPhone).
Additional hardware	• 1-GB RAM for Android and Windows.
Additional software	• A third-party telephony or VoIP app such as Skype is required for placing calls from Business Central. • A third-party email program such as Outlook is required for sending emails from Business Central. • Microsoft Office 2019, Office 2016, or Microsoft 365 is required for sending data to Microsoft Excel or to Microsoft Word.
Additional information	• Device diagonal screen size 7" for tablets. • Screen resolution 960 × 510 for tablets. • Device diagonal screen size 4" for phones. • Screen resolution 854 x 480 for phones.

AL Development Requirements

The following table shows the minimum system requirements for customizing or extending Business Central using the AL language in Visual Studio Code.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows Server 2019 • Windows Server, version 1809 or later • Window Server 2016. • Windows 10 - supported versions.
Required software	• Visual Studio Code • AL language extension
Hardware resources	• Hard disk space: 500 MB. • CPU: four cores minimum • Memory: 16 GB for development only. 16 GB for developing and locally deploying small extensions (<1000 objects>). 32-64 GB for developing and locally deploying large extensions (>1000 objects).
Reports	• For creating and editing RDL report layouts: ◦ Report Builder for SQL Server 2016, or ◦ Visual Studio 2017 with Microsoft Rdlc Report Designer for Visual Studio installed. • For creating and editing Word report layouts: ◦ Word 2016 or later

For more information, see Getting Started with AL.

Server Components

Business Central Server Requirements

The following table shows the minimum system requirements for Business Central Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter. .
Hardware resources	• Hard disk space: 500 MB. • Memory: 2 GB.
Dynamics 365 Sales integration	• Windows Identity Framework. For a list of supported Dynamics 365 Sales versions, see Microsoft Dynamics 365 for Sales Integration Requirements.

SPECIFICATION	REQUIREMENT
Additional software	• Microsoft .NET Framework 4.8 • Windows PowerShell 4.0.

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer:
	  ◦ Microsoft .NET Framework 4.8
	  ◦ Windows Identity Framework.

Business Central Web Server Components Requirements

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter. .
Web server	• Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0.
Additional software	• Microsoft .NET Framework 4.8 • Windows PowerShell 4.0.

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer.
	  ◦ Microsoft .NET Core 1.0 Windows Server Hosting. This software is installed by Business Central Setup if not already present.
	  ◦ Microsoft .NET Framework 4.8
	  ◦ Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0. The version depends on the operating system. Internet Information Server is installed with the required features enabled.
	• For more information about configuring IIS, see Configuring IIS

Business Central Database Components for SQL Server Requirements

The following table shows the minimum system requirements for Business Central database components for SQL Server.

SPECIFICATION	REQUIREMENT

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resources	For more information, see Hardware and Software Requirements for Installing SQL Server. From this page, you can also access requirements for other versions of SQL Server.
SQL Server	• Microsoft SQL Server 2019 Express, Standard, or Enterprise. • Microsoft SQL Server 2017 Express, Standard, or Enterprise. • Microsoft SQL Server 2016 Express, Standard, or Enterprise (Service Pack 1 or later). • Azure SQL Database Managed Instance, Elastic Pool, or Single Database.
Service Packs and Cumulative Updates	Unless explicitly stated, all released Service Packs and Cumulative Updates of the above Microsoft SQL Server versions are supported. it's recommended to always be on the latest released Service Pack and Cumulative Update.
Additional information	Business Central Setup installs the following software if it's not already present on the target computer: • SQL Server 2016 Express (64-bit edition). If the operating system on the target computer doesn't support SQL Server 2016 Express, Setup displays a pre-requisite warning. In this case, you should exit Setup. Then, update the operating system on the computer to one that does support SQL Server 2016 Express and run Setup again.

Business Central Help Server Requirements

The following table shows the minimum system requirements for the Business Central Help Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit editions). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resource	• Hard disk space: 500 MB. • Memory: 2 GB.

SPECIFICATION	REQUIREMENT
Web server	• Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0.
Additional software	• Microsoft .NET Framework 4.8

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer.
	  ◦ Microsoft .NET Framework 4.8
	  ◦ Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0. The version depends on the operating system. it's installed with the required features enabled.
	• Windows Search must be enabled on the computer that you install the Business Central Help Server on.

Additional Components and Features

Automated Data Capture System Requirements

The following table shows the minimum system requirements for Automated Data Capture System (ADCS) for Business Central.

SPECIFICATION	REQUIREMENT
Additional software	• MSXML version 6.0. • Telnet or Microsoft Windows HyperTerminal. • VT100 Plug-in for each computer on which you install ADCS. • Microsoft Loopback Adapter.
Additional information	• HyperTerminal is no longer included with Windows. For more information, see What happened to HyperTerminal? in the Windows Help. • VT100 Plug-in acts as a virtual Telnet server.

> **NOTE**
> Starting with Business Central 2020 release wave 1, the VT100 Plug-in for ADCS is no longer included on installation media. You can use VT100 Plug-in from previous releases. However, remember to test that it works properly.

Business Inbox in Microsoft Outlook Requirements

The following table shows the minimum system requirements for using Business Central as your business inbox in Outlook.

SPECIFICATION	REQUIREMENT

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2016 or later • Outlook Web App • Outlook Web App for iPad • Outlook Web App for iPhone • Outlook Web App for Android.
Supported Exchange Servers	• Exchange Online • Exchange Server 2019 • Exchange Server 2016 In deployments that use Exchange Server, the Exchange PowerShell endpoint must be accessible by Business Central Server.
Supported Authentication	• The Business Central Server must be configured to run with NavUserPassword, ACS, or AAD Credentials Type. Also, the Business Central Web client must be configured for Secure Sockets Layer (SSL).
Supported Browsers	• When using the Outlook Web App (Microsoft Outlook Web App), your computer must be running a supported browser listed in the Business Central Web client Requirements.
Supported Operating Systems	• When using Outlook Web App for iPad, iPad, or Android, your mobile device must use a supported Operating System listed in Business Central Mobile App Requirements.

Microsoft Outlook Add-In Requirements

The following table shows the minimum system requirements for the Business Central Add-In for Outlook for synchronization with Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2019 • Outlook 2016
Supported Exchange Servers	• Exchange Server 2019 • Exchange Server 2016 • Exchange Online.

Microsoft Dynamics 365 for Sales Integration Requirements

The following table shows the product version requirements for integrating Business Central with Dynamics 365 Sales, and the versions in which users can view the availability of items in Business Central from Dynamics 365 Sales.

SALES/DYNAMICS NAV/BUSINESS CENTRAL	2015/UPDATE 1/ONLINE	2016/UPDATE 1/ONLINE	SALES ENTERPRISE (V8.X)	SALES ENTERPRISE AND SALES PROFESSIONAL (V9.X)
Dynamics NAV 2016	Supported ***	Supported ***	Supported ***	Supported ***
Dynamics NAV 2017	Supported **	Supported *	Supported *	Supported *
Dynamics NAV 2018	Supported **	Supported *	Supported *	Supported *
Business Central (online)	Not supported	Not supported	Supported *	Supported *
Business Central (on-premises)	Supported **	Supported *	Supported *	Supported *

Legend:

- "*" item availability capability is supported.
- "**" integration solution can be installed from the Dynamics NAV 2016 DVD, but viewing item availability isn't supported.
- "***" viewing item availability isn't supported

> **NOTE**
> AD, IFD and Claims authentication types are supported for the 2015 and 2016 on-premises versions of Dynamics 365 Sales. OAuth and Microsoft 365 authentication are supported for the 2015, 2015 Update 1, and 2016 Update 1 online versions of Dynamics 365 Sales. For more information about authentication types, see Connection strings in XRM tooling to connect to Dynamics 365.

Business Central as an App for SharePoint Requirements

The following table shows the minimum system requirements for Business Central as an App for SharePoint.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows Server 2019 Standard, Essentials, or Datacenter. • Windows Server 2016 Standard, Essentials, or Datacenter.
Additional software	• SharePoint 2013 Service Pack 1. • SharePoint Online.

See Also

Welcome to the Developer and IT-Pro Help for Business Central
Product and Architecture Overview
Deployment

System Requirements for Dynamics 365 Business Central 2019 Release Wave 2

2/17/2021 • 9 minutes to read • Edit Online

The following sections list the minimum hardware and software requirements to use or connect to Business Central online, and to install and run Business Central on-premises (version 15). **Minimum** means that later versions (such as SP1, SP2, or R2 versions) of a required software product are also supported.

> **NOTE**
> Business Central Setup installs some software if it's not already present in the target computer. For more information, see the "Additional Information" section for each component.

Client Components

Web Client Requirements

The following table shows the minimum system requirements for the Business Central Web client on-premises.

SPECIFICATION	REQUIREMENT
Supported browsers	Recommended browsers: • New Microsoft Edge for Windows • Google Chrome for Windows (77.0 or later) • Mozilla Firefox for Windows (69.0 or later) • Safari for macOS (12.0 or later) Other supported browsers: • Internet Explorer 11 • Microsoft Edge Legacy Cookies and JavaScript must be enabled in the browser.
Business inbox in Outlook	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Sending data to Excel	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Editing in Excel using the Excel Add-in	• Excel 2019, Excel 2016, or Excel Online. For more information, see Exporting Your Business Data to Excel.
SharePoint Online links	• Microsoft Office 2019, Microsoft Office 2016, or Microsoft 365.

SPECIFICATION	REQUIREMENT
Printing reports to Excel or Word	• Microsoft Office 2019, Microsoft Office 2016, or Microsoft 365.
Additional information	If you experience problems using the Business Central Web client, you can try to turn off browser tools, such as translator tools that may run in the background.

Business Central Tablet Client and Phone Client (in a Browser) Requirements

The following table shows the minimum system requirements for the Business Central tablet client and Business Central phone client running in a browser when used for development and testing purposes.

SPECIFICATION	REQUIREMENT
Server component	Identical to the Business CentralWeb client.
Supported browsers	The following desktop browsers are supported: • Microsoft Edge • Internet Explorer 11 (build 11.0.9600.17239) for Windows 10. • Google Chrome 72.0 for Windows. • Mozilla Firefox 65.0 for Windows. • Safari 10.0 for macOS. Cookies and JavaScript must be enabled in the browser.

Business Central Mobile App Requirements

The following table shows the minimum system requirements for the Business Central Mobile App.

For the latest information, see the app in the Windows Store, App Store, or Google Play.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 S, Home, Pro, Enterprise, or Education (32-bit and 64-bit editions). • Android 6.0 or higher (tablet and phone). • iOS 10.0 or higher (iPad and iPhone).
Additional hardware	• 1-GB RAM for Android and Windows.
Additional software	• A third-party telephony or VoIP app such as Skype is required for placing calls from Business Central. • A third-party email program such as Outlook is required for sending emails from Business Central. • Microsoft Office 2019, Office 2016, or Microsoft 365 is required for sending data to Microsoft Excel or to Microsoft Word.

SPECIFICATION	REQUIREMENT
Additional information	• Device diagonal screen size 7" for tablets. • Screen resolution 960 × 510 for tablets. • Device diagonal screen size 4" for phones. • Screen resolution 854 x 480 for phones.

AL Development Requirements

The following table shows the minimum system requirements for customizing or extending Business Central using the AL language in Visual Studio Code.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows Server 2019 • Windows Server, version 1809 or later • Window Server 2016. • Windows 10 - supported versions.
Required software	• Visual Studio Code • AL language extension
Hardware resources	• Hard disk space: 500 MB. • CPU: Four cores minimum. • Memory: 16 GB for development only. 16 GB for developing and locally deploying small extensions (<1000 objects>). 32-64 GB for developing and locally deploying large extensions (>1000 objects).
Reports	• For creating and editing RDL report layouts: ◦ Report Builder for SQL Server 2016, or ◦ Visual Studio 2017 with Microsoft Rdlc Report Designer for Visual Studio installed. • For creating and editing Word report layouts: ◦ Word 2016 or later.

For more information, see Getting Started with AL.

Server Components

Business Central Server Requirements

The following table shows the minimum system requirements for Business Central Server.

SPECIFICATION	REQUIREMENT

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resources	• Hard disk space: 2 GB. • Memory: 2 GB for only running the application. 16 GB for publishing the application to the server.
Dynamics 365 Sales integration	• Windows Identity Framework. For a list of supported Dynamics 365 Sales versions, see Microsoft Dynamics 365 for Sales Integration Requirements.
Additional software	• Microsoft .NET Framework 4.7.2. or 4.8 • Windows PowerShell 4.0.
Additional information	• Business Central Setup installs the following software if it's not already present on the target computer: ◦ Microsoft .NET Framework 4.7.2. ◦ Windows Identity Framework.

Business Central Web Server Components Requirements

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Web server	• Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0.
Additional software	• Microsoft .NET Framework 4.7.2 or 4.8. • Windows PowerShell 4.0.

SPECIFICATION	REQUIREMENT

Additional information	• Business Central Setup installs the following software if it's not already present on the target computer.
	  ◦ Microsoft .NET Core 1.0 Windows Server Hosting. This version is installed by Business Central Setup if not already present.
	  ◦ Microsoft .NET Framework 4.7.2.
	  ◦ Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0, depending in the operating system, with the required features enabled.
	• For more information about configuring IIS, see Configuring IIS.

Business Central Database Components for SQL Server Requirements

The following table shows the minimum system requirements for Business Central database components for SQL Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resources	For more information, see Hardware and Software Requirements for Installing SQL Server. From this page, you can also access requirements for other versions of SQL Server.
SQL Server	• Microsoft SQL Server 2019 Express, Standard, or Enterprise. • Microsoft SQL Server 2017 Express, Standard, or Enterprise. • Microsoft SQL Server 2016 Express, Standard, or Enterprise. • Azure SQL Database Managed Instance, Elastic Pool, or Single Database.
Service Packs and Cumulative Updates	Unless explicitly stated, all released Service Packs and Cumulative Updates of the above Microsoft SQL Server versions are supported. It's recommended to always be on the latest released Service Pack and Cumulative Update.

SPECIFICATION	REQUIREMENT
Additional information	Business Central Setup installs the following software if it's not already present on the target computer: • SQL Server 2016 Express (64-bit edition). If the operating system on the target computer does not support SQL Server 2016 Express, Setup displays a pre-requisite warning. In this case, you should exit Setup and then update the operating system on the computer to one that does support SQL Server 2016 Express. Then run Setup again.

Business Central Help Server Requirements

The following table shows the minimum system requirements for the Business Central Help Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit editions). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resource	• Hard disk space: 500 MB. • Memory: 2 GB.
Web server	• Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0.
Additional software	• Microsoft .NET Framework 4.7.2 or 4.8.
Additional information	• Business Central Setup installs the following software if it's not already present on the target computer. ◦ Microsoft .NET Framework 4.7.2. ◦ Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0. depending on the operating system, with the required features enabled. • Windows Search must be enabled on the computer that you install the Business Central Help Server on.

Additional Components and Features

Automated Data Capture System Requirements

The following table shows the minimum system requirements for Automated Data Capture System (ADCS) for Business Central.

SPECIFICATION	REQUIREMENT
Additional software	• MSXML version 6.0. • Telnet or Microsoft Windows HyperTerminal. • VT100 Plug-in for each computer on which you install ADCS. • Microsoft Loopback Adapter.
Additional information	• HyperTerminal is no longer included with Windows. For more information, see What happened to HyperTerminal? in the Windows Help. • VT100 Plug-in acts as a virtual Telnet server.

Business Inbox in Microsoft Outlook Requirements

The following table shows the minimum system requirements for using Business Central as your business inbox in Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2016 or later • Outlook Web App (OWA) • OWA for iPad • OWA for iPhone • OWA for Android.
Supported Exchange Servers	• Exchange Online • Exchange Server 2019 • Exchange Server 2016 In deployments that use Exchange Server, the Exchange PowerShell endpoint must be accessible by Business Central Server.
Supported Authentication	• The Business Central Server must be configured to run with NavUserPassword, ACS, or AAD Credentials Type. Also, the Business Central Web client must be configured for Secure Sockets Layer (SSL).
Supported Browsers	• When using the Outlook Web App (OWA), your computer must be running a supported browser listed in the Business Central Web client Requirements.
Supported Operating Systems	• When using OWA for iPad, OWA for iPad, or OWA for Android, your mobile device must use a supported Operating System listed in Business Central Mobile App Requirements.

Microsoft Outlook Add-In Requirements

The following table shows the minimum system requirements for the Business Central Add-In for Outlook for synchronization with Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2019 • Outlook 2016
Supported Exchange Servers	• Exchange Server 2019 • Exchange Server 2016 • Exchange Online.

Microsoft Dynamics 365 for Sales Integration Requirements

The following table shows the product version requirements for integrating Business Central with Dynamics 365 Sales, and the versions in which users can view the availability of items in Business Central from Dynamics 365 Sales.

SALES/DYNAMICS NAV/BUSINESS CENTRAL	2015/UPDATE 1/ONLINE	2016/UPDATE 1/ONLINE	SALES ENTERPRISE (V8.X)	SALES ENTERPRISE AND SALES PROFESSIONAL (V9.X)
Dynamics NAV 2016	Supported ***	Supported ***	Supported ***	Supported ***
Dynamics NAV 2017	Supported **	Supported *	Supported *	Supported *
Dynamics NAV 2018	Supported **Supported *	Supported *	Supported *	
Business Central (online)	Not supported	Not supported	Supported *	Supported *
Business Central (on-premises)	Supported **Supported *	Supported *	Supported *	

Legend:

- "*" item availability capability is supported.
- "**" integration solution can be installed from the Dynamics NAV 2016 DVD, but viewing item availability is not supported.
- "***" viewing item availability is not supported

> **NOTE**
>
> AD, IFD and Claims authentication types are supported for the 2015 and 2016 on-premises versions of Dynamics 365 Sales. OAuth and Microsoft 365 authentication are supported for the 2015, 2015 Update 1, and 2016 Update 1 online versions of Dynamics 365 Sales. For more information on authentication types, see Connection strings in XRM tooling to connect to Dynamics 365.

Business Central as an App for SharePoint Requirements

The following table shows the minimum system requirements for Business Central as an App for SharePoint.

SPECIFICATION	REQUIREMENT

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows Server 2019 Standard, Essentials, or Datacenter. • Windows Server 2016 Standard, Essentials, or Datacenter.
Additional software	• SharePoint 2013 Service Pack 1. • SharePoint Online.

See Also

Welcome to the Developer and IT-Pro Help for Business Central
Product and Architecture Overview
Deployment

System Requirements for Dynamics 365 Business Central April '19

2/17/2021 • 11 minutes to read • Edit Online

The following sections list the minimum hardware and software requirements to use or connect to Business Central online, and to install and run Business Central on-premises (version 14). **Minimum** means that later versions (such as SP1, SP2, or R2 versions) of a required software product are also supported.

> **NOTE**
> Business Central Setup installs some software if it isn't already present in the target computer. For more information, see the "Additional Information" section for each component.

Client Components

Browser Requirements

The following table shows the minimum system requirements for using Business Central in a browser.

SPECIFICATION	REQUIREMENT
Supported browsers	Recommended browsers: • New Microsoft Edge for Windows • Google Chrome for Windows (77.0 or later) • Mozilla Firefox for Windows (69.0 or later) • Safari for macOS (12.0 or later) Other supported browsers: • Internet Explorer 11 • Microsoft Edge Legacy Cookies and JavaScript must be enabled in the browser.
Business inbox in Outlook	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Sending data to Excel	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Editing in Excel using the Excel Add-in	• Excel 2019, Excel 2016, or Excel Online. For more information, see Exporting Your Business Data to Excel.
SharePoint Online links	• Microsoft Office 2019, Microsoft Office 2016, or Microsoft 365.

SPECIFICATION	REQUIREMENT
Printing reports to Excel or Word	• Microsoft Office 2019, Microsoft Office 2016, or Microsoft 365.
Additional information	If you experience problems using the Business Central Web client, you can try to turn off browser tools, such as translator tools that may run in the background.

Business Central Mobile App Requirements

The following table shows the minimum system requirements for the Business Central Mobile App.

For the latest information, see the app in the Windows Store, App Store, or Google Play.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 S, Home, Pro, Enterprise, or Education (32-bit and 64-bit editions). • Android 6.0 or higher (tablet and phone). • iOS 10.0 or higher (iPad and iPhone).
Additional hardware	• 1-GB RAM for Android and Windows.
Additional software	• A third-party telephony or VoIP app such as Skype is required for placing calls from Business Central. • A third-party email program such as Outlook is required for sending emails from Business Central. • Microsoft Office 2019, Office 2016, or Microsoft 365 is required for sending data to Microsoft Excel or to Microsoft Word.
Additional information	• Device diagonal screen size 7" for tablets. • Screen resolution 960 × 510 for tablets. • Device diagonal screen size 4" for phones. • Screen resolution 854 x 480 for phones.

AL Development Requirements

The following table shows the minimum system requirements for customizing or extending Business Central using the AL language in Visual Studio Code.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows Server 2019 • Windows Server, version 1809 or later • Window Server 2016. • Windows 10 - supported versions.

SPECIFICATION	REQUIREMENT
Required software	• Visual Studio Code • AL language extension
Hardware resources	• Hard disk space: 500 MB. • CPU: Four cores minimum • Memory: 16 GB for development only. 32 GB for developing and deploying locally. 64 GB for developing large apps.

For more information, see Getting Started with AL.

Dynamics NAV Client connected to Business Central Requirements

The following table shows the minimum system requirements for using the Dynamics NAV Client connected to Business Central.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (32-bit and 64-bit editions). **Important:** Windows 10 S isn't supported. • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resources	• Hard disk space: 200 MB. • Memory: 1 GB.

Reports	• For editing RDLC report layouts:
	  ◦ Report Builder for SQL Server 2016, or
	  ◦ Visual Studio 2017 with Microsoft Rdlc Report Designer for Visual Studio installed.
	• For editing Word layouts:
	  ◦ Microsoft Word 2016 or later

Outlook client integration and mail merge	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Import and export with Microsoft Excel and Office XML, and SharePoint links	• Microsoft 365, Microsoft Office 2019, or Microsoft Office 2016.
Editing in Excel using the Excel Add-in	• Excel 2019 or Excel 2016. For more information, see Exporting Your Business Data to Excel. For Business Central on-premises, see Setting up the Excel Add-In for Editing Data since the same steps apply to Business Central on-premises.

SPECIFICATION	REQUIREMENT
Email logging	• Active Directory and Microsoft Exchange Server 2019 or Exchange Server 2016. • Microsoft Exchange Online, or Exchange Online as part of a Microsoft 365 subscription.
Additional software	• Microsoft .NET Framework 4.7.2 or 4.8
Additional information	• Business Central Setup installs the following software if it isn't already present in the target computer: ◦ Microsoft .NET Framework 4.7.2. • The Dynamics NAV Client is available in a 32-bit version and 64-bit version. On a 32-bit Windows operating system, the 32-bit version is run. On a 64-bit Windows operating system, the 64-bit version is run by default; however, you can also run the 32-bit version if it's required. • Business Central Setup can only install the Excel Add-in if Excel is present on the target computer. • Outlook synchronization isn't supported on 64-bit versions of Office.

Dynamics NAV Development Environment Requirements

The following table shows the minimum system requirements for the Dynamics NAV Development Environment.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (32-bit and 64-bit editions). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resources	• Hard disk space: 200 MB. • Memory: 1 GB.

Reports	• For creating and editing RDL report layouts:
	  ◦ Report Builder for SQL Server 2016, or
	  ◦ Visual Studio 2017 with Microsoft Rdlc Report Designer for Visual Studio installed.
	• For creating and editing Word report layouts:
	  ◦ Word 2016 or later

Additional software	• Microsoft .NET Framework 4.7.2 or 4.8

SPECIFICATION	REQUIREMENT

Additional information	• Business Central Setup installs the following software if it isn't already present in the target computer:
	  ◦ Microsoft .NET Framework 4.7.2.
	  ◦ SQL Server Native Client 11.0.
	  ◦ Report Builder for SQL Server 2016. This version isn't installed if a version of SQL Server Report Builder or Microsoft Visual Studio is already present on the target computer
	• If the development environment and Business Central Server are on the same computer, then only a 64-bit operating system is supported.

Server Components (on-premises)

Business Central Server Requirements

The following table shows the minimum system requirements for Business Central Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 • Windows Server, version 1809 or later • Windows Server 2016 Standard, Essentials, or Datacenter. .
Hardware resources	• Hard disk space: 500 MB. • Memory: 2 GB.
Dynamics 365 Sales integration	• Windows Identity Framework. For a list of supported Dynamics 365 Sales versions, see Microsoft Dynamics 365 for Sales Integration Requirements.
Additional software	• Microsoft .NET Framework 4.7.2 or 4.8 • Windows PowerShell 4.0.

Additional information	• Business Central Setup installs the following software if it isn't already present on the target computer:
	  ◦ Microsoft .NET Framework 4.7.2.
	  ◦ Windows Identity Framework.

Business Central Web Server Components Requirements

SPECIFICATION	REQUIREMENT

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter. .
Web server	• Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0.
Additional software	• Microsoft .NET Framework 4.7.2 or 4.8. • Windows PowerShell 4.0.

Additional information	• Business Central Setup installs the following software if it isn't already present on the target computer.
	  ◦ Microsoft .NET Core 1.0 Windows Server Hosting. This version is installed by Business Central Setup if not already present.
	  ◦ Microsoft .NET Framework 4.7.2.
	  ◦ Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0, depending in the operating system, with the required features enabled.
	• For more information about configuring IIS, see Configuring IIS

Business Central Database Components for SQL Server Requirements

The following table shows the minimum system requirements for Business Central database components for SQL Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit edition). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resources	For more information, see Hardware and Software Requirements for Installing SQL Server. From this page, you can also access requirements for other versions of SQL Server.

SPECIFICATION	REQUIREMENT
SQL Server	• Microsoft SQL Server 2019 Express, Standard, or Enterprise. • Microsoft SQL Server 2017 Express, Standard, or Enterprise. • Microsoft SQL Server 2016 Express, Standard, or Enterprise. • Azure SQL Database Managed Instance, Elastic Pool, or Single Database.
Service Packs and Cumulative Updates	Unless explicitly stated, all released Service Packs and Cumulative Updates of the above Microsoft SQL Server versions are supported. It is recommended to always be on the latest released Service Pack and Cumulative Update.
Additional information	Business Central Setup installs the following software if it isn't already present on the target computer: • SQL Server 2016 Express (64-bit edition). If the operating system on the target computer does not support SQL Server 2016 Express, Setup displays a pre-requisite warning. In this case, you should exit Setup and then update the operating system on the computer to one that does support SQL Server 2016 Express. Then run Setup again.

Business Central Help Server Requirements

The following table shows the minimum system requirements for the Business Central Help Server.

SPECIFICATION	REQUIREMENT
Supported operating systems	• Windows 10 Pro, Enterprise, or Education (64-bit editions). • Windows Server 2019 (Datacenter, Standard) • Windows Server, version 1809 or later (Datacenter, Standard) • Windows Server 2016 Standard, Essentials, or Datacenter.
Hardware resource	• Hard disk space: 500 MB. • Memory: 2 GB.
Web server	• Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0.
Additional software	• Microsoft .NET Framework 4.7.2 or 4.8.

SPECIFICATION	REQUIREMENT

Additional information	• Business Central Setup installs the following software if it isn't already present on the target computer.
	  ◦ Microsoft .NET Framework 4.7.2.
	  ◦ Internet Information Server 10, Internet Information Server 8.5, or Internet Information Server 8.0. depending on the operating system, with the required features enabled.
	• Windows Search must be enabled on the computer that you install the Business Central Help Server on.

Additional Components and Features

Automated Data Capture System Requirements

The following table shows the minimum system requirements for Automated Data Capture System (ADCS) for Business Central.

SPECIFICATION	REQUIREMENT
Additional software	• MSXML version 6.0. • Telnet or Microsoft Windows HyperTerminal. • VT100 Plug-in for each computer on which you install ADCS. • Microsoft Loopback Adapter.
Additional information	• HyperTerminal is no longer included with Windows. • VT100 Plug-in acts as a virtual Telnet server.

Requirements for using Business Central on-premises as your Business Inbox in Microsoft Outlook

The following table shows the minimum system requirements for using Business Central on-premises as your business inbox in Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2016 or later • Outlook Web App (OWA) • OWA for iPad • OWA for iPhone • OWA for Android.
Supported Exchange Servers	• Exchange Online • Exchange Server 2019 • Exchange Server 2016 In deployments that use Exchange Server, the Exchange PowerShell endpoint must be accessible by Business Central Server.

SPECIFICATION	REQUIREMENT
Supported Authentication	• The Business Central Server must be configured to run with NavUserPassword, ACS, or A<ure AD as the authentication type. Also, the Business Central Web client must be configured for Secure Sockets Layer (SSL).
Supported Browsers	• When using the Outlook Web App (OWA), your computer must be running a supported browser listed in the Business Central Web client Requirements.
Supported Operating Systems	• When using OWA for iPad, OWA for iPad, or OWA for Android, your mobile device must use a supported Operating System listed in Business Central Mobile App Requirements.

Microsoft Outlook Add-In Requirements

The following table shows the minimum system requirements for the Business Central Add-In for Outlook for synchronization with Outlook.

SPECIFICATION	REQUIREMENT
Supported Outlook Applications	• Outlook 2019 • Outlook 2016
Supported Exchange Servers	• Exchange Server 2019 • Exchange Server 2016 • Exchange Online.

Microsoft Dynamics 365 for Sales Integration Requirements

The following table shows the product version requirements for integrating Business Central with Dynamics 365 Sales, and the versions in which users can view the availability of items in Business Central from Dynamics 365 Sales.

SPECIFICATION	REQUIREMENT
Microsoft Dynamics CRM versions	• Microsoft Dynamics CRM 2015 or Microsoft Dynamics CRM 2016 Note: AD, IFD, and Claims authentication types are supported for above editions. • Microsoft Dynamics CRM Online 2015, Microsoft Dynamics CRM Online 2015 Update 1, or Microsoft Dynamics CRM Online 2016 Update 1 or Microsoft Dynamics 365 Note: OAuth and Microsoft 365 authentication types are supported for these editions. For more information on authentication types, see Connection strings in XRM tooling to connect to Dynamics 365.

SPECIFICATION	REQUIREMENT
Business Central Integration Solution (.zip)	For Dynamics CRM 2015, Dynamics CRM Online 2015, and Dynamics CRM Online 2015 Update 1: • Use the DynamicsNAVIntegrationSolution.zip file that is found on the Dynamics NAV 2016 installation media (DVD) to install the solution. • Item Availability isn't supported on Dynamics CRM 2015, versions Update 1 and Online. For more information, see Preparing Dynamics 365 for Sales for Integration.

SALES/DYNAMICS NAV/BUSINESS CENTRAL	2015/UPDATE 1/ONLINE	2016/UPDATE 1/ONLINE	SALES ENTERPRISE (V8.X)	SALES ENTERPRISE AND SALES PROFESSIONAL (V9.X)
Dynamics NAV 2016	Supported ***	Supported ***	Supported ***	Supported ***
Dynamics NAV 2017	Supported **	Supported *	Supported *	Supported *
Dynamics NAV 2018	Supported **	Supported *	Supported *	Supported *
Business Central (online)	Not supported **	Not supported **	Supported *	Supported *
Business Central (on-premises)	Supported **	Supported *	Supported *	Supported *

Legend:

- "*" item availability capability is supported.
- "**" integration solution can be installed from the Dynamics NAV 2016 DVD, but viewing item availability isn't supported.
- "***" viewing item availability isn't supported

> **NOTE**
> AD, IFD and Claims authentication types are supported for the 2015 and 2016 on-premises versions of Dynamics 365 Sales. OAuth and Microsoft 365 authentication are supported for the 2015, 2015 Update 1, and 2016 Update 1 online versions of Dynamics 365 Sales. For more information on authentication types, see Use connection strings in XRM tooling to connect to Dynamics 365 for Customer Engagement apps (on-premises).

See Also

Welcome to the Developer and IT-Pro Help for Business Central
Product and Architecture Overview
Deployment

Software Lifecycle Policy and Dynamics 365 Business Central On-Premises Updates

2/17/2021 • 2 minutes to read • Edit Online

This article outlines the lifecycle and support policies for Dynamics 365 Business Central on-premises updates.

Fixed Lifecycle Policy

Dynamics 365 Business Central (on-premises) October'18 and April '19 Update software is covered by the Fixed Lifecycle Policy.

Licensed customers must stay current with updates to the Dynamics 365 Business Central on-premises software in accordance with the following servicing and system requirements. This policy requires the customer to maintain an Enhancement Plan and deploy updates as noted later in this article.

RELEASE	VERSION	BUILD NUMBER	AVAILABILITY	MAINSTREAM SUPPORT ENDS
Dynamics 365 Business Central (on-premises)	October'18 Update (version 13.x)	24630	November 1, 2018	**
Dynamics 365 Business Central (on-premises)	April '19 Update (version 14.x)	29537	April 1, 2019	October 10, 2023

** In order to obtain mainstream support after April 14, 2020, customers must update to the April 2019 release or a later update for the April 2019 version

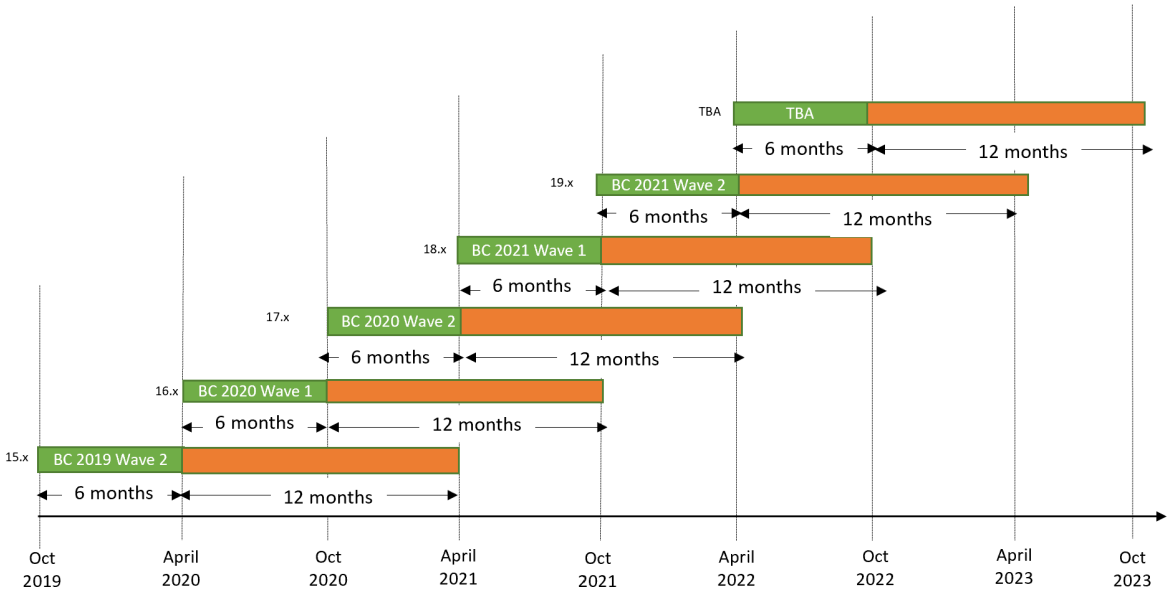
Modern Lifecycle Policy

Dynamics 365 Business Central (on-premises) 2019 release wave 2 and newer updates are covered by the Modern Lifecycle Policy.

The customer is in full control of its on-premises deployments and must follow this policy. The customer is in control of installing updates in its on-premises environments. Microsoft will support the Dynamics 365 Business Central (on-premises) software as indicated on the Microsoft Lifecycle Policy for Business Central on-premises page, but only if the customer keeps the deployed software current according to this policy. By keeping current, you're getting all the latest feature and bug fixes.

The following image illustrates the update schedule for Dynamics 365 Business Central.

Modern Lifecycle Policy



Modern Lifecycle Policy Legend

← 6 months →	6 months incidates features/bugs/regulatory fixes released for this cycle
← 12 months →	12 months incidates bugs/regulatory fixes released for this cycle

See Also

Microsoft Lifecycle Policy for Business Central on-premises
Configuring Technical Support
Welcome to the Developer and IT-Pro Help for Dynamics 365 Business Central
Upgrading to Dynamics 365 Business Central
Deployment of Dynamics 365 Business Central

FAQ About the Windows Client and Business Central

2/17/2021 • 5 minutes to read • Edit Online

The first releases of Business Central on premises included an installed client derived from Dynamics NAV. Starting with 2019 release wave 2, this legacy component, referred to as "the Windows client", will no longer be available for Business Central. Find answers for some of the most common questions here.

I have heard "modern clients only". What is this about?

Businesses and users want to be reassured that only the newest, most advanced, and up-to-date tools are being used to access their data. With Business Central 2019 release wave 2, released October 2019, users switch to the **modern experience** in the browser ("the web client"), the Android or iOS mobile apps, or the Windows 10 desktop app, which are available through the respective stores.

Connecting the Windows client to Business Central is not supported in Business Central 2019 release wave 2 and onwards.

Why is Microsoft discontinuing the Windows client?

Our customers must feel comfortable that the tools they use are fit for new hardware, operating systems, and changing environments. We have accelerated our investment in speed and productivity features for the modern clients, thereby achieving a major milestone in its transformation into a world-class desktop experience for both new and expert users.

While the Windows client was inherently bound to the Windows operating system, the modern clients allow us to reach more customers and more users within an organization, no matter their platform or device of choice. The latest technologies allow us to innovate at a rapid pace and respond to accelerating compliance requirements, the changing technology landscape, and requests from the community. In addition, the lower installation footprint on client devices makes it easier for IT departments and hosters to maintain and support their user base.

When is the Windows client discontinued in Business Central?

From **October 2019**, with Business Central 2019 release wave 2.

It was first announced in 2018 at various conferences and then with a detailed timeline earlier in 2019. For more information, see Business Central April 2019 Update and the road ahead.

Will the Windows client still be supported in older releases of Business Central and Dynamics NAV?

Yes. You can safely continue to use the Windows client on premises and receive support as long as you follow the lifecycle policy for your on-premises installations of Business Central. For more information, see Lifecycle FAQ - Dynamics.

The Windows client **remains supported** for the Business Central April 2019 release and all earlier releases of Business Central on premises and Dynamics NAV, in accordance with the support lifecycle process.

Does this impact me if I use Business Central online?

No. This change only impacts on-premises installations because the Windows client was only available on premises.

Does this impact me if I use Business Central on premises?

Yes. When you **choose to upgrade** to Business Central 2019 release wave 2 or later, you must switch to access Business Central using one of the modern clients. The most popular choice on desktop computers is the web-browser client where your browser is pointing to an on-premises web server using a URL, such as this example (not active): `https://myserver.mydomain.com/BC170`

What if I really want to have an installable component or at least an icon on my desktop?

You can always add a browser shortcut on your desktop or pin the web page with Business Central to your Windows task bar. Alternatively, the Business Central Windows 10 desktop app, which is available from Microsoft Store, is a great way to access Business Central both online and on-premises. To get the app, go to Microsoft Dynamics 365 Business Central in the store.

How does this impact mobile?

There is **no impact** on mobile apps for Business Central as they are already part of the modern-client family. For more information about the mobile apps, seeGetting Business Central on Your Mobile Device.

Can I still work with Business Central data in Excel?

Yes. There are multiple ways to work with Business Central and Excel, including the following:

- The Open in Excel feature that downloads any list as an Excel file for your processing or reporting
- The Edit in Excel feature that allows you to edit almost any list-based data in Excel and publish it back to Business Central

For more information, see Viewing and Editing in Excel From Business Central. For instructions on how to configure it for on-premises, see Setting up the Excel Add-In for Editing Business Central Data.

Note that the legacy, COM-based Excel plugin that used to be included on the installation media is no longer supported.

Can I still use the same Outlook add-in?

Yes. This change does not impact the Outlook add-in. In fact, the modern Outlook add-in is based on the same familiar web experience. For more information, see Using Business Central as your Business Inbox in Outlook.

What happened to C/SIDE, the legacy development environment?

In line with the retirement of the Windows client, Business Central 2019 release wave 2 marks a milestone as the first release without the legacy development environment (also known as C/SIDE). The modern developer experience, which is based on Visual Studio Code and the new AL language, supports developing large apps, such as the base application from Microsoft.

Therefore, C/SIDE is discontinued for Business Central going forward. Partners enjoy tremendous productivity and performance gains after moving to the newest tools. For more information, see Development in AL.

Which features are available in the modern clients and where can I find the roadmap?

Business Central is a highly adaptable modern business management solution. It is rich in features and options and is continuously being enhanced. The roadmap is best represented by the release plans, which are updated every six months. For more information, see Overview of Dynamics 365 Business Central 2019 release wave 2.

See Also

FAQ for Developing in AL
Features not implemented in on-premises deployments of Dynamics 365 Business Central
Business Central Component and System Topology, Additional Components
Software Lifecycle Policy and Dynamics 365 Business Central On-Premises Updates
Dynamics 365 Business Central Compliance
FAQ for Dynamics 365 Update Policies
Dynamics 365 Resources
Welcome to Dynamics 365 Business Central

Dynamics 365 Business Central On-Premises October'18 Updates

2/17/2021 • 4 minutes to read • Edit Online

This article lists cumulative updates released for the October'18 release of Microsoft Dynamics 365 Business Central on-premises. A cumulative update is a cumulative set of files that includes hotfixes and regulatory features for Business Central. If you have customers using Business Central October'18 on-premises, we recommend applying the latest cumulative update. For customers using Business Central online, check the Business Central Admin center to see whether the tenants have been updated.

Each cumulative update is intended mainly for solutions that are experiencing the problems described in the Support articles linked to below. However, you're advised to always keep your solution updated with the latest cumulative update. Support professionals in Customer Support Services are ready to help you if:

- You're in doubt about whether a cumulative update addresses your specific problem
- You want to confirm whether any special compatibility, installation, or download issues are associated with a particular cumulative update.

For more information, see https://support.microsoft.com/contactus/.

The latest cumulative update listed includes new hotfixes and regulatory, plus hotfixes and regulatory features released in previous cumulative updates.

We recommend that you install the latest cumulative update.

Available updates for Business Central October 2018

The following table lists the cumulative updates that have been released for the October'18 release of Business Central (version 13). The cumulative updates include hotfixes that apply to all countries and hotfixes that apply to specific local versions. Check the relevant Support article for a description.

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4549676	Cumulative Update 18	April 2020	Application Build 41909 Platform Build 41879	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4538886	Cumulative Update 17	March 2020	Application Build 41201 Platform Build 41152	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4539525	Cumulative Update 16	February 2020	Application Build 40469 Platform Build 40427	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4536556	Cumulative Update 15	January 2020	Application Build 39340 Platform Build 39276	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4536556	Cumulative Update 14	December 2019	Application Build 38692 Platform Build 38637	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4536556	Cumulative Update 13	November 2019	Application Build 37622 Platform Build 37612	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4522950	Cumulative Update 12	October 2019	Application Build 36481 Platform Build 36446	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4518534	Cumulative Update 11	September 2019	Application Build 35888 Platform Build 35820	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4514619	Cumulative Update 10	August 2019	Application Build 34590 Platform Build 34560	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4511343	Cumulative Update 09	July 2019	Application Build 33838 Platform Build 33825	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4506821	Cumulative Update 08	June 2019	Application Build 32990 Platform Build 32943	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4501063	Cumulative Update 07	May 2019	Application Build 31809 Platform Build 31719	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4492596	Cumulative Update 06	April 2019	Application Build 29777 Platform Build 29718	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4492596	Cumulative Update 05	March 2019	Application Build 29483 Platform Build 29358	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4487772	Cumulative Update 04	February 2019	Application Build 28874 Platform Build 28871	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4483882	Cumulative Update 03	January 2019	Application Build 27233 Platform Build 27183	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4479230	Cumulative Update 02	December 2018	26556	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4470115	Cumulative Update 01	November 2018	Application Build 25940 Platform Build 25924	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

See Also

Software lifecycle policy and on-premises releases
Installing a Business Central Cumulative Update
Dynamics 365 Business Central On-Premises April'19 Updates
Resources for Help and Support
System Requirements
Upgrading to Business Central
Countries and Translations Supported
Welcome to Dynamics 365 Business Central
Microsoft Dynamics 365 Business Central on the Dynamics 365 blog
Dynamics NAV developer and ITpro content

Dynamics 365 Business Central On-Premises Spring 2019 Updates

2/17/2021 • 4 minutes to read • Edit Online

This article lists updates released for the Spring '19 release of Business Central on-premises. A cumulative update is a cumulative set of files that includes hotfixes and regulatory features for Business Central. We recommend applying the latest cumulative update. For customers using Business Central online, check the Business Central Admin center to see whether the tenants have been updated.

Each cumulative update is intended mainly for solutions that are experiencing the problems described in the linked Support. However, you're advised to always keep your solution updated with the latest cumulative update. Support professionals in Customer Support Services are ready to help you if:

- You're in doubt about whether a cumulative update addresses your specific problem
- You want to confirm whether any special compatibility, installation, or download issues are associated with a particular cumulative update.

For more information, see https://support.microsoft.com/contactus/.

The latest cumulative update listed includes new hotfixes and regulatory, plus hotfixes and regulatory features released in previous cumulative updates.

We recommend that you install the latest cumulative update.

Available updates for Business Central April 2019

The following table lists the cumulative updates released for the Spring 2019 release of Business Central (version 14). The cumulative updates include hotfixes that apply to all countries and hotfixes that apply to specific local versions. Check the relevant Support article for a description.

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
5000693	Cumulative Update 21	February 2021	Application 14.22.46358 Platform Build 14.0.46351	AT, administrative unit, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4595149	Cumulative Update 20	January 2021	Application 14.21.46103 Platform Build 14.0.46080	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583550	Cumulative Update 19	December 2020	Application 14.20.45741 Platform Build 14.0.45739	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583511	Cumulative Update 18	November 2020	Application 14.19.45386 Platform Build 14.0.45365	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4583496	Cumulative Update 17	October 2020	Application 14.18.44963 Platform Build 14.0.44962	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4576662	Cumulative Update 16	September 2020	Application 14.17.44663 Platform Build 14.0.44656	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563410	Cumulative Update 15	August 2020	Application 14.16.44342 Platform Build 14.0.44327	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563405	Cumulative Update 14	July 2020	Application 14.15.43800 Platform Build 14.0.43793	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4564070	Cumulative Update 13	June 2020	Application 14.14.43294 Platform Build 14.14.43286	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4549684	Cumulative Update 12	May 2020	Application 42648 Platform Build 42627	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4549677	Cumulative Update 11	April 2020	Application 41935 Platform Build 41862	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4538887	Cumulative Update 10	March 2020	Application 41204 Platform Build 41143	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4539529	Cumulative Update 09	February 2020	Application 40471 Platform Build 40464	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4536555	Cumulative Update 08	January 2020	Application 39327 Platform Build 39277	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4533396	Cumulative Update 07	December 2019	Application 38658 Platform Build 38650	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4528705	Cumulative Update 06	November 2019	Application 37609 Platform Build 37587	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4522949	Cumulative Update 05	October 2019	Application 36463 Platform Build 36457	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4518535	Cumulative Update 04	September 2019	Application Build 35970 Platform Build 35916	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4515445	Cumulative Update 03	August 2019	Application Build 35602 Platform Build 35570	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4514872	Cumulative Update 02	July 2019	Application Build 34444 Platform Build 34251	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4501146	Cumulative Update 01	May 2019	Application Build 32615 Platform Build 32600	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

See Also

Software lifecycle policy and on-premises releases
Installing a Business Central Cumulative Update
Dynamics 365 Business Central On-Premises October'18 Updates
Resources for Help and Support
System Requirements April '19
Upgrading to Business Central
Countries and Translations Supported
Welcome to Dynamics 365 Business Central
Microsoft Dynamics 365 Business Central on the Dynamics 365 blog
Dynamics NAV developer and ITpro content

Dynamics 365 Business Central On-Premises 2019 Release Wave 2 Updates

2/17/2021 • 3 minutes to read • Edit Online

This article lists updates released for the Microsoft Dynamics 365 Business Central 2019 release wave 2 for on-premises. An update is a set of files that includes all hotfixes and regulatory features that have been released for Business Central. If you have customers using Business Central 2019 release wave 2 on-premises, we recommend you apply this update so that your customers are using the latest version of Business Central. If you have customers using Business Central online, check the Business Central Admin center to see if the tenants have been updated.

Each update is intended mainly for solutions that are experiencing the problems described in the Support articles linked to below. However, you're advised to always keep your solution updated with the latest update. If you're in doubt about whether this update addresses your specific problem, or you want to confirm whether any special compatibility, installation, or download issues are associated with a particular update, then contact Customer Support Services. They are ready to help you. For more information, see https://support.microsoft.com/contactus/.

The latest update listed in this article includes hotfixes and regulatory features released for Business Central. The latest update also includes hotfixes and regulatory features released in previous updates.

You should always install the latest update.

Available updates for Business Central 2019 Release Wave 2

The following table lists the updates that have been released for 2019 release wave 2 of Business Central (version 15). The updates include hotfixes that apply to all countries and hotfixes that apply to specific local versions. Check the relevant Support article for a description.

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
5000694	Update 15.15	February 2021	Application Build 15.15.46359 Platform Build 15.0.46345	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4595150	Update 15.14	January 2021	Application Build 15.14.46107 Platform Build 15.0.46099	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583552	Update 15.13	December 2020	Application Build 15.13.45740 Platform Build 15.0.45731	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583512	Update 15.12	November 2020	Application Build 15.12.45390 Platform Build 15.0.45381	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4583500	Update 15.11	October 2020	Application Build 15.11.44966 Platform Build 15.0.44957	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4576663	Update 15.10	September 2020	Application Build 15.10.44664 Platform Build 15.10.44655	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563414	Update 15.9	August 2020	Application Build 15.9.44343 Platform Build 15.0.44325	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563406	Update 15.8	July 2020	Application Build 15.8.43801 Platform Build 15.0.43794	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4564071	Update 15.7	June 2020	Application Build 15.7.43293 Platform Build 15.0.43283	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4549685	Update 15.6	May 2020	Application Build 15.6.42646 Platform Build 15.0.42629	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4549678	Update 15.5	April 2020	Application Build 15.5.41926 Platform Build 15.0.41893	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4538888	Update 15.4	March 2020	Application Build 15.4.41213 Platform Build 15.0.41193	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4539530	Update 15.3	February 2020	Application Build 15.3.40822 Platform Build 15.0.40791	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4533389	Update 15.2	December 2019	Application Build 15.2.39040 Platform Build 15.0.38951	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4528699	Update 15.1	November 2019	Application Build 15.1.38071 Platform Build 15.0.37898	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

See Also

Software lifecycle policy and on-premises releases
Installing a Business Central Update
Dynamics 365 Business Central On-Premises Spring 2019 Updates
Dynamics 365 Business Central On-Premises October'18 Updates
Resources for Help and Support
System Requirements April '19
Upgrading to Business Central
Countries and Translations Supported
Welcome to Dynamics 365 Business Central
Microsoft Dynamics 365 Business Central on the Dynamics 365 blog
Dynamics NAV developer and IT Pro content

Dynamics 365 Business Central On-Premises 2020 Release Wave 1 Updates

2/17/2021 • 3 minutes to read • Edit Online

This article lists updates released for the Microsoft Dynamics 365 Business Central 2020 release wave 1 for on-premises. An update is a set of files that includes all hotfixes and regulatory features that have been released for Business Central. If you have customers using Business Central 2020 release wave 1 on-premises, we recommend you apply this update so that your customers are using the latest version of Business Central. If you have customers using Business Central online, check the Business Central Admin center to see if the tenants have been updated.

Each update is intended mainly for solutions that are experiencing the problems described in the Support articles linked to below. However, you're advised to always keep your solution updated with the latest update. If you're in doubt about whether this update addresses your specific problem, or you want to confirm whether any special compatibility, installation, or download issues are associated with a particular update, then contact Customer Support Services. They are ready to help you. For more information, see https://support.microsoft.com/contactus/.

The latest update listed in this article includes hotfixes and regulatory features released for Business Central. The latest update also includes hotfixes and regulatory features released in previous updates.

You should always install the latest update.

Available updates for Business Central 2020 Release Wave 1

The following table lists the updates that have been released for 2020 release wave 1 of Business Central (version 16). The updates include hotfixes that apply to all countries and hotfixes that apply to specific local versions. Check the relevant Support article for a description.

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
5000695	Update 16.10	February 2021	Application Build 16.10.21502 Platform Build 16.0.21469	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4595151	Update 16.9	January 2021	Application Build 16.9.20537 Platform Build 16.0.20513	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583553	Update 16.8	December 2020	Application Build 16.8.19389 Platform Build 16.0.19389	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583513	Update 16.7	November 2020	Application Build 16.7.18411 Platform Build 16.0.18359	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
4583501	Update 16.6	October 2020	Application Build 16.6.17046 Platform Build 16.0.17024	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563415	Update 16.5	September 2020	Application Build 16.5.15953 Platform Build 16.5.15941	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563415	Update 16.4	August 2020	Application Build 16.4.15445 Platform Build 16.0.15420	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4563407	Update 16.3	July 2020	Application Build 16.3.14238 Platform Build 16.0.14195	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4549686	Update 16.2	June 2020	Application Build 16.2.13779 Platform Build 16.0.13772	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4549686	Update 16.1	May 2020	Application Build 16.1.12805 Platform Build 16.0.12758	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

See Also

Software lifecycle policy and on-premises releases
Installing a Business Central Update
Dynamics 365 Business Central On-Premises Spring 2019 Updates
Dynamics 365 Business Central On-Premises October'18 Updates
Resources for Help and Support
System Requirements April '19
Upgrading to Business Central
Countries and Translations Supported
Welcome to Dynamics 365 Business Central
Microsoft Dynamics 365 Business Central on the Dynamics 365 blog
Dynamics NAV developer and IT Pro content

Dynamics 365 Business Central On-Premises 2020 Release Wave 2 Updates

2/17/2021 • 2 minutes to read • Edit Online

This article lists updates released for the Microsoft Dynamics 365 Business Central 2020 release wave 2 for on-premises. An update is a set of files that includes all hotfixes and regulatory features that have been released for Business Central. If you have customers using Business Central 2020 release wave 2 on-premises, we recommend you apply this update so that your customers are using the latest version of Business Central. If you have customers using Business Central online, check the Business Central Admin center to see if the tenants have been updated.

Each update is intended mainly for solutions that are experiencing the problems described in the Support articles linked to below. However, you're advised to always keep your solution updated with the latest update. If you're in doubt about whether this update addresses your specific problem, or you want to confirm whether any special compatibility, installation, or download issues are associated with a particular update, then contact Customer Support Services. They are ready to help you. For more information, see https://support.microsoft.com/contactus/.

The latest update listed in this article includes hotfixes and regulatory features released for Business Central. The latest update also includes hotfixes and regulatory features released in previous updates.

You should always install the latest update.

Available updates for Business Central 2020 Release Wave 2

The following table lists the updates that have been released for 2020 release wave 2 of Business Central (version 17). The updates include hotfixes that apply to all countries and hotfixes that apply to specific local versions. Check the relevant Support article for a description.

KNOWLEDGE BASE ID	TITLE	RELEASE DATE	BUILD NO.	LOCAL VERSIONS INCLUDED
5000696	Update 17.4	February 2021	Application Build 17.4.21531 Platform Build 17.0.21516	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4595152	Update 17.3	January 2021	Application Build 17.3.20605 Platform Build 17.0.20517	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583554	Update 17.2	December 2020	Application Build 17.2.19367.19396 Platform Build 17.0.19353.19391	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK
4583515	Update 17.1	November 2020	Application Build 17.1.18256.18792 Platform Build 17.0.18204.18738	AT, AU, BE, CH, CZ, DE, DK, ES, FI, FR, IS, IT, NA, NL, NO, NZ, RU, SE, UK

See Also

Software lifecycle policy and on-premises releases
Installing a Business Central Update
Dynamics 365 Business Central On-Premises Spring 2019 Updates
Dynamics 365 Business Central On-Premises October'18 Updates
Resources for Help and Support
System Requirements April '19
Upgrading to Business Central
Countries and Translations Supported
Welcome to Dynamics 365 Business Central
Microsoft Dynamics 365 Business Central on the Dynamics 365 blog
Dynamics NAV developer and IT Pro content

Running a Container-Based Development Environment

2/17/2021 • 3 minutes to read • Edit Online

Dynamics 365 Business Central is available as artifacts for running on Docker on a Windows system with Docker installed.

> **TIP**
> Use the `Get-BCArtifactUrl` from the BCContainerHelper PowerShell module to get the artifact URL for the version of Dynamics 365 Business Central that you want.

Install and configure Docker

Install Docker and configure it for Windows Containers.

1. Please choose the version of Docker that is appropriate for the host operating system.
   - For Windows 10, use Docker Community Edition. For more information, see Install instructions.
   - For Windows Server, use Docker Enterprise Edition. For more information, see Install instructions.
2. For Windows 10, switch Docker to use Windows containers. By default Docker Community Edition uses Linux containers.

   To switch to Windows containers, in the Taskbar, right-click the Docker icon  , and then select **Switch to Windows Containers**. For more information, see Switch between Windows and Linux containers.

> **NOTE**
> You can run Business Central on Docker using Docker commands, or you can use the BCContainerHelper PowerShell module. The BCContainerHelper module removes a lot of the complexity of running Docker.

Using Docker commands

Open a Command Prompt as Administrator. In the command prompt, identify your version of Windows (example 10.0.19041.329). Run this command to pull the latest version of the generic image used to to run Business Central on Docker:

```
docker pull mcr.microsoft.com/businesscentral:10.0.19041.329
```

Use this command to run a sandbox container with the US localization of version 16.3.14085.14363 on Docker:

```
docker run -e accept_eula=Y -m 4G -e
artifacturl=https://bcartifacts.azureedge.net/sandbox/16.3.14085.14363/us
mcr.microsoft.com/businesscentral:10.0.19041.329
```

> **IMPORTANT**
> You must specify the correct Windows Version in the generic image name. If your version of Windows doesn't have a corresponding generic Docker image, you might need to use Hyper-V isolation.

After starting the `docker run` command above, you will see log entries similar to the following:

```
Initializing...
Starting Container
Hostname is b8c6941bb168
...
Container IP Address: 172.21.4.208
Container Hostname  : b8c6941bb168
Container Dns Name  : b8c6941bb168
Web Client          : https://b8c6941bb168/BC/
Admin Username      : admin
Admin Password      : Zupa5925
Dev. Server         : https://b8c6941bb168
Dev. ServerInstance : BC

Files:
http://b8c6941bb168:8080/ALLanguage.vsix
http://b8c6941bb168:8080/certificate.cer

Initialization took 66 seconds
Ready for connections!
```

At this point, you can open your browser and type in the Web client URL from the log. You will be prompted to log in with the Admin Username/Password that is shown.

> **NOTE**
> The container image uses a so called self-signed certificate for HTTPS communication. Because of that, your browser might warn you that the page you are requesting is unsafe. In those specific circumstances, and only for test and development environments, it is safe to ignore this warning. If you want to resolve this warning, you can install the certificate on your PC. For more information, see the link under **Files** in the log entries.

Using the BCContainerHelper PowerShell module

To support the use of containers, optional PowerShell scripts are available, which support setup of development environments. Use the `BCContainerHelper` to work with containers. On a Windows 10, Windows Server 2016 or Windows server 2019 machine, start Powershell as an Administrator and type:

```
install-module BCContainerHelper -force
```

To see which functions are available in the BCContainerHelper module use the following command:

```
Write-BCContainerHelperWelcomeText
```

To get quickly get started, run the following command from the BCContainerHelper module:

```
$artifactUrl = Get-BcArtifactUrl -type sandbox -country us -select Latest
New-BCContainer -accept_eula -containerName mysandbox -artifactUrl $artifactUrl
```

The `BCContainerHelper` will create a folder on the C:\ drive called *bcartifacts.cache* for caching artifacts. It will also create a folder under C:\ProgramData called BCContainerHelper and will place all working files underneath that folder. The C:\ProgramData\BCContainerHelper folder will be shared to the container for transfer of files etc. If you do not specify a username and a password, it will ask for your password and use the current Windows username. If you specify your windows password, the container setup will use Windows Authentication integrated with the host. The `BCContainerHelper` will also create shortcuts on the desktop for the Dynamics 365 Business Central Web client, a container prompt, and a container PowerShell prompt.

The `BCContainerHelper` module also allows you to add the `-includeCSide` switch (For Business Central versions 14 or earlier) in order to add the Dynamics 365 Business Central Windows client and C/SIDE to the desktop and export all objects to a folder underneath C:\ProgramData\BCContainerHelper\Extensions for the object handling functions from the module to work.

See Also

Getting Started with AL
Get started with the Container Sandbox Development Environment
Keyboard Shortcuts
AL Development Environment
FAQ for Developing in AL

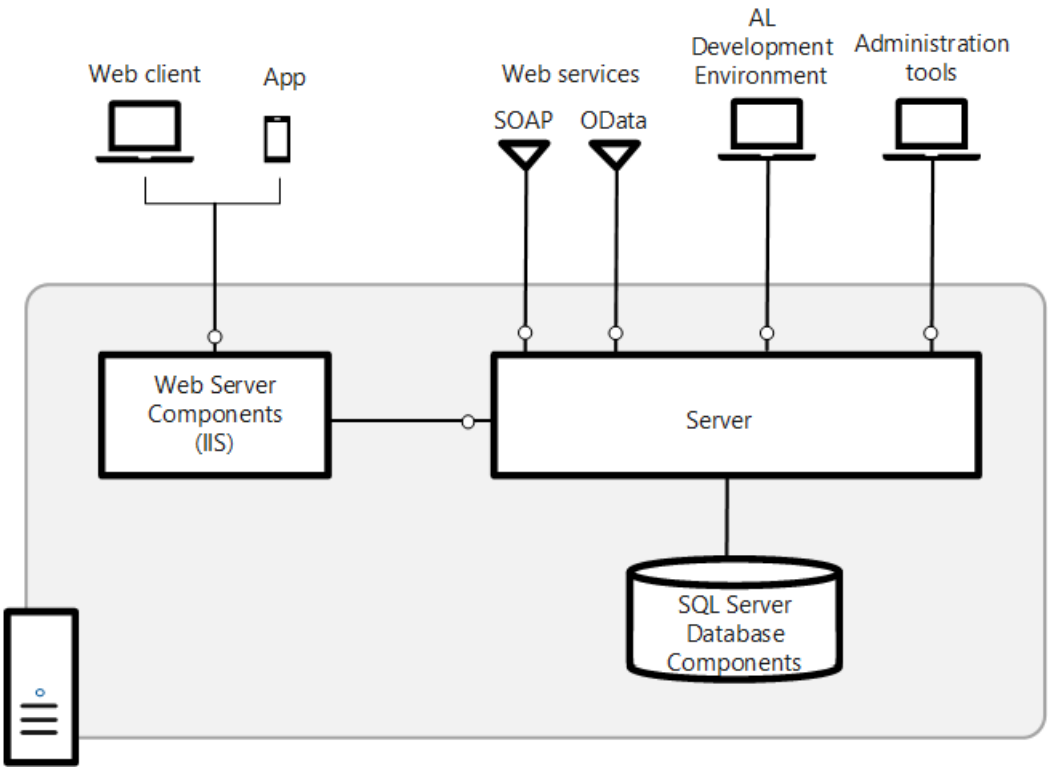
Business Central Component and System Topology

2/17/2021 • 3 minutes to read • Edit Online

The Business Central deployment includes three core components for serving the application to users. It also includes various tools and components for managing, developing, extending, and testing the application.

Multi-Tier System Topology

To understand the components is useful to first look at the base topology of a Business Central deployment, as illustrated in the following diagram:



Components

Main components

Every deployment must include the core components: Web server, Server, and SQL Database.

COMPONENT	DESCRIPTION	MORE INFORMATION
SQL Database	An SQL Server or Azure SQL Database database that contains application object definitions and business data. In a multitenant deployment, the application and business data can be separated into different databases: the application database and the tenant, which is the database that contains the business data. In this case, there can be one or more tenants for a single application database.	Creating Databases Deploy a Business Central Database to Azure SQL Database

COMPONENT	DESCRIPTION	MORE INFORMATION
Server	Business Central Server is a .NET-based Microsoft service application that uses Windows Communication Framework to handle communication between clients and databases. It controls authentication, event logging, scheduled tasks, reporting, and more.	Configuring Business Central Server
Web Server	An Internet Information Server (IIS) web site, provisioned with the Business Central Web Server components, that enables access from the Business Central Web client and mobile apps.	Business Central Web Server
Business Central App	A desktop, phone, and tablet app for Business Central.	Windows Store App Store Google Play
Web services	SOAP and OData Web Services for exposing application functionality to external systems and users. Developers can create and publish functionality as web services. They expose pages, codeunits, or queries, and even enhance a page web service by using an extension codeunit.	Web Services

Development and administration components

COMPONENT	DESCRIPTION	MORE INFORMATION
AL development environment	An AL language extension for Visual Studio Code for developing applications and extensions.	Getting Started with C/SIDE and AL for On-Premises.
Business Central Server Administration tool	A Microsoft Management Console (MMC) for creating and configuring Business Central Server instances.	Business Central Server Administration Tool
Business Central Administration Shell	Windows PowerShell modules for managing the deployment, including tasks such adding and configuring Business Central Server and Web server instances, databases, and users, and administering extension packages.	Windows PowerShell Cmdlets for Business Central

Additional components

COMPONENT	DESCRIPTION	MORE INFORMATION
Demo Database	A database that contains application objects and sample business data for demonstration purposes.	

COMPONENT	DESCRIPTION	MORE INFORMATION
Dynamics NAV Development Environment	The C/SIDE client that was available in Dynamics NAV for developing applications using C/AL. In Business Central, this component is only required for doing upgrades but you can still use it to develop applications.	**DISCONTINUED AFTER:** Business Central Spring 2019 Development in C/AL in the Dynamics NAV Developer and IT Pro Help.
Dynamics NAV Development Shell	Windows PowerShell modules for merging and modifying application object files and creating extension packages. Installed with the Dynamics NAV Development Environment.	**DISCONTINUED AFTER:** Business Central Spring 2019 Windows PowerShell Cmdlets for Business Central
Dynamics NAV Client connected to Business Central	Windows Desktop application for accessing Business Central.	**DISCONTINUED AFTER:** Business Central Spring 2019
Microsoft Outlook Integration	A Business Central Server component for integrating with Microsoft Outlook.	**DISCONTINUED AFTER:** Business Central Spring 2019
Microsoft Outlook Add-in	A component to synchronize data, such as to-dos, contacts, and tasks, between Business Central and Outlook. The Outlook Add-In uses Business Central web services.	Setting Up the Office Add-ins for Outlook Integration
Microsoft Excel Add-in	A component that enables users to export data from Business Central to Excel.	**DISCONTINUED AFTER:** Business Central Spring 2019
Excel Add-in	A component that enables users to export data from Business Central to Excel.	Setting up the Excel Add-In
Page Testability	A Business Central Server component for testing pages.	
Automated Data Capture System	A system that tracks the movement of items in a warehouse.	Use Automated Data Capture Systems (ADCS **DISCONTINUED AFTER:** Business Central 2019 Release Wave 1. The VT100 Plug-in is no longer included on the product installation media.
ClickOnce Installer Tools	Tools for implementing ClickOnce installation for the Dynamics NAV Client connected to Business Central.	**DISCONTINUED AFTER:** Business Central Spring 2019 Deploying Microsoft Dynamics NAV Windows client Using ClickOnce in the Dynamics NAV Developer and IT Pro Help.

COMPONENT	DESCRIPTION	MORE INFORMATION
NAS Service	A server component that executes business logic without a user interface or user interaction. NAS services in Business Central Server support applications such as Microsoft Outlook Integration and the Job Queue.	Instead of using NAS services, we recommend that you use the Task Scheduler (see Task Scheduler. If you decide to use NAS, and want to read more about its configuration, see Configuring NAS Services in the Dev and IT Pro Help for Microsoft Dynamics NAV 2018.

See Also

Deployment
Installing Business Central Using Setup
Multitenant Deployment Architecture

Planning Your Dynamics 365 Business Central Deployment

2/17/2021 • 2 minutes to read • Edit Online

This article outlines some of the things you should consider and decide on before you install Business Central.

Most of the topics discussed in this article can be changed at any time after the initial installation.

Network Topology

A Business Central deployment consists of various components that support the production, development, and testing. These components can be installed on various computers. The deployment process varies depending on the topology that you implement.

For more information, see Deployment Topologies.

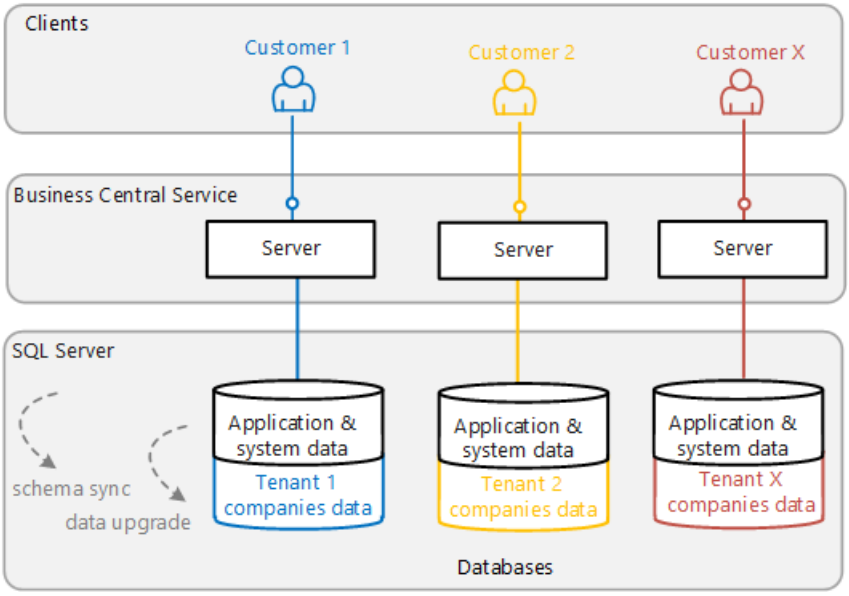
Single-tenancy and Multitenancy

A Business Central solution consists of an application (the code) and business data (the customer's data). This information is stored in a database and accessed through a Business Central Server instance. There are two different two deployment architectures to choose from: single-tenant and multitenant.

> **NOTE**
> For this discussion, a *customer* refers to a business or a group of legal entities whose data can be stored in one database, isolated from other customers. In Business Central, a customer can consist of one or more companies.

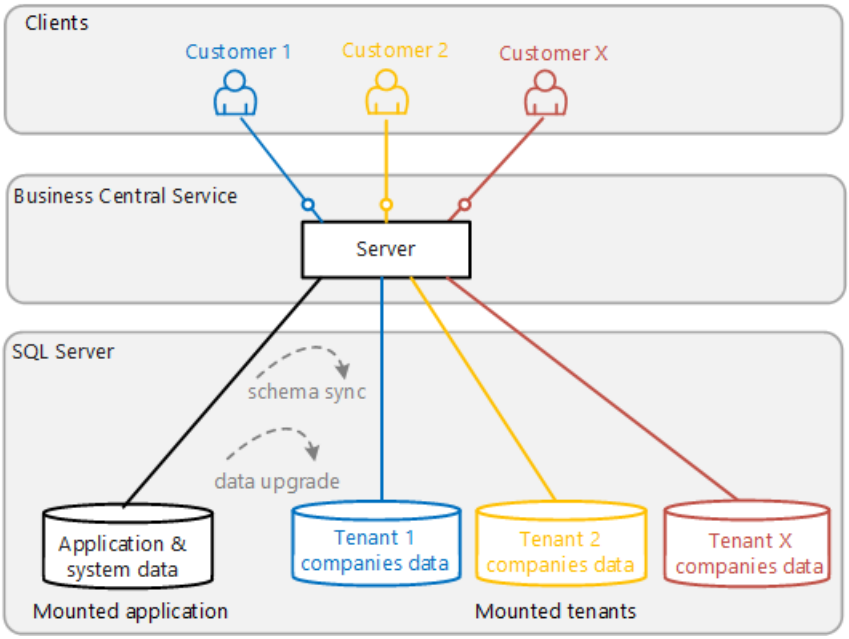
Single-tenant architecture

Business Central is installed as a single-tenant deployment by default. In a single-tenant deployment, the application and the business data are stored in the same database. Each customer solution has its own Business Central Server and database.



Multinenant architecture

In a multitenant deployment, the application and business data are stored in separate databases. There's a single Business Central Server and application database for multiple customers. But each individual customer has their own database for storing business data, which is referred to as a *tenant*.



Multitenancy centralizes the maintenance of the application, and at the same, time isolates each tenant. This architecture makes upgrading easier compared with a single-tenant deployment. For more information, see Multitenant Deployment Architecture.

User Authentication

Business Central supports several credential mechanisms for authorizing users trying to access data. By default, Windows authentication is used.

For more information, see Authentication and Credential Types.

Business Central Server Service Account

The central component of a Business Central deployment is the Business Central Server, which handles all communication between the client and the databases. The Business Central Server requires a log on account, referred to as the service account. By default, the Network Service Account is used. The Network Service Account is acceptable in a test environment, but we recommend that you use a domain account in your production environment.

For more information, see Provisioning the Business Central Server Service Account.

Enhancing Connection Security

Business Central offers features that help secure connections over a wide area network (WAN), such as connections from the Business Central Web Server, Dynamics NAV Client connected to Business Central, and web services to the Business Central Server. The implementation of these security features requires that you obtain a certificate from a certification authority or trusted provider.

For more information, see:

Using Security Certificates with Business Central On-Premises

Configuring SSL to Secure the Business Central Web Client Connection

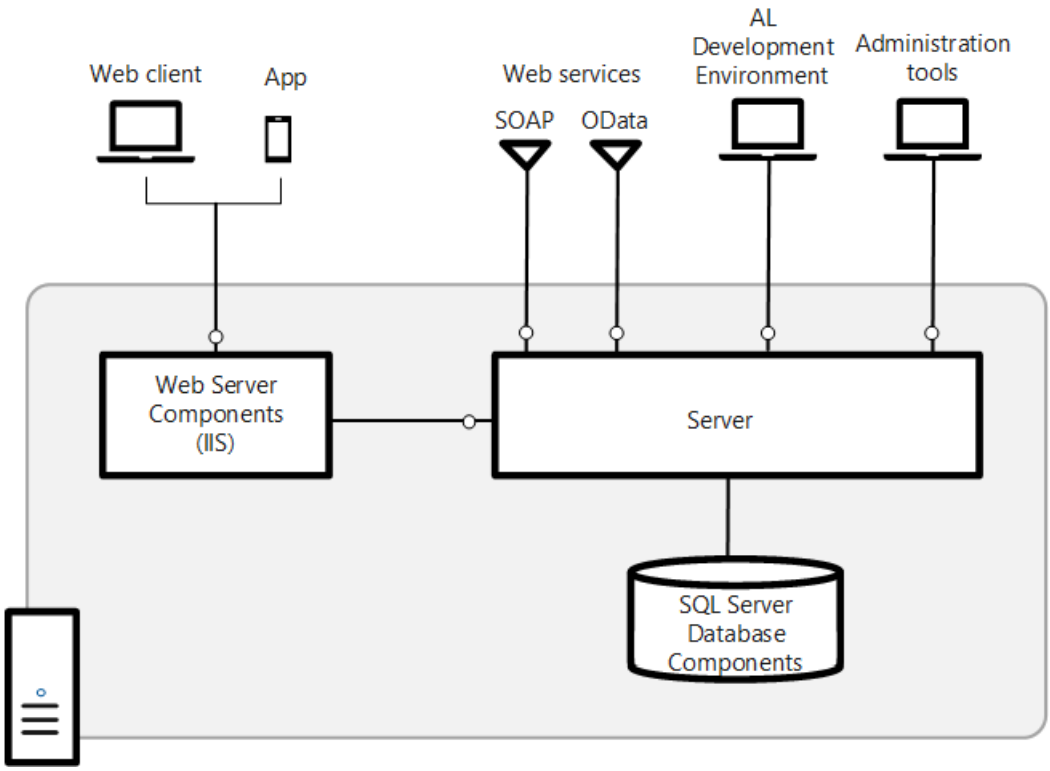
See Also

Upgrading to Business Central Business Central Components
System Requirements

Deployment Topologies

2/17/2021 • 2 minutes to read • Edit Online

A Business Central deployment consists of various components that support the production, development, and testing. These components can be installed on various computers, in different combinations. The deployment process varies depending on the topology that you implement. This article provides an overview of the supported topologies.



Deployment Scenarios

The following table lists some of the most typical deployment topologies for the Business Central, with links to detailed instructions. However, components can be installed on one, two, or three machines in any combination. So your scenario might vary.

TOPOLOGY	DESCRIPTION	MORE INFORMATION
Demonstration	Installs an end-to-end environment, complete with the base application and demonstration data for a single company, on a single computer. The installation enables access to Business Central from the Web client and App, and development. The deployment requires minimal hardware resources, preparation, and configuration.	Deploying a Demonstration Environment

TOPOLOGY	DESCRIPTION	MORE INFORMATION
Single-computer	Installs the Business Central Web Server components, Business Central Server, and the SQL Server database components on the same computer.	Deploying in a Single Computer Environment
Two-computer	Installs the Business Central Web Server components on one computer and the Business Central Server and the SQL Server database components on another computer.	Deploying in a Two Computer Environment
Three-computer	Installs the Business Central Web Server components, Business Central Server, and the SQL Server database components on separate computers.	Deploying in a Three Computer Environment

See Also

Install Business Central Using Setup
Business Central Web Server Overview

Deploying a Business Central Demonstration Environment

2/17/2021 • 2 minutes to read • Edit Online

This deployment scenario installs the major Business Central components on a single computer, complete with a base application and database with demonstration data. After the installation, you will have an end-to-end environment, where you can access Business Central data from the Web client. The installation requires minimal hardware resources, preparation, and configuration.

Installed Components and Configuration

Components

This scenario installs the following components:

- Business Central Web Server components
- Internet Information Services

  If IIS is already installed, then the setup will enable any required features that are not currently enabled.
- Business Central Server
- SQL Server Database Components, including CRONUS International Ltd. demonstration database and demo license.

  For information about what you can do with this license, see Demonstration License for Business Central On-Premises.
- Business Central Server Administration tool
- Al Development Environment

Configuration

This scenario uses the default setting of Business Central Setup, which includes the following:

- Business Central Web Server components
  - Port: 8080 (inbound rule automatically added to Windows Firewall)
  - Protocol: HTTP
- Windows authentication for authenticating users.
- Business Central Server configuration:
  - Service instance: BC160
  - Client service port: 7046
  - SOAP web services port: 7047
  - OData web services port: 7048
- Business Central database components configuration:
  - Service instance: BCDEMO

    - Database: Demo Database BC (15-0)
- NETWORK SERVICE account is used as the service account for Business Central Server and database.

Prepare for the Business Central Web client installation

1. Get access to the Business Central installation media. For example, this could be a DVD or network drive that contains the Business Central installation files.
2. Make sure that the computer meets the hardware and software requirements.

   For more information, see System Requirements.

Run Business Central Setup

1. From the Business Central installation media, run the setup.exe file to start the Business Central Setup.
2. Follow the setup until you get to the **Dynamics 365 Business Central** page, then choose **Advanced installation option** > **Install Demo**.

   The installation starts. This can take several minutes.

Open the Business Central Web client

- To open the Business Central Web client from the computer where you installed Business Central, on the **Start** menu, choose **All Programs**, and then choose **Business Central Web Client**.
- To open the Business Central Web client from other devices on the network, open an Internet browser, and type the following URL in the address box:

  ```
  https://ComputerName:8080/BC150
  ```

  Substitute **ComputerName** with the name of the computer where you installed Business Central. If you are working on the computer where you installed Business Central, then you can use **localhost**.

  For example:

  ```
  https://localhost:8080/BC150
  ```

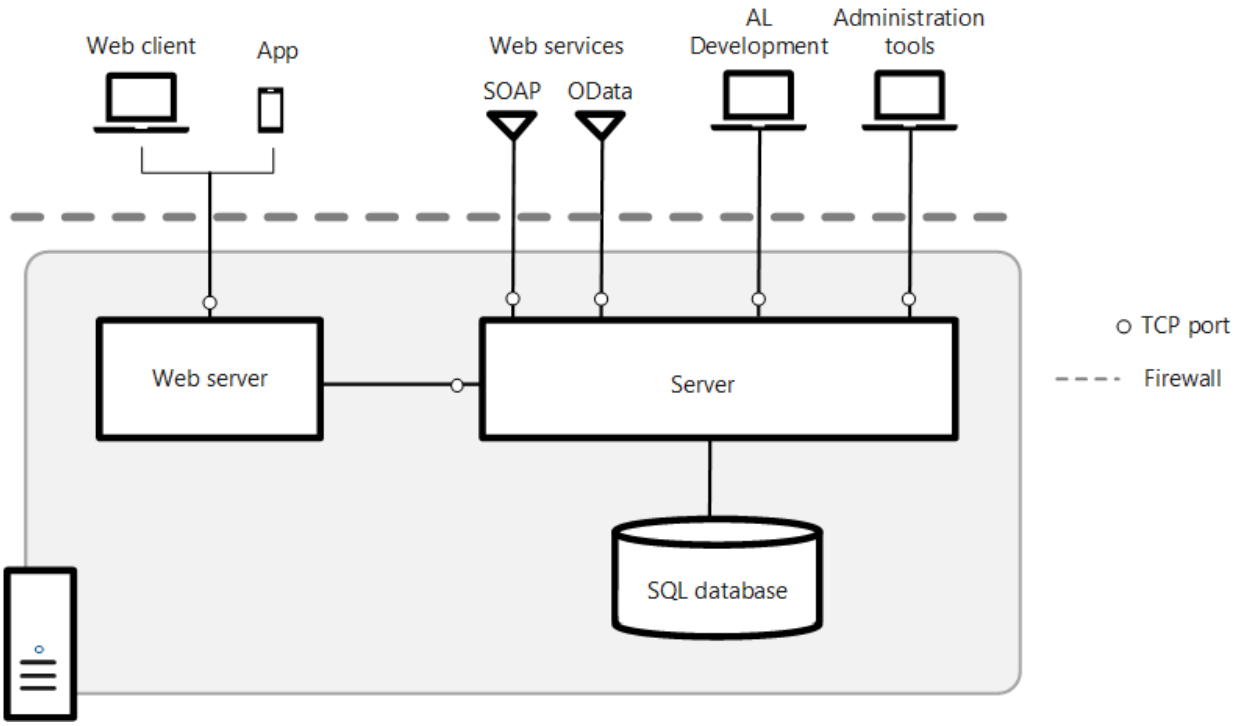
See Also

Business Central Web Server Overview

Deploying Business Central in a Single-Computer Topology

2/17/2021 • 3 minutes to read • Edit Online

In this scenario, you install the Business Central Web Server components, Business Central Server, and the SQL Server database components on the same computer.



Pre-Installation Tasks

The following table includes tasks to perform before you install.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Make sure that system requirements are met.	Verify that the computer has the required hardware and software installed.	System Requirements
Install Internet Information Services.	When you install the Business Central Web Server components, Business Central Setup creates a website for the Business Central Web client on IIS. If IIS is already installed, then make sure that the required features are enabled. **Note:** This step is optional because instead of installing and configuring IIS manually, you can use Business Central Setup to install IIS and enable the required features by setting the **Install IIS Prerequisites** option to **Yes**.	Configure Internet Information Services

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Determine the TCP ports for the Business Central Web client, client services, and SOAP/OData web services (optional) and allow communication on the port through Windows Firewall.	Business Central Setup creates a website on IIS. During Setup, you will have to choose the port to use for the site. The default port is port 8080. The default client services port is 7046. If you will enable SOAP and OData web services, you will also need to specify a port for each. The default ports are 7047 and 7048. If you choose to do so, Business Central Setup will automatically create an inbound rule in Windows Firewall that allows communication on the ports. Otherwise, you will have to do this manually.	Create an Inbound Port Rule in the Windows documentation.
Set up the service account for Business Central Server and the SQL Server database.	Optional. When you install Business Central Server, you can specify a user account that will be used to log on to the Business Central Server instance and Business Central database. The default service account is Network Service. If you want to use Network Service, then no action is required for this task.	Provisioning a Service Account
Obtain and install an SSL certificate.	Optional. If you want to configure SSL on the connection to Business Central Web client, then complete the following procedures: - Obtain an SSL certificate. - Import the certificate into the local computer store of the computer on which you will install the Business Central Web Server components. - Obtain the certificate's thumbprint. **Note:** You can also configure SSL after you have installed the Business Central Web client. For more information, see Post-installation Tasks.	Configure SSL to Secure the Web Client Connection

Installation Tasks

The following table includes tasks for installing the Business Central components.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Install Business Central Web Server components, Business Central Server, and SQL Server database components.	Run the Business Central Setup setup.exe file, choose the **Advanced installation options** > **Choose an installation option** > **Custom**, and then choose the **Server**, **SQL Server Database Components**, **Server**, and **Web Server Components** options.	Install Business Central Using Setup

Post-installation Tasks

The following table includes tasks that configure the Business Central Web Server components after installation. These tasks are optional depending on your organizational and network requirements.

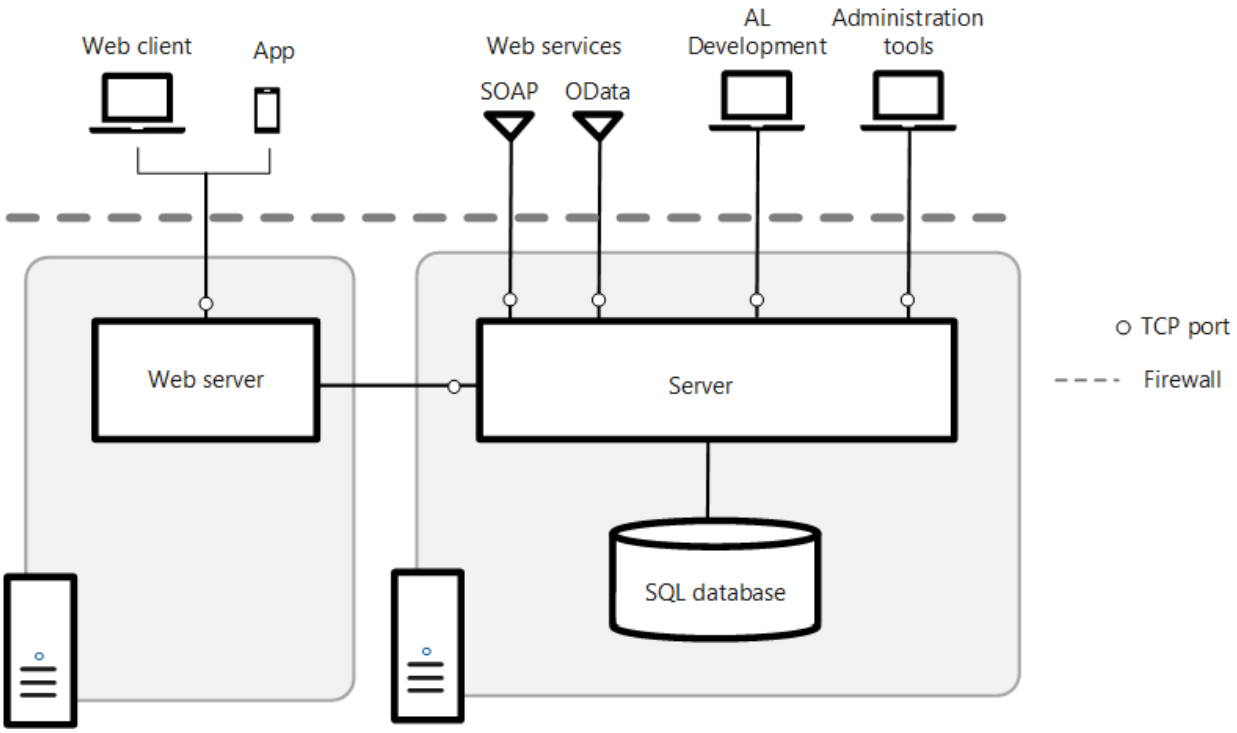
TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Change the user authentication method.	The Business Central supports the following authentication methods: Windows, UserName, NavUserPassword, and AccessControlService. By default, Windows authentication is used.	Authentication and User Credential Type
Secure the connection to the Business Central Web client with SSL.	You can help secure Business Central data that is transmitted over the Internet by enabling Secure Sockets Layer (SSL) on the connection to the Business Central Web client.	Configure SSL to Secure the Web Client Connection
Change the configuration of the Business Central Web Server.	There are several parameters in the navsettings.json configuration file for the Business Central Web Server that you can modify to change the behavior of the Business Central Web client. Some of the more common parameters include the Business Central Server instance, company, language, time zone, regional settings, session time out, and online Help URL.	Configuring Business Central Web Server
Set up multiple Business Central Web client applications.	You can set up multiple web server instances for the Business Central Web client on the existing website. The web server instances will use the same address (URL) except with an alias that specifies the specific application.	Creating and Managing Business Central Web Server Instances Using PowerShell
Configure web browsers on devices.	The Business Central Web client supports several different web browsers. To access the Business Central Web client, the web browser must be enabled on a device with cookies and JavaScript.	Web Client Requirements

See Also

Deploying Business Central in a Two-Computer Topology

2/17/2021 • 4 minutes to read • Edit Online

In this scenario, you install the Business Central Web Server components on a computer separate than Business Central Server and the SQL Server database components.



Pre-Installation Tasks

The following table includes tasks to perform before you install the Business Central Web Server components.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Make sure that system requirements are met.	Verify that the computer has the required hardware and software installed.	System Requirements
Install Internet Information Services.	When you install the Business Central Web Server components, Business Central Setup creates a website for the Business Central Web client on IIS. If IIS is already installed, then make sure that the required features are enabled. **Note:** This step is optional because instead of installing and configuring IIS manually, you can use Business Central Setup to install IIS and enable the required features by setting the **Install IIS Prerequisites** option to **Yes**.	Configure Internet Information Services

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Determine the TCP ports for the Business Central Web client, client services, and SOAP/OData web services (optional) and allow communication on the port through Windows Firewall.	Business Central Setup creates a website on IIS. During Setup, you will have to choose the port to use for the site. The default port is port 8080. The default client services port is 7046. If you will enable SOAP and OData web services, you will also need to specify a port for each. The default ports are 7047 and 7048. If you choose to do so, Business Central Setup will automatically create an inbound rule in Windows Firewall that allows communication on the ports. Otherwise, you will have to do this manually.	Create an Inbound Port Rule in the Windows documentation.
Set up the service account for Business Central Server and the SQL Server database.	Optional. When you install Business Central Server, you can specify a user account that will be used to log on to the Business Central Server instance and Business Central database. The default service account is Network Service. If you want to use Network Service, then no action is required for this task.	Provisioning a Service Account
Obtain and install an SSL certificate.	Optional. If you want to configure SSL on the connection to Business Central Web client, then complete the following procedures: - Obtain an SSL certificate. - Import the certificate into the local computer store of the computer on which you will install the Business Central Web Server components. - Obtain the certificate's thumbprint. **Note:** You can also configure SSL after you have installed the Business Central Web client. For more information, see Post-installation Tasks.	Configure SSL to Secure the Web Client Connection

Installation Tasks

The following table includes tasks for installing the Business Central Web Server components.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
On one computer, install Business Central Server and SQL Server Database Components on one computer	Run the Business Central Setup setup.exe file, choose **Advanced installation options** > **Choose an installation option** > **Custom**, and then choose the **Server** and **SQL Server Database Components** options.	Install Business Central Using Setup
On the other computer, install the Business Central Web Server components.	Run Business Central Setup, choose **Advanced installation options** > **Choose an installation option** > **Custom**, and then the **Web Server Components** option.	Install Business Central Using Setup Business Central Web Server Overview
Configure delegation from the web server to Business Central Server.	Because Business Central Server is running on a different computer than the Business Central Web Server components, you must configure the computer that is running Business Central Web Server components to delegate its access to Business Central Server on behalf of the device trying to access from the Business Central Web client.	Configure Delegation for Business Central Web Server

Post-installation Tasks

The following table includes tasks to configure the Business Central Web Server components after installation. These tasks are optional depending on your organizational and network requirements.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Change the user authentication method.	The Business Central supports the following authentication methods: Windows, UserName, NavUserPassword, and AccessControlService. By default, Windows authentication is used.	Authentication and User Credential Type
Secure the connection to the Business Central Web client with SSL.	You can help secure Business Central data that is transmitted over the Internet by enabling Secure Sockets Layer (SSL) on the connection to the Business Central Web client.	Configure SSL to Secure the Web Client Connection
Change the configuration of the Business Central Web Server.	There are several parameters in the navsettings.json configuration file for the Business Central Web Server that you can modify to change the behavior of the Business Central Web client. Some of the more common parameters include the Business Central Server instance, company, language, time zone, regional settings, session time out, and online Help URL.	Configuring Business Central Web Server

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Set up multiple Business Central Web client applications on a website.	You can set up multiple web server instances for the Business Central Web client on the existing website. The web server instances will use the same address (URL) except with an alias that specifies the specific application.	Creating and Managing Business Central Web Server Instances Using PowerShell
Configure web browsers on devices.	The Business Central Web client supports several different web browsers. To access the Business Central Web client, the web browser must be enabled on a device with cookies and JavaScript.	Web Client Requirements

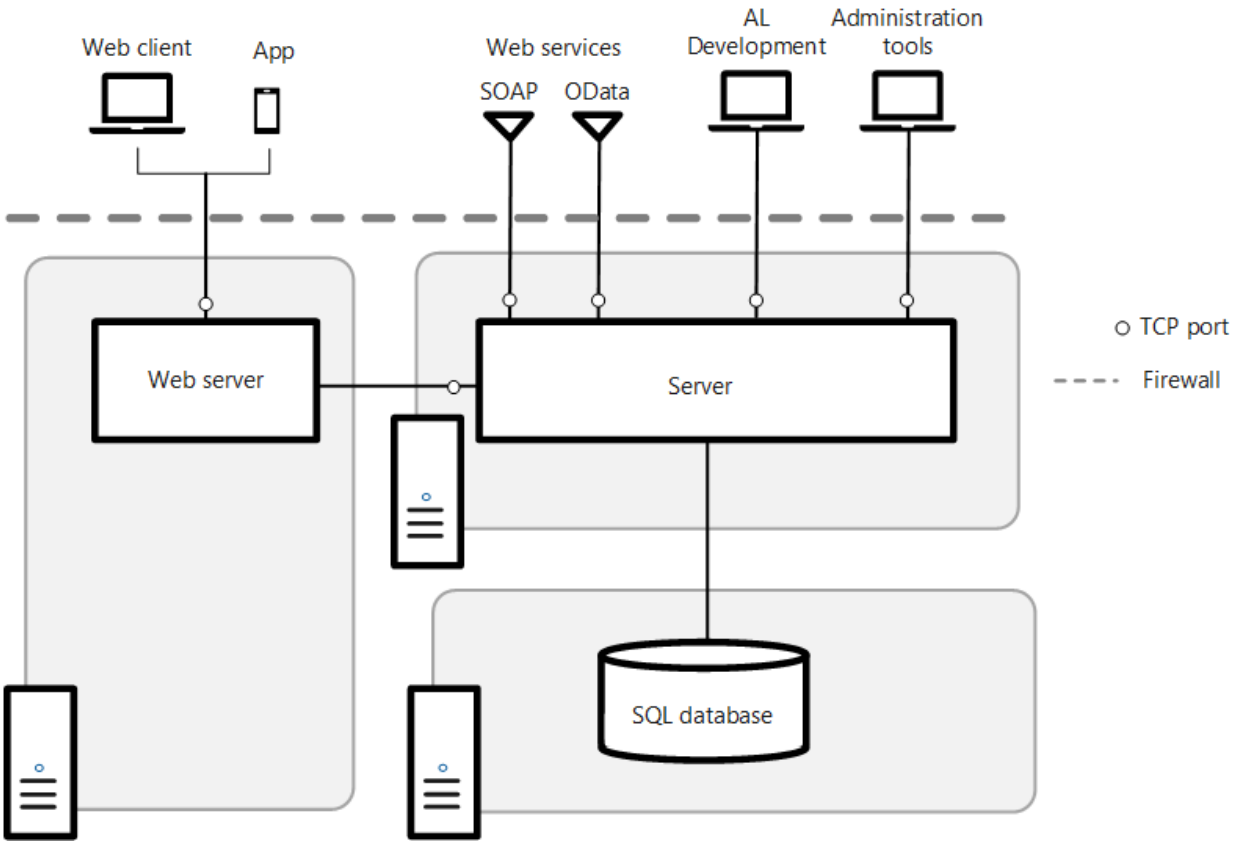
See Also

Business Central Web Server Overview
Installing Business Central in a Single Computer Environment
Installing Business Central in a Three Computer Environment

Deploying Business Central in a Three-Computer Topology

2/17/2021 • 5 minutes to read • Edit Online

In this scenario, you install the Business Central Web Server components, Business Central Server, and the SQL Server database components on separate computers.



This article also applies to deploying the Business Central phone client and Business Central tablet client.

Pre-Installation Tasks

The following table includes tasks to perform before you install the Business Central Web Server components.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Make sure that system requirements are met.	Verify that the computer has the required hardware and software installed.	System Requirements

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Install Internet Information Services.	When you install the Business Central Web Server components, Business Central Setup creates a website for the Business Central Web client on IIS. If IIS is already installed, then make sure that the required features are enabled. **Note:** This step is optional because instead of installing and configuring IIS manually, you can use Business Central Setup to install IIS and enable the required features by setting the **Install IIS Prerequisites** option to **Yes**.	Configure Internet Information Services
Determine the TCP ports for the Business Central Web client, client services, and SOAP/OData web services (optional) and allow communication on the port through Windows Firewall.	Business Central Setup creates a website on IIS. During Setup, you will have to choose the port to use for the site. The default port is port 8080. The default client services port is 7046. If you will enable SOAP and OData web services, you will also need to specify a port for each. The default ports are 7047 and 7048. If you choose to do so, Business Central Setup will automatically create an inbound rule in Windows Firewall that allows communication on the ports. Otherwise, you will have to do this manually.	Create an Inbound Port Rule in the Windows documentation.
Set up the service account for Business Central Server and the SQL Server database.	Optional. When you install Business Central Server, you can specify a user account that will be used to log on to the Business Central Server instance and Business Central database. The default service account is Network Service. If you want to use Network Service, then no action is required for this task.	Provisioning a Service Account
Obtain and install an SSL certificate.	Optional. If you want to configure SSL on the connection to Business Central Web client, then complete the following procedures: - Obtain an SSL certificate. - Import the certificate into the local computer store of the computer on which you will install the Business Central Web Server components. - Obtain the certificate's thumbprint. **Note:** You can also configure SSL after you have installed the Business Central Web client. For more information, see Post-installation Tasks.	Configure SSL to Secure the Web Client Connection

Installation Tasks

The following table includes tasks for installing the Business Central Web client.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
On the first computer, install the Business Central database components.	Run the Business Central Setup setup.exe file, choose **Advanced installation options** > **Choose an installation option** > **Custom**, and then choose the **SQL Server Database Components** option.	Install Business Central Using Setup
Start the SQL Server Browser Service on the SQL Server computer.	This task is only required if you are using a named database instance for Business Central. By default, Business Central uses the database instance named NAVDEMO. The SQL Server Browser Service is required so that the database instance can be discovered by the Business Central Server instance, which in this scenario, is another computer. To start the SQL Server, use SQL Server Configuration Manager.	Start, Stop, Pause, Resume, Restart SQL Server Services
On second computer, install Business Central Server.	Run the Business Central Setup setup.exe file, choose **Advanced installation options** > **Choose an installation option** > **Custom**, and then choose the **Server** option.	Install Business Central Using Setup
On the third computer, install the Business Central Web Server components.	Run the Business Central Setup setup.exe file, choose **Advanced installation options** > **Choose an installation option** > **Custom**, and then choose **Web Server Components** option.	Install Business Central Using Setup
Configure delegation from the web server to Business Central Server.	Because Business Central Server is running on a different computer than the Business Central Web Server components, you must configure computer running Business Central Web Server components to delegate its access to Business Central Server on behalf of the device trying to access from the Business Central Web client.	Configure Delegation for Business Central Web Server

Post-installation Tasks

The following table includes tasks that configure the Business Central Web Server components after installation. These tasks are optional depending on your organizational and network requirements.

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE

TASK	DESCRIPTION	FOR MORE INFORMATION, SEE
Change the user authentication method.	The Business Central supports the following authentication methods: Windows, UserName, NavUserPassword, and AccessControlService. By default, Windows authentication is used.	Authentication and User Credential Type
Secure the connection to the Business Central Web client with SSL.	You can help secure Business Central data that is transmitted over the Internet by enabling Secure Sockets Layer (SSL) on the connection to the Business Central Web client.	Configure SSL to Secure the Web Client Connection
Change the configuration of the Business Central Web Server.	There are several parameters in the navsettings.json configuration file for the Business Central Web Server that you can modify to change the behavior of the Business Central Web client. Some of the more common parameters include the Business Central Server instance, company, language, time zone, regional settings, session time out, and online Help URL.	Configuring Business Central Web Server
Set up multiple Business Central Web client applications on a website.	You can set up multiple web server instances for the Business Central Web client on the existing website. The web server instances will use the same address (URL) except with an alias that specifies the specific application.	Creating and Managing Business Central Web Server Instances Using PowerShell
Configure web browsers on devices.	The Business Central Web client supports several different web browsers. To access the Business Central Web client, the web browser must be enabled on a device with cookies and JavaScript.	Web Client Requirements

See Also

Business Central Web Server Overview
Installing Business Central in a Single Computer Environment
Installing Business Central in a Two Computer Environment

Installing Business Central Using Setup

2/17/2021 • 6 minutes to read • Edit Online

You use Business Central Setup to install the different components that comprise a Business Central production, demonstration, or development environment. For a list of components, see Components and Topology.

About Setup

Setup is available on the installation media (DVD) in the setup.exe file. When you run the setup.exe file, a wizard leads you through installation process. You can install individual components or select predefined options that install a logical collection of components.

Configuration settings

During Setup, you're presented with various configuration settings. Some settings require that you set them. Other settings have a default value. The default value in many cases is sufficient for the initial installation. After you run Setup, you can change the configuration settings by using other tools such as the Business Central Server Administration tool and Business Central Administration Shell.

Prerequisite Installations by Setup

There are some components that require other software to run. For example, the database requires SQL Server and the Web client requires IIS. Setup will install several of these prerequisites, like installing SQL Server Express and enabling IIS. You can see which prerequisites Setup installs in the System Requirements.

Downloading Business Central for installation

Business Central is available for downloading from Microsoft Support. For each major release, minor updates are published on a regular basis. The downloaded files contain the installation media, which includes the setup.exe file.

> **IMPORTANT**
> We recommend that you install the latest update for the release you want to install. However, if you are installing a version for upgrade, make sure that you choose a target version that is compatible with the version that you will be upgrading. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

Download the files

1. Go to the update page for the release that you want to install:
   - Business Central 2018
   - Business Central Spring 2019
   - Business Central 2019 Release Wave 2
   - Business Central 2020 Release Wave 1
2. In the **Cumulative Updates** table, select the link in the **Knowledge Base ID** column for the update you want.
3. In the **Resolution** section, select the link under **How to obtain the Microsoft Dynamics 365 Business Central files**.
4. Follow the instructions.

Before you run Setup

1. Plan your deployment and identify the components that you want to install.
2. Verify that the target computer meets the hardware and software requirements for the components that you want to install. For more information, see System Requirements.
3. Make sure that you're an administrator on the computer where you run Setup.
4. Determine the HTTP ports that you'll use for components.

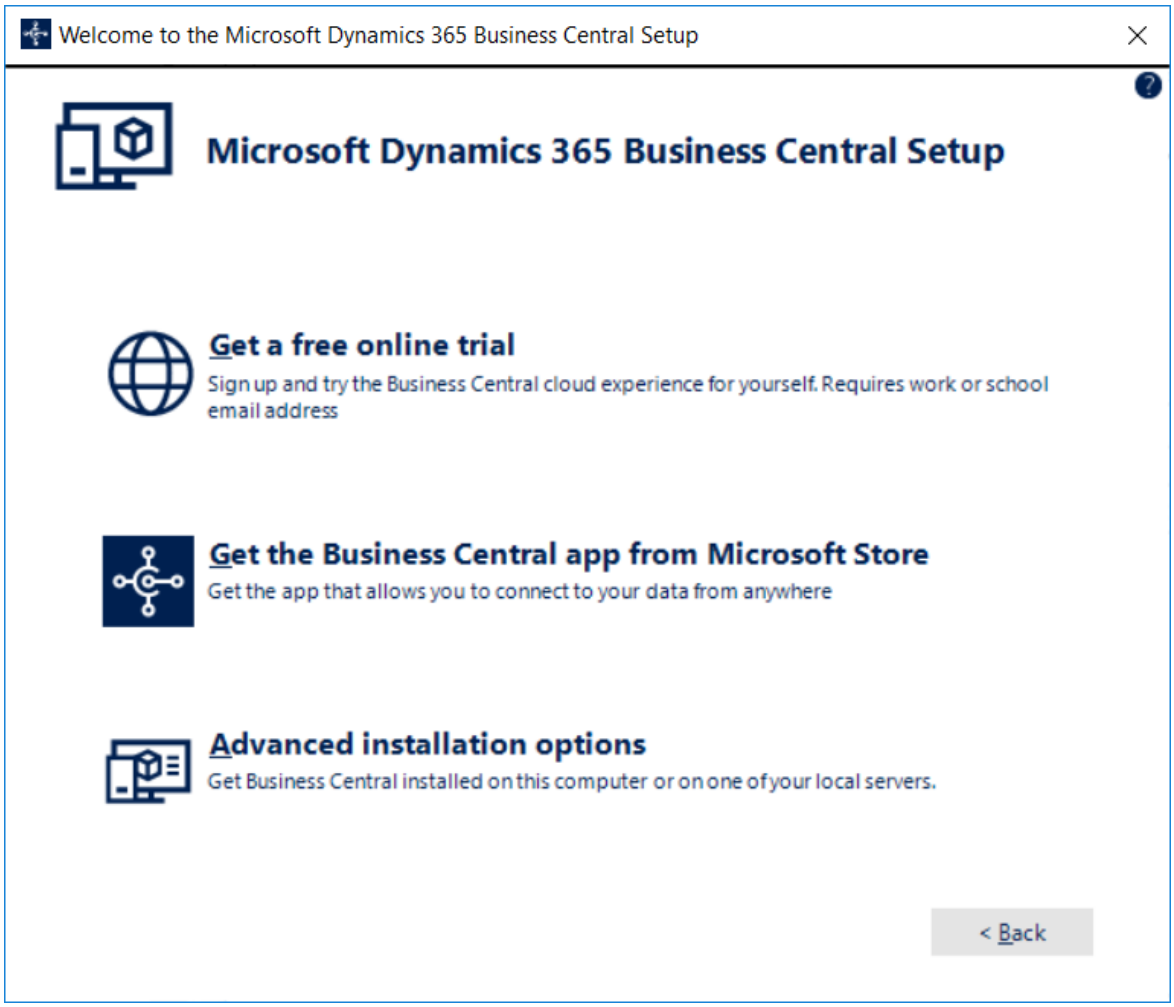
   During setup, you'll have to specify the following HTTP ports:

PORT	DESCRIPTION	DEFAULT
Management services	The listening TCP port for the Business Central Server Administration tool.	7045
Client services	The listening HTTP port for client services.	7046
SOAP services	The listening HTTP port for SOAP web services.	7047
OData service	The listening HTTP port for OData web services.	7048
Developer services	The listening HTTP port for Microsoft Dynamics NAV Developer web services	7049

   Select a port number that isn't being used by another service. Or, if you use a port that's used by another service, stop the other service before you run setup. For example, if you have an older version of Dynamics NAV Server or Business Central Server installed, then configure the new server instance to use other ports than the old server instance, or stop the old server instance before you install the new one.

Run Setup

1. In the installation media (DVD) folder, double-click the setup.exe.
2. Follow Setup until you get to the **Dynamics 365 Business Central** page.



- Choose **Get a free online trial to sign up** if you interested in hearing about and trying the cloud experience.
- Choose **Get the Business Central app from the Microsoft Store** to download a companion app that mimics that Web client but has the same look-and-feel as the mobile apps. For more information, see Installing the Microsoft Dynamics 365 Business Central App.
- Choose **Advance installation options** to install a demonstration environment or individual components. Then, follow the on-screen instructions to complete the installation.

Cancel Setup

Setup doesn't provide a **Cancel** button on all pages. But, you can cancel the installation from any page by choosing the **Close** button. All Business Central components are removed from the computer. The only software that Setup can't remove are:

- Database files, such as the Demo database.
- Prerequisites for Business Central components that Setup can install, such as the .NET Framework.

Run Setup from a command prompt

You can run Business Central Setup from a command prompt. Before installation, run it from the installation media. After the initial installation, you can run it from the location where it's automatically stored on your computer. The default location is:

```
C:\Program Files (x86)\Common Files\Microsoft Dynamics 365 Business Central\<Version number>\Setup
```

You can use the following options with Setup.exe.

SETUP OPTION	DESCRIPTION
/config <Setup config file>	Specifies path and file name information for a Setup configuration file to load. This option is the only required option.
/help	Displays Help about Setup.exe options.
/log <log path>	Specifies path and file name information for a Setup log file to be created by Setup. The file must not exist before you run Setup.
/quiet	Specifies that Setup doesn't display anything on the screen. All configuration information is taken from the specified configuration file.
/repair	Repairs the current installation of Business Central.
/uninstall	Removes the current installation Business Central.

Save, Edit, and Load a Setup Configuration File

During Setup, you can save the configuration settings to a file before you finish. Then later, you can load the configuration file. Using a configuration file makes it faster to replicate the same configuration for another deployment.

Save to a Setup configuration file

1. Choose **Save** on the **Specify parameters** page in Setup. This page is available when you run Setup unless you select **Install Demo**, which skips all other Setup pages.
2. Type a file name for the configuration file. An .xml extension is added automatically.
3. Choose **Save**.

   You now return to the **Specify parameters** page, where you can continue with installing software. You can also close Setup if you only have to create a Setup configuration file.

Edit a Setup configuration file

You edit the file using an XML editor or text editor. Setup configuration files contain two types of settings.

SETTING TYPE	PURPOSE
Component	For each component, there are three separate values, all displayed on a single line: - **ShowOptionNode** Specifies whether the component should be displayed in Setup. For silent installs, this parameter isn't relevant. - **State** There are two possible values: **Local**, indicates that the component is included in the install. **Absent** indicates that the component isn't included. - **Id** Identifies the component You can change value for **State** or **ShowOptionNode**, but not for **Id**. Also, you can't add or remove a component.

SETTING TYPE	PURPOSE
Parameter	These settings contain configuration information for components. As with Components, you can modify a parameter's **Value**, but not its **Id**.

Load a Setup configuration file

The option to load a Setup configuration file is on the **Choose an installation option** page in Setup.

> **NOTE**
>
> If you are using a Setup configuration file that was created from an earlier version of Business Central or Dynamics NAV, be aware that there might be some elements that are no longer supported because the feature has been deprecated. For example, the elements that have the following IDs are no longer supported as og 2019 release wave 2: "RoleTailoredClient", "ExcelAddin, "ClassicClient", "ClickOnceInstallerTools", "STOutlookIntegration", "PublicWinBaseUrl", and "ACSUri".

1. On the **Choose an installation option** page, choose **Load Configuration**.

   This option is located under **Custom Components**.

   > **IMPORTANT**
   >
   > A Setup configuration file contains information about which components to install and which settings to apply to each component. Therefore, you should not customize the list of components or configure components in Setup before you load a Setup configuration file because loading the configuration overwrites all prior customization and configuration.

2. In the **Open** dialog box, select or browse to the Setup configuration file that you want to open. Then, double-click the file.

   Setup now shows the **Customize the installation** page. It's modified according to the component selection in the configuration file.

3. Modify the list of components to install or choose **Next** to continue to the **Specify parameters** page.

4. Configure these settings or choose **Apply** to accept these values and continue.

See Also

Components
Deployment

Provisioning the Business Central Server Service Account

2/17/2021 • 9 minutes to read • Edit Online

The Business Central Server account is used by Business Central clients to log on to the Business Central Server instance. The Business Central Server then uses the service account to log on to the Business Central database. When you install Business Central Server, you identify an Active Directory account to provide credentials for the server. By default, Setup runs Business Central Server under the Network Service account, a predefined local account used by the service control manager. This account has minimum privileges on the local computer and acts as the computer on the network.

Domain user account versus Network Service account

We recommend that you create a domain user account for running Business Central Server. The Network Service account is considered less secure because it is a shared account that can be used by other unrelated network services. Any users who have rights to this account have rights to all services that are running on this account. If you create a domain user account to run Business Central Server, you can use the same account to run SQL Server, whether or not SQL Server is on the same computer.

There is no specific action required for provisioning the Network Service account. The only recommendation is to verify that the account has the necessary database privileges in SQL Server as described in the Giving the service account database privileges in SQL Server section.

Provisioning a domain user account

If you are running the Business Central Server under a domain user account, you must:

- Enable the account to log in as a service
- Enable the account to register an SPN on itself
- Add the account to the SMSvcHost.exe.config file
- Give the account necessary database privileges in SQL Server

Prerequisite

Delete the **Dynamics 365 Business Central** folder in the **ProgramData** folder of your system drive, for example, `C:\ProgramData\Microsoft\Microsoft Dynamics 365 Business Central`.

The **ProgramData** is folder is typically hidden, so you might have to change the folder options for your system drive to show hidden files, folders, and drives.

Enabling the account to log in as a service

Depending on various factors, the account may already have this ability to log in as a service. For example, if you have already installed SQL Server and configured it to run under the same account, SQL Server will have modified the account to log in as a service.

When this permission is lacking, Business Central Server instances may not be able to start.

For instructions for enabling an account to log in as a service, see Manage User Accounts in Windows Server.

Enabling the account to register an SPN on itself

To enable secure mutual authentication between clients and Business Central Server, you must configure the Business Central Server account to self-register Service Principal Names (SPNs). Mutual authentication is recommended in a production environment but may not be necessary in a testing or staging environment. This is done by modifying the account in Active Directory.

For more information, see Service Principal Names in the Active Directory documentation.

Add the account to the SMSvcHost.exe.config file

Business Central uses Net.TCP Port Sharing Service, which is managed by SMSvcHost.exe. The SMSvcHost.exe.config contains information about the identities (or accounts) that can use the service. These accounts are specified as security identifiers (SIDs) in the section of the SMSvcHost.exe.config file. By default, permission is implicitly granted to system accounts, such as NetworkService. For other accounts, you must explicitly add the SID for the account to the SMSvcHost.exe.config file as follows:

1. Get the SID of the user account.

   The SID is an alphanumeric character string, such as S-1-5-20 or S-1-5-32-544. There are different ways to get the SID, such using Windows Management Instrumentation Control Command-line (WMIC) or the computer's registry.

   - To use WMIC, open a command prompt, and run the following command:

     ```
     wmic useraccount get name,sid
     ```

     This will display a list of user accounts and their SIDs.

   - To use the registry, run regedit, and then go to the *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList* folder. This folder list the SIDs for each user account. To find the SID that corresponds to the user account that you want, look at the *ProfileImagePath* key data.

2. Using a text editor, open the SMSvcHost.exe.config file.

   You will find the SMSvcHost.exe.config file in the installation folder for the latest :NET Framework version on the Business Central Server computer; for example, `C:\Windows\Microsoft.NET\Framework64\v4.0.30319`.

3. Add the SID to the element as follows, and then save the file:

   ```
   <system.serviceModel.activation>
       <net.tcp listenBacklog="10" maxPendingConnections="100" maxPendingAccepts="2"
   receiveTimeout="00:00:10" teredoEnabled="false">
           <allowAccounts>
               <!-- Your Business Central Server account -->
               <add securityIdentifier="N-N-N-N"/>
           </allowAccounts>
       </net.tcp>
   ```

For more information about SMSvcHost.exe and the SMSvcHost.exe.config file, see Configuring the Net.TCP Port Sharing Service.

There is not action required for provisioning the Network Service account. The only recommendation is to verify that the account has the necessary database privileges in SQL Server as described in the next section.

Giving the service account database privileges in SQL Server

The Business Central Server service account must have specific roles and permissions in SQL Server to access a Business Central database. The roles and permissions are applied on the server-level and database-level, as

outlined in the following table and explained in detail in the sections that follow:

LEVEL	ROLES AND PERMISSIONS
Server-level	Login: **dbcreator** role On master database: **Select** permission on the database **Select** permission on the dbo.ndosrvproperty table
Database-level	**db_datareader**, **db_datawriter**, and **db_ddladmin** roles **View change tracking** permission on the database schema

When you install the Business Central database by using Business Central Setup or the New-NAVDatabase cmdlet, you can specify the Business Central Server account. In these cases, the server account that you specify will be given privileges in SQL Server. However, we recommend that you use the guidelines in this section to ensure that the service account has the minimum required privileges.

To verify server-level and database-level privileges on SQL Server after you create your Business Central database, use SQL Server Management Studio and, if necessary, modify privileges. If you do not already have it, you can download and install SQL Server Management Studio from here.

Assign privileges on the server-level

On the server-level, the service account must be set up with a login that has the following roles and permissions:

- **dbcreator** server role

  This privilege is only required for database creation. Consider removing it from the service account once the system has been set up. Or run the setup operations of the Business Central database with a service account that has this extended privilege.

- On the master database, the service account must have the following permissions:
  - **Select** permission on the database
  - **Select** permission on the **dbo.ndosrvproperty** table of the database

To assign privileges on the server-level, complete the following tasks in SQL Server Management Studio:

1. Start SQL Server Management Studio and connect to the instance where the Business Central database is installed.
2. Create a login for the Business Central Server account:
   a. Navigate the tree view: **Security**, **Logins**.
   b. Right-click **Logins** and select **New Login**.
   c. Choose **Search** and use the **Select User or Group** dialog box to identify the Business Central Server account.
   d. Choose **OK** to exit the New Login dialog box.
3. Grant the login the server-level role **dbcreator**:
   a. Navigate the tree view: **Security**, **Logins**.
   b. Right-click the Business Central Server account, and then choose **Properties**.
   c. Click on **Server Roles**.
   d. Check the **dbcreator** box.

e. Choose **OK**.

4. Add the login as a user on the master database:

   a. Navigate the tree view: **Databases**, **System Databases**, **master**, **Security**, **Users**.
   b. Right-click **Users** and choose **New User**.
   c. Choose the ellipse button at the far right of the second line in the **Database User – New** dialog box.
   d. In the **Select Login** dialog box, enter or browse for the login you created for the Business Central Server account.
   e. Enter a name in the **User name** field (the first line in the **Database User - New** dialog box).
   f. Choose **OK** to exit the **Database User - New** dialog box.

5. Grant **Select** permission to Business Central Server login on the master database.

   a. In the tree view, right-click **master** and choose **Properties**. Then do the following in the **Database Properties – master** dialog box.
   b. Under **Select a Page**, choose **Permissions**.
   c. Under **Name**, choose the login you created for the Business Central Server account name.
   d. Under **Permissions for <username>**, on the **Explicit** tab, scroll down to down to the **Select** line, and select the check box in the **Grant** column.
   e. Choose **OK** to exit the **Database Properties – master** dialog box.

6. Grant **Select** permission to Business Central Server login on the dbo.ndosrvproperty table.

   a. Navigate the tree view: **Databases**, **System Databases**, **master**, **Tables**, **System Tables**.
   b. Right-click the **dbo.ndosrvproperty** table and choose **Properties**.
   c. Under **Select a Page**, choose **Permissions**.
   d. Choose **Search** and use the **Select User or Group** dialog box to identify the login for the Business Central Server account.
   e. Under **Permissions for <username>**, on the **Explicit** tab, scroll down to down to the **Select** line, and select the check box in the **Grant** column.
   f. Choose **OK** to exit the **Table Properties – dbo.ndosrvproperty** dialog box.

Assign privileges on the Business Central database-level

On the Business Central database, the service account must have the following roles and permissions:

- **db_datareader**, **db_datawriter**, and **db_ddladmin** database roles
- **View change tracking** permission on the database schema (dbo)

You can set permissions directly on the database user that you set up for service account. However, we recommend that you create a role that includes the permissions, and then assign the user to the role.

To set up these permissions, complete the following steps:

1. Add the login as a user on the database and assign database roles

   a. Navigate the tree view: **Databases**, **<your Business Central database>**, **Security**, **Users**.
   b. Right-click **Users** and choose **New User**.
   c. In the **Database User – New** dialog box, choose the ellipse button at the far right of the second line.
   d. Choose the login you created for the Business Central Server account name, and then choose **OK**.

2. Create a database role for the runtime permissions on the database:

   a. In the tree view, under the database, navigate to **Security**, **Roles**.

   b. Right-click **Database Roles** and choose **New Database Role**.

   c. Choose the **General** page, and then:

a. Enter a role name, such as **BCServer_runtime**.
b. Under **Schemas owned by the role**, select **db_datareader**, **db_datawriter**, and **db_ddladmin**.
c. Under **Members of this role**, choose **Add**, then browse for and select the server account user.

d. Choose the **Securables** page, and then:

a. Choose **Search**.
b. Choose **All objects of the types...**, and then **OK**.
c. Select **Schemas**, and then **OK**.
d. Under **Securables**, select the **dbo** schema.
e. Under **Permissions for dbo**, select **Grant** for the **View change tracking** permission.
f. Choose **OK** to finish.

Sample SQL queries

Instead of manually completing the previous steps, you can use the following SQL queries, which you can execute separately or combine and run as one.

Server-level query

```
USE [master]
GO

CREATE LOGIN [domain\accountname] FROM WINDOWS
CREATE USER [domain\accountname] FOR LOGIN [domain\accountname]
GRANT SELECT ON [master].[dbo].[$ndo$srvproperty] TO [domain\accountname]
ALTER SERVER ROLE [dbcreator] ADD MEMBER [domain\accountname]
GO
```

Database-level query

```
USE [Business Central Database]
GO

CREATE ROLE bc_server_runtime
ALTER ROLE db_datareader ADD MEMBER bc_server_runtime
ALTER ROLE db_datawriter ADD MEMBER bc_server_runtime
ALTER ROLE db_ddladmin ADD MEMBER bc_server_runtime
GRANT VIEW CHANGE TRACKING on schema::[dbo] TO  bc_server_runtime
GRANT VIEW DATABASE STATE TO bc_server_runtime
GRANT ALTER ON DATABASE::[Business Central Database] TO bc_server_runtime
GO

CREATE USER [domain\accountname] FOR LOGIN [domain\accountname]
GO

ALTER ROLE bc_server_runtime ADD MEMBER [domain\accountname]
GO
```

See Also

Creating Databases
Configuring Business Central Server

Using Security Certificates with Business Central On-Premises

2/17/2021 • 7 minutes to read • Edit Online

> **NOTE**
> Dynamics NAV Client connected to Business Central is **DISCONTINUED AFTER:** Business Central Spring 2019.

You use certificates to help secure connections over a wide area network (WAN), such as connections from the Business Central Web Server, Dynamics NAV Client connected to Business Central, and web services to the Business Central Server. Implementing security certificates on your deployment environment requires modifications to various components, like the Business Central Server, Business Central Web Server, and clients.

About Security Certificates

A certificate is a file that Business Central Server uses to prove its identity and establish a trusted connection with the client that is trying to connect. Business Central can support the following configurations:

- *Chain trust*, which specifies that each certificate must belong to a hierarchy of certificates that ends in a root authority at the top of the chain.
- *Peer trust*, which specifies that both self-issued certificates and certificates in a trusted chain are accepted.

The implementation in this section describes the chain trust configuration, which is the more secure option.

> **NOTE**
> An instance of Business Central Server that has been configured for secure WAN communication always prompts users for authentication when they start the client, even when the client computer is in the same domain as Business Central Server.

Certificates for Production

In a production environment, you should obtain a certificate from a certification authority or trusted provider. Some large organizations may have their own certification authorities, and other organizations can request a certificate from a third-party organization.

Obtaining Certificates

You implement chain trust by obtaining X.509 service certificates from a trusted provider. These certificates and their root certification authority (CA) certificates must be installed in the certificates store on the computer that is running Business Central Server. The CA certificate must also be installed in the certificate store on computers that are running the Business Central Web Server and Dynamics NAV Client connected to Business Central so that clients can validate the server.

Most enterprises and hosting providers have their own infrastructure for issuing and managing certificates. You can also use these certificate infrastructures. The only requirement is that the service certificates must be set up for key exchange and therefore must contain both private and public keys. Additionally, the service certificates that are installed on Business Central Server instances must have the Service Authentication and Client Authentication certificate purposes enabled.

IMPORTANT

Microsoft recommends against using wildcard SSL certificates in Business Central installations. Wildcard certificates pose security risks because if one server or sub-domain is compromised, all sub-domains may be compromised. Wildcard certificates also introduce a new style of impersonation attack. In this attack, the victim is lured to a fraudulent resource in the certified domain through phishing. Conventional certificates detect this attack, because the user's browser checks that the private key is hosted on a server whose name matches the one displayed in the browser's address window.

Run the Certificates Snap-in for Microsoft Management Console

Some of the following procedures use the Certificates snap-in for Microsoft Management Console (MMC). If you do not already have this snap-in installed, you can add it to the MMC. For information see Add the Certificates Snap-in to an MMC.

Install and Configure the Certificates

You install the security certificates on the computers running Business Central Server, Business Central Web Server, and Dynamics NAV Client connected to Business Central. The root CA certificate and the service certificate are used in the configuration, but client certificates are not.

Install Certificates on components

1. Follow the installation instructions that are available from your certificate provider to install the root CA and service certificates on the following computers:

   - Install the root CA on the computer that is running Business Central Server and all computers that are running Business Central Web Server instances and Dynamics NAV Client connected to Business Central.
   - Install the service certificate on the computer that is running Business Central Server only.

2. Make sure that the **Server Authentication** and **Client Authentication** certificate purposes are enabled for the service certificate.

   A certificate can be enabled for several different purposes. The **Server Authentication** and **Client Authentication** purposes must be enabled. You can enable or disable other purposes to suit your requirements.

   You enable certificate purposes by using the Certificates Snap-in for MMC. For more information, see Modify the Properties of a Certificate.

Grant access to the Business Central Server service account

After you have installed the root CA and the service certificate on the computer running Business Central Server, you must grant access to the service account that is associated with the server so that the service account can access the service certificate's private key.

1. In the left pane of MMC, expand the **Certificates (Local Computer)** node, expand the **Personal** node, and then select the **Certificates** subfolder.
2. In the right pane, right-click the certificate, select **All Tasks**, and then choose **Manage Private Keys**.
3. In the **Permissions** dialog box for the certificate, choose **Add**.
4. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, enter the name of the dedicated domain user account that is associated with Business Central Server, and then choose the **OK** button.
5. In the **Full Control** field, select **Allow**, and then choose the **OK** button.

6. In the right pane, select the certificate.
7. In the **Certificate** dialog box, choose the **Details** tab, and then select the **Thumbprint** field.
8. Copy the value of **Thumbprint** field.

   For example, copy the hexadecimal characters to text editor, such as Notepad. Delete all spaces from the thumbprint string. If the thumbprint is `c0 d0 f2 70 95 b0 3d 43 17 e2 19 84 10 24 32 8c ef 24 87 79`, then change it to `c0d0f27095b03d4317e219841024328cef248779`.

> **TIP**
> It is important that the thumbprint does not contain any invisible extra characters; otherwise you will experience problems when using it later. To avoid this, see Certificate thumbprint displayed in MMC certificate snap-in has extra invisible unicode character.

Configure the Business Central Server instance

The Business Central Server instance configuration includes several settings for certificates and enabling remote logins. You can modify a server instance by using Business Central Server Administration tool or Business Central Administration Shell. For details about how to modify a server instance, see Configuring Business Central Server.

1. Run the Business Central Server Administration tool.
2. Under **General**, change the following settings for the Business Central Server instance.

SETTING	NEW VALUE	DESCRIPTION
Credential Type	`NavUserPassword`, `Username`, or `AccessControlService`	The default value is `Windows`. When you change it to `NavUserPassword`, `Username`, or `AccessControlService`, client users who connect to the server are prompted for user name and password credentials.
Certificate Thumbprint	Value of the **Thumbprint** field in the previous procedure.	Remove any leading or trailing spaces in the thumbprint.

3. If you want to use secure web services, then under **SOAP Services** and **OData Services**, select the **Enable SSL** check box.
4. Save and the new values for the server instance.
5. Restart the Business Central Server instance.

   If there is a problem, see Windows Event Viewer.

Configure the Business Central Web Server and Dynamics NAV Client connected to Business Central

The chain trust configuration allows client users to log on to one or more instances of Business Central Server as long as their login credentials have been associated with user accounts in Business Central. The client validates that the server certificate is signed with the root CA.

After you have installed the root CA on the computer running the Business Central Web Server or Dynamics

NAV Client connected to Business Central, you must modify the client configuration file.

Modify the Business Central Web client configuration file

1. On the computer that is installed the Business Central Web Server, open the navsetting.json configuration file in a text editor, such as Notepad.

2. Change the following settings:

KEY	NEW VALUE	DESCRIPTION
ClientServicesCredentialType	`NavUserPassword`, `Username`, or `AccessControlService`	The default value is `Windows`. When you change it to `NavUserPassword`, `Username`, or `AccessControlService`, client users who connect to the server are prompted for user name and password credentials.
DnsIdentity	The subject name of the service certificate	The default value is <identity>. Replace this with the subject name or common name (CN) of the certificate that is used on the computer that is running Business Central Server.

3. Save the navsettings.json configuration file.

Modify the Dynamics NAV Client connected to Business Central configuration file

1. Open the ClientUserSettings.config configuration file.

   The location of this file is *Users\<username>\AppData\RoamingLocal\Microsoft\Dynamics 365 Business Central*.

   By default, this file is hidden. Therefore, you may have to change your folder options in Windows Explorer to view hidden files.

   > **NOTE**
   > If you want to change default Dynamics NAV Client connected to Business Central settings for all future users, edit the default ClientUserSettings.config file — that is, the one in C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160. Be sure that you run your text editor with Administrator privileges when you do so.

2. Modify the following settings.

KEY	NEW VALUE	DESCRIPTION
ClientServicesCredentialType	`NavUserPassword`, `Username`, or `AccessControlService`	The default value is `Windows`. When you change it to `NavUserPassword`, `Username`, or `AccessControlService`, client users are prompted for user name and password credentials.

KEY	NEW VALUE	DESCRIPTION
DnsIdentity	The subject name of the service certificate.	The default value is <identity>. Replace this with the subject name or common name (CN) of the certificate that is used on the computer that is running Business Central Server.

3. Save and close the ClientUserSettings.config file.

When starting the Dynamics NAV Client connected to Business Central, users are prompted for a valid user name and password.

See Also

Authentication and User Credential Types

Business Central Web Server Overview

2/17/2021 • 4 minutes to read • Edit Online

Giving users access to data from the Business Central Web client, companion app, and Outlook add-in requires a Internet Information Services (IIS) website as part of your deployment. The website, which we refer to as Business Central Web Server instance, hosts the files that provide content and services to client users over the Internet. This article highlights several factors to consider to help you set up Business Central Web Server instances that suit your deployment requirements.

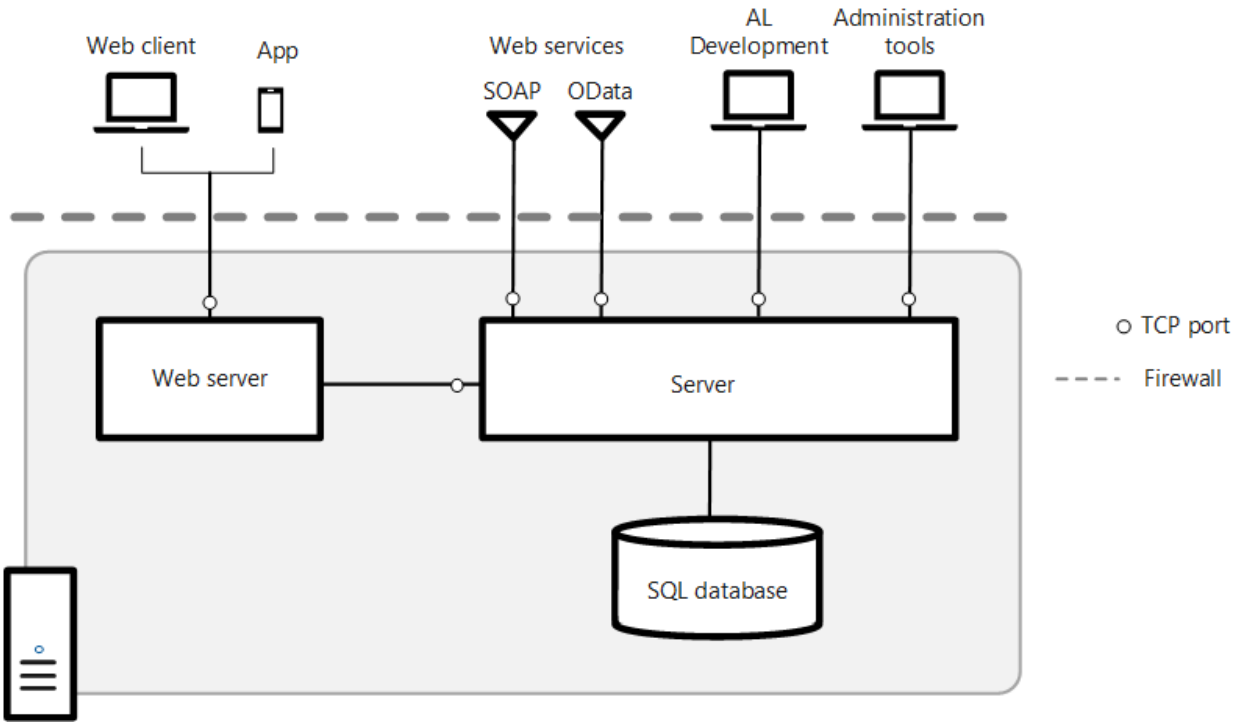
If you just want to get started installing the Business Central Web Server components, see Install Business Central Using Setup.

ASP .NET Core on IIS

Business Central Web Server instances run on ASP.NET Core on IIS, which in part dictates the directory structure of the instances. For more information about ASP .NET Core, see Introduction to ASP.NET Core.

Network Topology

The following illustration shows the component infrastructure that supports Business Central Web Server instances on your network.



Each Business Central Web Server instance must connect to a Business Central Server, which in turn connects to the database that contains the application and business data. Multiple Business Central Web Server instances can connect to the same Business Central Server. You can deploy these components on one computer or on separate computers. For example, you can install the Business Central Web Server instance on one computer and the Business Central Server and SQL Server database on another computer. The topology that you choose depends on the network resources and the infrastructure of the Business Central components. The installation and configuration process is different for each scenario.

For information about the common deployment scenarios, see Deployment Topologies.

Creating a Business Central Web Server instance

There are two ways to create a Business Central Web Server instance. You can use the Business Central Setup Setup or the Business Central Web Server PowerShell cmdlets.

Using Business Central Setup Setup

Setup is the quickest way to get a web server instance up and running, and is typically how you install the first Business Central Web Server instance in your deployment.

- Setup installs the Business Central Web Server components, which does the following:
  - Installs and configure IIS with the required prerequisites, including Microsoft .NET Core - Windows Server Hosting
  - Installs a web server instance on IIS.
  - Installs components and files in a **WebPublish** folder that enables you to add additional web server instances without having to use the Business Central installation media (DVD).
- You can only use Setup to install a single Business Central Web Server instance.
- Setup does not let you choose the site deployment type for the web server instance. By default, it creates a SubSite instance. For more information, see Site Deployment Types.
  For information about how to install the Business Central Web Server components, see Install Business Central Using Setup.

Using Business Central Web Server PowerShell cmdlets

There are several PowerShell cmdlets that enable you to create, configure, and remove Business Central Web Server instances from a command line interface. To create a web server instance, you use the New-NAVWebServerInstance cmdlet, which has the following advantages over Setup:

- You can create multiple web server instances.
- You have more flexibility regarding the site deployment type of the Business Central Web Server instances on IIS. For example, you can create a root-level website instance or a subsite application instance under a container website.

> **IMPORTANT**
> Using New-NAVWebServerInstance cmdlet requires that Microsoft .NET Core Windows Server Hosting is installed and IIS is installed and configured with the prerequisites. So unless you have previously installed the Business Central Web Server components by using Setup, you will have to install and configure the prerequisites manually. For more information about the prerequisites, see Configure Internet Information Services.

For information about how to create a Business Central Web Server instance by using the New-NAVWebServerInstance cmdlet, see Creating and Managing Business Central Web Server Instances Using PowerShell.

Deployment Phases

Typically, you will deploy the Business Central Web client in phases, which can influence the network topology and security settings that you deploy. For example, in the development phase, you develop, test, and fine-tune the application. In this phase, you might consider deploying the Business Central Web client in a single-computer scenario. When you move to the production phase, you deploy the Business Central Web client in the full network infrastructure.

Security

User Authentication

Business Central supports four methods for authenticating users who try to access the Business Central Web client: Windows, UserName, NavUserPassword, and AccessControlService. Windows authentication is configured by default. For more information, see Users and Credential Types and Authentication and User Credential Type.

Service Account for Business Central Server and Business Central Database Access

When you install Business Central Server and Business Central database components, you must identify an Active Directory account to provide credentials for the servers. By default, Setup runs Business Central Server and the Business Central database under the Network Service account, a predefined local account that is used by the service control manager.

> **TIP**
> We recommend that you create and use a domain user account for running Business Central Server and accessing the Business Central database. The Network Service account is considered less secure because it is a shared account that can be used by other unrelated network services.

For more information, see Provisioning a Service Account.

Securing the Connection to Microsoft Dynamics NAV Web Client With SSL

You can help secure Business Central data that is transmitted over the Internet by enabling Secure Sockets Layer (SSL) on the connection to the Business Central Web client. You can configure SSL when you install the Business Central Web Server components or after the installation.

For more information, see Configure SSL to Secure the Web Client Connection.

See Also

Configure Internet Information Services
Configuring-the Business Central Web Server

Configuring Business Central Web Server Instances

2/17/2021 • 11 minutes to read • Edit Online

Accessing Business Central from the Business Central Web client or App requires a Business Central Web Server instance on IIS. You create a Business Central Web Server instance for the Business Central Web Server by using the Business Central Setup to install the Business Central Web Server or by running the New-NAVWebServerInstance cmdlet. When you set up a web server instance, you are configuring the connection from the Business Central Web Server to the Business Central Server instance. The connection settings, along with several other configuration settings, are saved in a configuration file for the web server instance.

About the configuration file

The configuration file for the web server instances is a .json file type called **navsettings.json**. The navsettings.json file is a Java Script Object Notification file type that is similar to files that use the XML file format.

After installation, you can change the configuration by modifying the navsettings.json. There are two ways to modify this file: directly or using PowerShell.

Where to find the navsettings.json file

The navsettings.json file is stored in the physical path of the web server instance, which is by default is *%systemroot%\inetpub\wwwroot\[WebServerInstanceName]*.

[WebServerInstanceName] corresponds to the name (alias) of the web server instance in IIS, for example, *c:\inetpub\wwwroot\BC160*.

Modify the navsettings.json file directly

1. Open the navsettings.json in any text or code editor, such as Notepad or Visual Studio Code.

   Each setting is defined by a key-value pair.

   - In the navsettings.json file, a setting has the format:

     ```
     "keyname": "keyvalue",
     ```

   The `keyname` is the name of the configuration setting and the `keyvalue` is the value.

   For example, the configuration setting that specifies the Windows credential type for authenticating users is:

   ```
   "ClientServicesCredentialType":  "Windows",
   ```

   Include values in double quotes.

2. Find the configuration settings that you want to change, and then change the values.

   See the Settings section for a description of each setting.

3. When you are done making changes, save the file.

4. Restart the Business Central Web Server instance for the changes to take effect.

For example, in IIS Manager, in the **Connections** pane, select website node for Business Central Web Server, and then in the **Actions** pane, choose **Restart**. Or, from your desktop, run `iisreset`.

Modify the navsettings.json file by using the Set-NAVWebServerInstanceConfiguration cmdlet

The PowerShell script module **NAVWebClientManagement.psm1** includes the Set-NAVWebServerInstanceConfiguration cmdlet that enables you to configure a web server instance.

1. Depending on your installation, run the Dynamics NAV Development Shell or Windows PowerShell as an administrator.

   For more information, see Get started with the Business Central Web Server cmdlets.

2. For each setting that you want to change, at the command prompt, run the following command:

   ```
   Set-NAVWebServerInstanceConfiguration -Server [MyComputer] -ServerInstance [ServerInstanceName] -
   WebServerInstance [MyBCWebServerInstance] -KeyName [Setting] -KeyValue [Value]
   ```

   Replace:

   - `[MyComputer]` with the name of the computer that is running the Business Central Server
   - `[ServerInstanceName]` with the name of the server instance, such as **BC160**.
   - `[MyBCWebServerInstance]` with the name of the web server instance for the Business Central Web Server.
   - `[KeyName]` with the name of the setting. Refer to the next section in this article.
   - `[KeyValue]` with the new value of the setting.

Settings in the navsettings.json

The navsettings.json has the following structure, where settings are included under one of two root elements: `NAVWebSettings` and `ApplicationIdSettings`:

```
{
  "NAVWebSettings": {
     "//ServerInstance":  "Name of the Business Central Server instance to connect to (for client) or listen
on (for server).",
     "ServerInstance":  "BC150",
      [...more keys]
 },
  "ApplicationIdSettings": {
      "BlogLink": "https://myBCBlog.com",
      [...more keys]
  }
}
```

`//` indicates a comment that provides help for the setting, and has no effect on the Web Server instance.

The following table describes the settings that are available in the navsettings.json for each root element. If you do not see a setting in the file, this is because some settings are not automatically included as a key in the file. For these settings, you can add the key manually. If you do not add the key, the default value of the setting is used.

> **IMPORTANT**
> If modifying the file directly, place values in double quotes `""`.

`NAVWebSettings` element settings

SETTING/KEYNAME	DESCRIPTION
AllowedFrameAncestors	Specifies the host name of any web sites in which the Business Central Web client or parts of are embedded. By default, the Business Central Web Server will not allow a website to display it inside an iframe unless the website is hosted on the same web server. This value of this setting is a comma-separated list of host names (URIs). Wildcard names are accepted. For example: `https:mysite.sharepoint.com, https:*.myportal.com`
AllowNtlm	Specifies whether NT LAN Manager (NTLM) fallback is permitted for authentication. To require Kerberos authentication, set this value to `false`. Values: `true`, `false` Default value: `true`
AuthenticateServer	Specifies whether to authenticate the server by enabling service identity checks on protocol between the Web server and the Business Central Server instance. Values: `true`, `false` Default value: `true`
ClientServicesCredentialType	Specifies the authorization mechanism that is used to authenticate users who try to connect to the Business Central Web Server. For more information, see Authentication and User Credential Type. The credential type must match the credential type configured for the Business Central Server instance that the Business Central Web Server connects to. For information about how to set up the credential type on Business Central Server, see Configuring Business Central Server. Values: Windows, UserName, NavUserPassword, AccessControlService Default value: Windows

SETTING/KEYNAME	DESCRIPTION
ClientServicesChunkSize	Sets the maximum size, in kilobytes, of a data chunk that is transmitted between Business Central Web Server and Business Central Server. Data that is transmitted between Business Central Web Server and Business Central Server is broken down into smaller units called chunks, and then reassembled when it reaches its destination. Values: From 4 to 80. Default value: 28
ClientServicesCompressionThreshold	Sets the threshold in memory consumption at which Business Central Web Server starts compressing data sets. This limits amount of consumed memory. The value is in kilobytes. Default value: 64
ClientServicesPort	Specifies the TCP port for the Business Central Server. This is part of the Business Central Server's URL. Values: 1-65535 Default value: 7046
ClientServicesProtectionLevel	Specifies the security services used to protect the data stream between the Business Central Web Server and Business Central Server. This value must match the value that is specified in the Business Central Server configuration file. For more information, see Configuring Business Central Server. Values: EncryptAndSign, Sign, None Default value: EncryptAndSign
DefaultRelativeHelpPath	Specifies the default Help article to open if no other context-sensitive link is specified. Default value: none
Designer	Specifies whether the in-client designer is enabled in the Business Central Web client. Set to `true` to enable designer. For more information, see Using Designer.

SETTING/KEYNAME	DESCRIPTION
DnsIdentity	Specifies the subject name or common name of the service certificate for Business Central Server. This parameter is only relevant when the ClientServicesCredentialType is set to UserName, NavUserPassword, or AccessControlService, which requires that security certificates are used on the Business Central Web Server and Business Central Server to protect communication. **Note:** You can find the subject name by opening the certificate in the Certificates snap-in for Microsoft Management Console (MMC) on the computer that is running Business Central Web Server or Business Central Server. For more information, see Authentication and User Credential Type. Value: The subject name of the certificate. Default value: none
GlobalEndPoints	Specifies the comma-separated list of global endpoints that are allowed to call this website. The values must include http scheme and fully qualified domain name (FDQN), such as `https://businesscentral.microsoft.com`. "GlobalEndPoints": "null,ms://businesscentral,ms://dynamicsnav" Default value: none
HelpServer	Specifies the name of the Business Central Help Server if the deployment uses Help Server. If the deployment uses an online library, remove this setting. Default value: none
HelpServerPort	Specifies the TCP port on the specified Business Central Help Server if the deployment uses Help Server. If the deployment uses an online library, remove this setting. Default value: none
LoadScriptsFromCdn	Specifies whether to load scripts from Content Distribution Networks (CDNs). This only applies to scripts that are available from a CDN, like jQuery. If set to `false`, scripts will be loaded from the Web server, which is useful in, for example, an intranet scenario where there is no internet access. Default value: `true` **DISCONTINUED AFTER:** Business Central Spring 2019
ManagementServicesPort	The listening TCP port for the Business Central management endpoint. Valid range: 1-65535 Default value: 7045

SETTING/KEYNAME	DESCRIPTION
OfficeSuiteShellServiceClientTimeout	Defines the time Business Central will wait for the Office Suite Shell Service to respond. **Important:** This setting has been deprecated in Business Central, and it has no effect on the Web Server instance. Default value: 10
PersonalizationEnabled	Specifies whether personalization is enabled in the Business Central Web client. Set to `true` to enable personalization. For more information, see Managing Personalization.
ProductName	Specifies the full name of the application.
ProductNameShort	Specifies the short name of the application.
ProductNameMarketing	Specifies the marketing name of the application.
RequireSsl	Specifies whether SSL (https) is required. If the value is set to `true`, all cookies will be marked with a `\u0027secure\u0027` attribute. If SSl is enabled on the Web server, you should set this to `true`. Values: `true`, `false` Default value: `false`
Server	Specifies the name of the computer that is running the Business Central Server. Default value: localhost
ServerInstance	Specifies the name of the Business Central Server instance that the Business Central Web Server connects to. Default value: BC160
ServicePrincipalNameRequired	If this parameter is set to `true`, then the Business Central Web Server can only connect to a Business Central Server instance that has been associated with a service principal name (SPN). If this parameter is set to `false`, then the Business Central Web Server attempts to connect to the configured Business Central Server service, regardless of whether that service is associated with an SPN. For more information about SPNs, see Configure Delegation. Default: `false`

SETTING/KEYNAME	DESCRIPTION
SessionTimeout	Specifies the maximum time that a connection between the Business Central Web Server and the Business Central Server can remain idle before the session is stopped. In the Business Central Web Server, this setting determines how long an open Business Central page or report can remain inactive before it closes. For example, when the SessionTimeout is set to 20 minutes, if a user does not take any action on a page within 20 minutes, then the page closes and it is replaced with the following message: **The page has expired and the content cannot be displayed.** The time span has the format [dd.]hh:mm:ss[.ff]: - dd is the number of days - hh is the number of hours - mm is the number of minutes - ss is the number of seconds - ff is fractions of a second Default value: 00:20:00
ShowPageSearch	Specifies whether to show the  **Tell me what you want to do** icon in the Business Central header. This feature lets users find Business Central objects, such as pages, reports, and actions. If you do not want to show the **Tell me what you want to do** icon, then set the parameter to `false`. Default value: `true`

SETTING/KEYNAME	DESCRIPTION
UnknownSpnHint	Specifies whether to use a server principal name when establishing the connection between the Business Central Web Server server and Business Central Server. This setting is used to authenticate the Business Central Server, and it prevents the Business Central Web Server server from restarting when it connects to Business Central Server for the first time. You set values that are based on the value of the ServicePrincipalNameRequired key. Value: The value has the following format. (net.tcp://BCServer:Port/ServerInstance/Service)=NoSpn|SPN - BCServer is the name of the computer that is running the Business Central Server. - Port is the port number on which the Business Central Server is running. - ServerInstance is the name of the Business Central Server instance. - NoSpn|SPN specifies whether to use an SPN. If the ServicePrincipalNameRequired key is set to `false`, then set this value to NoSpn. If the ServicePrincipalNameRequired key is set to `true`, then set this value to Spn. Default value: (net.tcp://localhost:7046/BC160/Service)=NoSpn If you set this key to the incorrect value, then during startup, the Business Central Web Server will automatically determine a correct value. This will cause the Business Central Web Server to restart. **Note:** For most installations, you do not have to change this value. Unlike the Dynamics NAV Client connected to Business Central, this setting is not updated automatically. If you want to change the default value, then you must change it manually.
UseAdditionalSearchTerms	Specifies whether **Tell me** uses the additional search terms that are defined on pages and reports. The additional search terms are specified by the AdditionalSearchTerms and AdditionalSearchTermsML properties. If you set this to `false`, the additional search terms are ignored. Default value: true

`ApplicationIdSettings` element settings

SETTING/KEYNAME	DESCRIPTION
BaseHelpUrl	Specifies the link to the online Help library that the deployment uses, such as *https://mysite.com/{0}/mylibrary/*. Default value: none For more information, see Configuring the Help Experience.

SETTING/KEYNAME	DESCRIPTION
BlogLink	Specifies the target of the **blog** link on the **Help & Support** page. Use this link to give users access to your end-user blog. Value: a valid URL Default value: https://go.microsoft.com/fwlink/?linkid=2076643|
ComingSoonLink	Specifies the target of **coming soon** link on the **Help & Support** page. Use this link to give users access to information about your roadmap or any upcoming features and fixes. Value: A valid URL. Default value: https://go.microsoft.com/fwlink/?linkid=2047422
CommunityLink	Specifies the URL to a community or resource for sharing information. Value: a valid URL Default value: https://go.microsoft.com/fwlink/?LinkId=287089|
ContactSalesLink	Specifies the target of the contact sales link on the **Help & Support** page. Use this link to give users access to your sales-focused web page where users can engage with your sales process. **Note:** This setting and link are not used for Business Central on-premises.
SigninHelpLink	Specifies the URL to open if the user selects help on the sign-in page box. Value: a valid URL Default value: none

See Also

Setting up Multiple Business Central Web Server Instances
Install Business Central Components
Business Central Web Server Overview
Configuring Business Central Server
Configuring the Help Experience

Configuring Internet Information Services for Business Central

2/17/2021 • 2 minutes to read • Edit Online

This topic describes the configuration of Internet Information Service (IIS) that is required for running Business Central. When you install the Business Central Web Server components, you can use the Business Central Setup to install and configure the IIS features, so you do not have to do this manually.

Required IIS Features

IIS must have the following features enabled:

- HTTP Activation
- NET Extensibility 4.5, .NET Extensibility 4.6, or .NET Extensibility 4.6 (depending on Windows version)
- ASP.NET 4.5, ASP.NET 4.6, or ASP.NET 4.7 (depending on Windows version)
- ISAPI Extensions
- ISAPI Filters
- Request Filtering
- Windows Authentication
- Default Document
- Directory Browsing
- HTTP Errors
- Static Content

Configure Headers in Application Request Routing (ARR) Rules

If you are hosting the Business Central Web Server components on an IIS server farm that is using Application Request Routing (ARR), you must configure headers. Business Central Web Server runs on ASP .NET Core, which requires both an `X-Forwarded-For` header and `X-Forwarded-Proto` header in ARR routing rules. However, by default, ARR only adds the `X-Forwarded-For` header; not the `X-Forwarded-Proto` header. So will have to configure the `X-Forwarded-Proto` header manually.

On the server farm in IIS, add or edit a routing rule to include a server variable for `X-Forwarded-Proto`. For example, using IIS Manager, select **Routing Rules** > **URL Rewrite** > **Edit** > **Server Variables**, and then add a server variable that has the following settings:

NAME	VALUE	REPLACE
`HTTP_X_FORWARED_PROTO`	`http` or `https`	`true`

See Also

Business Central Web Server Overview

Configuring SSL to Secure the Business Central Web Client Connection

2/17/2021 • 3 minutes to read • Edit Online

We recommend that you secure Business Central data that is transmitted over the Internet by enabling Secure Sockets Layer (SSL) on the connection to Business Central Web client.

SSL is a web protocol that encrypts data that is transmitted over a network to make the data and the network more secure and reliable. A website that is enabled with SSL uses Hypertext Transfer Protocol Secure (HTTPS) instead of Hypertext Transfer Protocol (HTTP) as a communication protocol. Enabling SSL on a website requires that an SSL certificate is installed on the web server. An SSL certificate is a small file that the web server uses to prove its identity and establish a trusted connection with the browser that is trying to access Business Central Web client. When a browser connects to the Business Central Web client, the web server replies by sending its certificate to the browser. This certificate contains the web server's public encryption key and the name of the authority that granted the certificate. The browser verifies the certificate using the authority's public key.

To configure SSL, you must follow the steps in this article.

> **NOTE**
> You can configure SSL when you install the Business Central Web Server components using Business Central Setup .

Obtaining and Installing an SSL Certificate

In a production environment, you should obtain an SSL certificate from a certification authority. Some large organizations may have their own certification authorities, and other organizations can request a certificate from a third-party organization. In a test environment or development environment, you can create your own self-signed certificate.

Adding an HTTPS Binding That Uses the Certificate on the Dynamics 365 Business Central Website

After you get the certificate, you add a binding to the https protocol on the website. When you add the binding, you associate it with the certificate.

Add an https binding with the certificate to the website

1. Open Internet Information Services (IIS) Manager.
2. In the **Connections** pane, expand the **Sites** node, and then choose the Business Central Web client site to which you want to add the binding.

   By default, the site has the name **Dynamics 365 Business Central Web Client**.
3. In the **Actions** pane, choose **Bindings**.
4. In the **Site Bindings** dialog box, choose **Add**.
5. In the **Add Site Binding** dialog box, set the **Type** field to **https**.

   You can use the default port 443 or change it to another port. If you change it to another port, you will have to provide the port number in the URL when you try to open the client.

6. Set the **SSL certificate** field to the certificate that you obtained or created for the site.
7. Choose the **OK** button, and then choose the **Close** button.

Redirecting HTTP to HTTPS (Optional)

To ensure that users always access the site that is secured with SSL, you can automatically redirect HTTP requests to HTTPS. This means that users do not have to explicitly include https in the URL in the browser. For example, the nonsecure URL of the Business Central Web client could be `https://MyWebclient:8080/BC150` and the secure URL could be `https://MyWebclient:443/BC150`. If a user types `https://MyWebclient:8080/BC150`, the browser automatically redirects to `https://MyWebclient:443/BC150`.

There are different ways to redirect HTTP requests to HTTPS. The following procedure describes how to redirect HTTP requests to HTTPS by installing the Microsoft Application Request Routing for IIS 8 and modifying the configuration file for the Business Central Web Server instance.

Redirect HTTP to HTTPS

1. Download and install Microsoft Application Request Routing for IIS. For example, you can download from Microsoft Application Request Routing.
2. On the computer that is running Business Central Web Server components, open the web.config file for the Business Central Web Server instance. Use a text editor, such as Notepad.

   The web.config file is located in the physical path of the web application on IIS. By default, the path is %systemroot%\inetpub\wwwroot\[VirtualDirectoryName]. For example, the folder for the default application is %systemroot%\inetpub\wwwroot\BC160.
3. In the `<system.webServer>` element, add the following elements.

```
<rewrite>
  <rules>
    <rule name="Redirect to HTTPS">
      <match url="(.*)" />
      <conditions>
        <add input="{HTTPS}" pattern="off" ignoreCase="true" />
      </conditions>
      <action type="Redirect" url="https://{SERVER_NAME}/{R:1}" redirectType="SeeOther" />
    </rule>
  </rules>
</rewrite>
```

4. Save the navsettings.json file.

See Also

Business Central Web Server Overview

Setting Up Multiple Business Central Web Server Instances Using PowerShell

2/17/2021 • 5 minutes to read • Edit Online

Although you can use the Business Central Setup to install the Business Central Web Server components and create a single web server instance in IIS for client connection, there may be scenarios when you want to set up multiple instances. For example, you could set up a separate Business Central Web Server instance for the different companies of a business. For this scenario, you can use the Business Central Web Server PowerShell cmdlets, which are outlined in the following table.

CMDLET	DESCRIPTION
New-NAVWebServerInstance	Creates a new Business Central Web Server instance and binds this instance to a Business Central Server instance.
Set-NAVWebServerInstanceConfiguration	Specifies configuration values for a named Business Central Web Server instance.
Get-NAVWebServerInstance	Gets the information about the Business Central Web Server instances that are registered on a computer.
Remove-NAVWebServerInstance	Removes an existing Business Central Web Server instance.

Get started with the Business Central Web Server cmdlets

The Business Central Web Server cmdlets are contained in the PowerShell script module **NAVWebClientManagement.psm1**, which is available on the Business Central installation media (DVD).

The module is installed with the Business Central Server or the Business Central Web Server components.

There are different ways to launch this module and start using the cmdlets:

- If you are working on the computer where the Business Central Server was installed, run the Business Central Administration Shell as an administrator.

  For more information, see Business Central PowerShell Cmdlets.

- If you installed the Business Central Web Server components, just start Windows PowerShell as an administrator.

- Otherwise, start Windows PowerShell as an administrator, and use the Import-Module cmdlet to import the **NAVWebClientManagement.psm1** file:

  ```
  Import-Module -Name [filepath]
  ```

  For example:

  ```
  Import-Module -Name "C:\Program Files\Microsoft Dynamics 365 Business
  Central\130\Service\NAVWebClientManagement.psm1"
  ```

For more information, see the Windows PowerShell docs.

Creating Business Central Web Server instances

Get Access to the WebPublish folder

To create a new web server instance, you need access to the **WebPublish** folder that contains the content files for the Business Central Web Server components.

- This folder is available on the Business Central installation media (DVD) and has the path "DVD\WebClient\Microsoft Dynamics NAV\13x\Web Client\WebPublish".
- If you installed the Business Central Web Server components, this folder has the path "C:\Program Files\Microsoft Dynamics 365 Business Central\160\Web Client\WebPublish".

You can use either of these locations or you can copy the folder to more convenient location on your computer or network.

Decide on the site deployment type for the instance

When you create a new Business Central Web Server instance, you can choose to create either a RootSite or SubSite type. Each instance type has a different hierarchical structure in IIS, which influences its configuration and the URLs for the accessing from the Business Central Web client.

RootSite

A *RootSite* instance is a root-level web site that is complete with content files for serving the Business Central Web client. It is configured with its own set of bindings for accessing the site, such as protocol (http or https) and communication port. The structure in IIS looks like this:

```
- Sites
  - BusunessCentralWebSite (web site)
    + nn-NN (language versions)
    + www (content)
    navsettings.json
    ...
```

The Business Central Web client URL for the RootSite instance has the format:

`https://[WebserverComputerName]:[port]`

For example: `https://localhost:8080`.

SubSite

A *SubSite* instance is a web application that is under a container web site. The container web site is configured with a set of bindings, but the site itself has no content files. The content files are contained in the application (SubSite). The SubSite inherits the bindings from the container web site. This is the deployment type that is created when you install Business Central Web Server components in the Setup wizard. Using the New-NAVWebServerInstance cmdlet, you can add multiple SubSite instances in the container web site. The structure in IIS for two instances looks like this in IIS:

```
- Sites
  - BusunessCentralWebSite (web site)
    - BusinessCentralWebInstance1 (application)
      + nn-NN (language versions)
      + www
      navsettings.json
      ...
    - BusinessCentralWebInstance2 (application)
      + nn-NN (language versions)
      + www
      navsettings.json
      ...
```

The Business Central Web client URL of a SubSite instance is generally longer than a RootSite because it also contains the application's alias (or virtual path) for the instance, which you define. The URL for a SubSite instance has the format:

`https://[WebserverComputerName]:[port]/[WebServerInstance]`

For example: `https://localhost:8080/BusinessCentralWebInstance1` and `https://localhost:8080/BusinessCentralWebInstance2`.

Run the New-NAVWebServerInstance cmdlet

At the command prompt, run the New-NAVWebServerInstance cmdlet. The following are simple examples for creating a RootSite and SubSite deployment type.

RootSite example:

```
New-NAVWebServerInstance -WebServerInstance MyBCWebsite -Server MyBCServer -ServerInstance
MyBCServerInstance -SiteDeploymentType RootSite -WebSitePort 8081 -PublishFolder "C:\Web Client\WebPublish"
```

SubSite example:

```
New-NAVWebServerInstance -WebServerInstance MyWebApp -Server MyBCServer -ServerInstance MyBCServerInstance -
SiteDeploymentType Subsite -ContainerSiteName MySiteContainer -WebSitePort 8081 -PublishFolder
"C:\WebClient\WebPublish"
```

- Susbtitute *MyBCWebsite* with the name that you want to give the web application in IIS for the web server instance. If you are creating a SubSite deployment type, this name will become part of the URL for opening the Business Central Web client application, for example, https://MyWebServer:8081/MyWebApp.
- Susbtitute *MyBCServer* to the name of the computer that is running the Business Central Server to which you want to connect.
- Susbtitute *MyBCServerInstance* with the name of the Business Central Server instance to use.
- Substitute *MySiteContainer* with name of the container web site under which you want to add the instance. If you specify a name that does not exist, then a new container web site will be created, which contains the new web server instance.
- Susbtitute *8081* with the port number that you want to bind the instance to. If you do not specify a port number, then port 80 is used.
- Substitute *C:\WebClient\WebPublish* with the path to your WebPublish folder. By default, the cmdlet looks in the'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Web Client' folder. So if you are working on a computer where the Business Central Web Server components are installed, you do not

have to set this parameter.

> **NOTE**
>
> This command only sets the required parameters of the NAVWebServerInstance cmdlet. The cmdlet has several other parameters that can use to configure the web server instance. For more information about the syntax and parameters, see New-NAVWebServerInstance.

Modifying a Business Central Web Server instance

After you create the web server instance, if you want to change its configuration, you can use the Set-NAVWebServerInstanceConfiguration cmdlet. Or, you can modify the configuration file (navsettings.json) of the instance directly. For more information, see Configuring Web Server Instances.

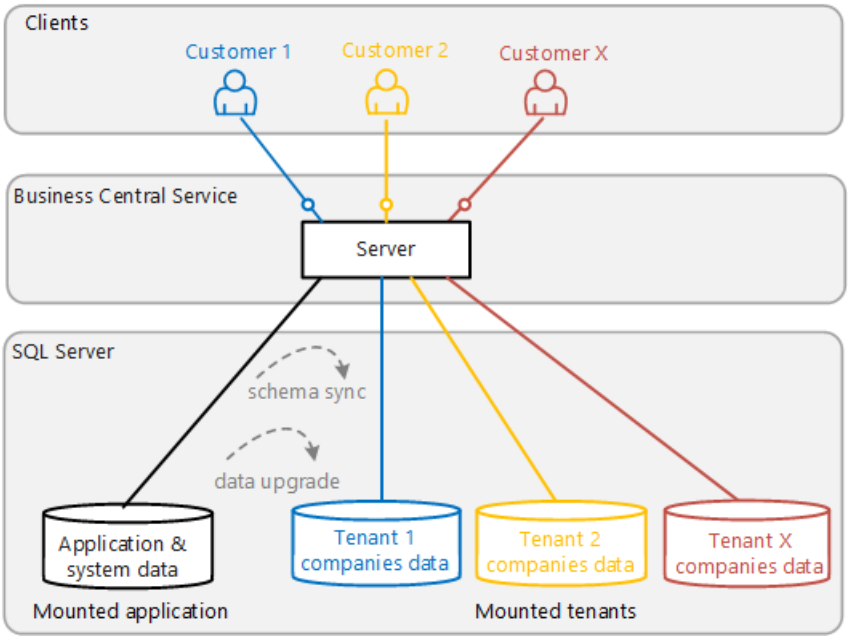
See Also

Business Central Web Server Overview
Configuring Web Server Instances

Multitenant Deployment Architecture in Business Central

2/17/2021 • 6 minutes to read • Edit Online

Business Central supports deployments where several different companies access a centrally maintained Business Central application. By using this *multitenancy* support, you can add new customers to your solution easily, and you can roll out updates quickly with limited downtime for your customers.



NOTE

You don't have to turn your Business Central solution into a multitenant deployment. You can install and run Business Central as a classic one-server-one-database deployment.

In a multitenant deployment, information about the Business Central application is stored in a separate application database. Your customers' data is stored in separate business databases, each of which is a *tenant* in your deployment. By separating application from data, you can deploy the same solution to many customers with centralized maintenance of the application and isolation of each tenant. The application database contains the tables that define an application, such as the **Object** table and other system tables.

For example, if your current solution contains 10 companies in the Business Central database, you can choose to create separate Business Central databases to store each company's business data. The knowledge about the shared application is then stored in a dedicated application database. Business Central includes Windows PowerShell cmdlets that create an application database, and other cmdlets that enable you to create and administer tenant-specific databases.

You can choose to upgrade to Business Central and not change your deployment so that you still have a single database that has one or more companies in it. You can also choose to extract the application tables to an application database but still have one business data database that has one or more companies in it. In both scenarios you have not migrated to multitenancy, but in the second scenario you have prepared your solution so that you can move to multitenancy at a later point.

Tenants, Companies, and Databases

A tenant is an entity that uses your solution and stores data in a business database. This is often either a business or a group of legal entities whose data can be stored in one database. In practical terms, a tenant is a database that stores business data for one or more Business Central companies. Each tenant is connected to a Business Central Server instance, but the Business Central Server instance can support multiple tenants.

When you deploy and maintain a Business Central solution, you must activate the relationship between the Business Central Server instance by mounting the tenant to the Business Central Server instance. You can do this by using the Business Central Server Administration tool or by running the **Mount-NAVTenant** and **Sync-NAVTenant** cmdlets from the Business Central Administration Shell. Similarly, to disconnect a tenant, you can use the Business Central Server Administration tool or run the **Dismount-NAVTenant** cmdlet. For more information, see How to: Mount or Dismount a Tenant on a Microsoft Dynamics Server Instance.

When tenants are mounted, the tenant configurations are stored in the **dbo.ndotenants** table of the application database that is connected to the Business Central Server instance. If you connect additional Business Central Server instances to the same application database, the added server instances will automatically inherit the tenant configurations from the application database. This means that existing tenants will be automatically mounted to the new server instance. In addition, if you must mount or dismount a tenant, you only have to perform the operation on one of the Business Central Server instances. The other server instances will automatically detect and update to the changes.

When you refer to a tenant, you refer to it by the tenant ID. The first tenant that is mounted against a Business Central Server instance has the tenant ID **default**. However, you can choose to set up host names for the tenants in your deployment. For example, if you want a tenant to access Business Central through a URL, you can set up a tenant-specific subdomain. The users in that tenant will then access Business Central through a URL such as *https://mytenant.myservice.com*. The tenant host name, *mytenant.myservice.com*, must be specified as an alternative ID in the tenant configuration. You can specify alternative IDs for a tenant by using the **Mount-NAVTenant** Windows PowerShell cmdlet.

Companies

A tenant database can contain one or more Business Central companies. It's not the number of companies in a database that determines whether you are running a multitenant environment. The deciding factor is whether you have created an application database, and if you have more than one tenant database connected to the application database.

Databases

When information about the application is stored in a separate application database, you maintain the application centrally without affecting the various tenants that use the application. Each tenant database contains the business data for one or more specific companies and doesn't contain all of the application metadata.

For example, if you want to modify a report, and your solution is used by 25 customers, you modify the report in the application database. When each customer then accesses the report, they see the modified report.

Deployment Scenarios Supported in Business Central

The following table compares deployment scenarios.

INCLUDES APPLICATION DATABASE	NO. OF BUSINESS DATABASES PER APPLICATION DATABASE	NO. OF COMPANIES IN BUSINESS DATABASE	MULTITENANT DEPLOYMENT
No	1	1	No
Yes	1	1	No

INCLUDES APPLICATION DATABASE	NO. OF BUSINESS DATABASES PER APPLICATION DATABASE	NO. OF COMPANIES IN BUSINESS DATABASE	MULTITENANT DEPLOYMENT
Yes	1	2	No
Yes	2	1	Yes
Yes	2	2	Yes

In the table, the number of companies and business databases are shown as either 1 or 2. But most of the time there are either 1 or more than 2.

The table describes different deployments of a Business Central solution. For example, a deployment with one database and a single company versus a deployment with two or more business databases for each application database. Of those two scenarios, only the second is a multitenant deployment because it connects multiple tenant databases (the business databases) with a single application database. The table also illustrates that you can have multiple companies in a business database. Finally, you can have an application database and a single business database that contains multiple companies. This is a single-tenant deployment.

The Application in Multitenant Deployments

In a Business Central application that is used in a multitenant deployment, some areas require you to set up web services. Since web services are created in the application database, you must create at least one tenant that has write access to the application database. This setting is determined by the *Allow application database writes* parameter when you mount a tenant against a Business Central Server instance. For more information, see How to: Mount or Dismount a Tenant on a Microsoft Dynamics Server Instance.

For example, you can create a dedicated administration tenant that you mount against the Business Central Server instance when you create web services for an application.

If you have an existing Business Central application that you want to use in a multitenant deployment, there are a number of changes that you have to make. This includes setting up the permission sets in a way that supports all tenants that use that application.

URLs and Tenants

In multitenant deployments, URLs must specify the tenant that the URL applies to. If you have C/AL code that constructs URLs, you must update the code to include the tenant. Alternatively, update your code with the GETURL Function to get the URLs calculated for you. The same applies to hyperlinks in report objects, for example.

The URL can specify the tenant ID or the tenant host name if you specify host names as alternative IDs for tenants. For example, the following URL consumes the **Customer** ODATA web service for a specific tenant:

```
https://localhost:7048/BC/OData/Company('CRONUS-International-Ltd.')/Customer?Tenant=Tenant1
```

If the *mytenant.myservice.com* host name has been specified as an alternative ID for the tenant Tenant1, then the following URL returns the same ODATA web service:

```
https://mytenant.myservice.com:7048/BC/OData/Company('CRONUS-International-Ltd.')/Customer
```

See Also

Migrating to Multitenancy

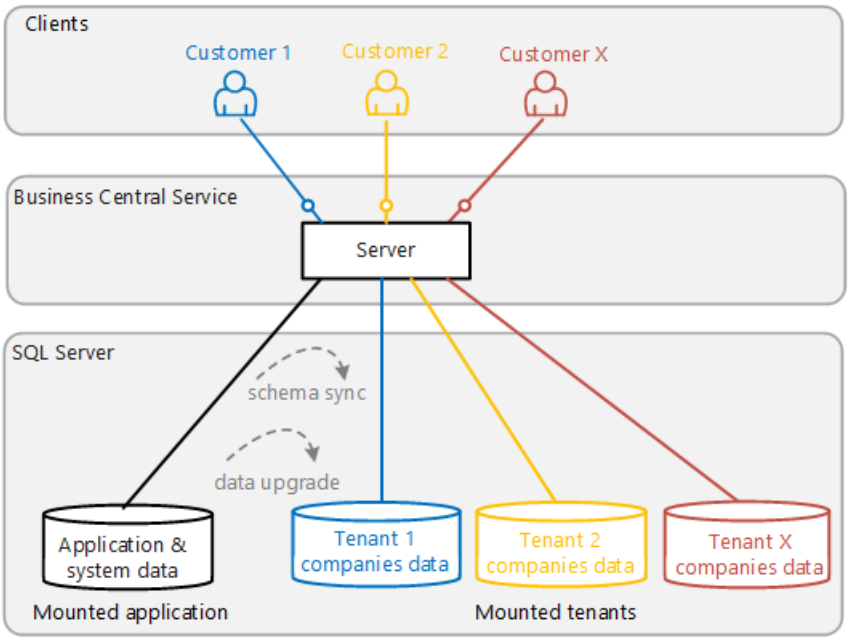
Microsoft Dynamics 365 Windows PowerShell Cmdlets

How to: Mount or Dismount a Tenant on a Microsoft Dynamics Server Instance

Multitenant Deployment Setup Guide

2/17/2021 • 2 minutes to read • Edit Online

This article provides an introduction to the most common tasks you'll do to set up and maintain a Business Central Server multitenant deployment.



Create an application database

The application database includes tables that store information about the application and system, required by all tenants. There are two ways to create an application database:

- Start from scratch and use the New-NAVApplicationDatabase cmdlet.

  This cmdlet is available in the Business Central Administration Shell. It creates a database in SQL Server that's initialized with the required tables and data. For more information, see Create an Application Database.

- Export the application objects from an existing database to a new database.

  If you already have a database that includes the application, you can use the Export-NAVApplication cmdlet to export only the application objects to a new database. For more information, see Exporting the Application Tables to a Dedicated Application Database.

Configure the Business Central Server for multitenancy

To set up the Business Central Server instance for multitenancy, make the following changes to the server instance configuration:

1. Enable the **Multitenant** setting.
2. Set the server instance to connect to the application database by changing these settings: **DatabaseServer**, **DatabaseServer**, and **DatabaseServer**.

For more information, see Configuring Business Central Server and Connecting a Business Central Server Instance to a Database.

TIP

Instead of doing the two previous steps, you could use the Mount-NAVApplication cmdlet. Running this cmdlet will enable the **Multitenant** and set up the data base connection.

Publish the symbols and extensions

Before mounting tenants, publish the system symbols and extensions that make up your application. Publish the system application and base application extensions as a minimum. Then, as needed, publish any Microsoft and third-party extensions tenants may use.

For more information, see Publishing and Installing an Extension.

Create, mount, and synchronize tenants

The tenant database includes tables for storing customer data that's accessible from the application. The tenant database is often referred to as just the *tenant*. The basics tasks for getting a new tenant up and running, and keeping it up to date, are as follows:

1. Create an empty database with the wanted collation in SQL Server.
2. Mount the database as a tenant of the Business Central Server instance.

   This operation connects the tenant database to the Business Central Server so it can process data requests from the application. See Mount or Dismount a Tenant.
3. Synchronize the tenant database with the application.

   The application database stores information that defines the basic structure or schema required for all tenant databases. This operation updates the tenant database schema with the definitions in application database. See Synchronizing the Tenant.
4. Synchronize and install extensions.

   The last step is to install all the extensions that you want to use on the tenant. Many extensions define table objects and table extension objects for storing data. So, before you can install an extension, you have to synchronize it with the tenant to update the tenant database's schema. See Publishing and Installing an Extension.

TIP

For more detailed steps, see Create a Tenant Database.

See Also

Multitenant Deployment Overview Architecture
Multitenant Deployment Architecture

Migrating to Multitenancy

2/17/2021 • 2 minutes to read • Edit Online

You can choose to migrate your Business Central solution to a multitenant deployment architecture where you maintain a single application that is used by two or more companies that store their data in separate databases.

This can make maintenance of your solution easier if you support multiple customers with the same application functionality.

Tenants and Companies

When you upgrade your application and the data to Business Central, you have a database that has the same number of companies as you had before the upgrade. This database is considered a *tenant*. This does not mean that you have to turn your solution into a multitenant deployment. But it means that you can if you want to.

For example, your Business Central deployment in the earlier version consisted of a database that has 20 companies. In other words, you support 20 companies that all share the same application functionality. In this example, the companies are separate companies that have nothing to do with each other except that they are supported by you in one database. In Business Central, you can choose to extract the application-wide tables into a separate database and keep the data for all 20 companies in the original database. This becomes a single-tenant business data database. Then, you can choose to split the business data database into one for each company so that you run a truly multitenant environment. The application is stored separately in the application database, and you maintain application functionality centrally. When you modify the application, you make the changes available to one tenant at a time. As a result, if something goes wrong, all other tenants are not affected.

Compare this to earlier versions of Business Central where a database could contain several companies. These companies could be related or not, but they would all use the same application and write to the same database. Also, when you modified the application, it would affect all companies immediately. So if something went wrong, all companies would be affected.

> **NOTE**
> The email logging functionality in Business Central requires the Business Central Server service account to have access to the Exchange server. But in a multitenant deployment, this is not always possible.

In multitenant deployments of Business Central, permission sets are stored centrally in the application database, so only the administrator of the central application can add, remove, or modify permission sets. Instead, the tenants can use user groups to manage permissions.

Migration Process

If you decide to move to a multitenant architecture, you must complete the following steps:

1. Move the tables that describe the application to a separate database. For more information, see Separating Application Data from Business Data.

   After this step, you have two databases: an application database and a business data database.

2. Split the business data database into one for each company. For more information, see Creating Tenants from Companies.

   After this step, you have an application database and a business data database for each company in the

original database. The company-specific business data databases are tenants, and your solution is multitenant.

If you want to move back to storing application tables and business data in a single database, you can use the Business Central Windows PowerShell cmdlets to merge the databases. For more information, see Merging an Application Database with a Tenant Database.

See Also

Separating Application Data from Business Data
Creating Tenants from Companies
Upgrading the Application Code
Upgrading the Data
Upgrading to Business Central

Separating Application Data from Business Data

2/17/2021 • 8 minutes to read • Edit Online

Business Central separates tables that describe the application from the tables that contain business data. Depending on your deployment scenario, you can choose to store all Business Central tables in one database, or you can export the application tables to a dedicated database. In multitenant deployments, the application must be stored in a dedicated database.

Application Database versus Business Data Databases

The application database contains tables that describe your application. This includes a description of the objects that your application consists of, and other data that is common to all tenants. The data that users enter in your application is stored in the business data database because this data is specific to their company. Optionally, you can create multiple business data databases, such as if you want to support your customers as tenants.

When you have exported the application tables to a separate database, you can no longer access the business database from the Dynamics NAV Development Environment. This is because the metadata for the tables in the business database is stored in the application database and modified in that database.

For example, if you want to modify a report, you modify the report object in the application database. Then, when you deploy the updated application to your production environment, when a user accesses the report, they see the modified report.

Business Central includes Windows PowerShell cmdlets that help you export application tables to a dedicated database, and other cmdlets to help you maintain a multitenant deployment. For more information, see Business Central Windows PowerShell Cmdlets.

Distribution of the System Tables in Each Database

The application tables are system tables that define the application. Other system tables remain in the business data database. For example, the following table lists some of the system tables that are moved to the application database when you run the Export-NAVApplication cmdlet and other tables that remain in the business data database. This is a partial list to give an idea of how tables are distributed:

APPLICATION DATABASE	BUSINESS DATA DATABASE
Add-in	Access Control
Application Dependency	Active Session
Application Object Metadata	API Webhook Notification
Application Resource	API Webhook Notification Aggr
Chart	API Webhook Subscription
Configuration Package File	Company
Debugger Breakpoint	Device
Debugger Watch	Data Sensitivity

APPLICATION DATABASE	BUSINESS DATA DATABASE
Entitlement	Document Service
Entitlement Set	Intelligent Cloud
Installed Application	Intelligent Cloud Status
Media	Isolated Storage
Media Resources	Object Options
Media Set	Object Metadata Snapshot
Membership Entitlement	Object Translation
Object	Integration Page
Object Metadata	Object Metadata Snapshot
Object Tracking	Object Translation
OData Edm Type	Object Access Intent Override
Page Documentation	Page Data Personalization
Power BI Blob	Printer Selection
Power BI Selection	Record Link
Permission	Report List Translation
Permission Set	Session Event
Profile	Scheduled Task
Profile Metadata	Tenant Media
Profile Page Metadata	Tenant Permission
Published Application	Tenant Profile
Report Layout	Tenant Profile Extension
Send-To Program	Tenant Web Service
Server Instance	User
Style Sheet	User Default Style Sheet
Upgrade Blob Storage	Web Hook Notification

APPLICATION DATABASE	BUSINESS DATA DATABASE
Web Service	Web Hook Subscription

Exporting the Application Tables to a Dedicated Application Database

To export the application tables from an existing database to another database, Business Central provides a Windows PowerShell cmdlet as part of the Business Central Administration Shell.

The following procedure illustrates how you can separate the application tables in an existing database into two databases: an application database and a business data database. You can automate this process and combine it with the use of other cmdlets. For more information, see the samples in the Windows PowerShell scripts in the **...\WindowsPowerShellScripts\Multitenancy** folder on the Business Central product media.

Export the application tables to a dedicated database

1. Stop all Business Central Server services that access the database that you are modifying. This includes the Business Central Server instance.
2. Back up your databases.
3. Run the Business Central Administration Shell as an administrator.

   > **IMPORTANT**
   > You must run the program as administrator. Also, you must ensure that scripting is enabled on the computer.

   For more information, see Business Central Windows PowerShell Cmdlets.
4. Uninstall the extensions.

   To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

   ```
   Uninstall-NAVApp -ServerInstance <server instance name> -Tenant default -Name <extensions name> -
   Version <extension version>
   ```

   Replace `<extension name>` and `<extension version>` with the exact name and version the installed extension. Set `<tenant ID>` to `default` or omit the `-Tenant` parameter.

   For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance name> -Tenant default | % { Uninstall-NAVApp -
   ServerInstance <server instance name> -Tenant default -Name $_.Name -Version $_.Version -Force}
   ```
5. To export the application tables, type the following command:

   ```
   Export-NAVApplication –DatabaseServer <server name> -DatabaseInstance <instance name> –DatabaseName
   <name of original database> –DestinationDatabaseName <name of new application database> -
   ServiceAccount <the account used by server instance if not NT AUTHORITY\NETWORK SERVICE>
   ```

   For example, to run the cmdlet against the CRONUS International Ltd. demonstration database, type the following command:

```
Export-NAVApplication –DatabaseServer 'MyServer' –DatabaseInstance 'BCDemo' –DatabaseName 'Demo
Database BC' –DestinationDatabaseName 'Business Central App'
```

In the example, the database server name is **MyServer** , and the SQL Server instance is **BCDemo**. The name of the new application database can be anything. You can specify a name that reflects your application.

The application database is created on the same SQL Server instance as the original database. In the example, if you connect to the **BCDemo** instance using SQL Server Management Studio you will see two databases: the original database, **Demo Database BC**, and the new application database, **Business Central**.

At this stage, the original database still contains the application tables, and you can still access it using the Dynamics NAV Development Environment. Next, you must remove the application tables from the original database to make it a tenant database.

> **TIP**
> Optionally, you can combine the Export-NAVApplication and Remove-NAVApplication cmdlets. For an example of how you can combine the two cmdlets, see the **Example** section.

6. To remove the application tables from the original database, run the Remove-NAVApplication cmdlet as follows:

```
Remove-NAVApplication –DatabaseServer <server name> -DatabaseInstance <instance name> –DatabaseName
<name of the original database>
```

For example, to run the cmdlet against the CRONUS International Ltd. demonstration database where you have exported the application tables, type the following command:

```
Remove-NAVApplication –DatabaseServer 'MyServer' –DatabaseInstance 'BCDEMO' –DatabaseName 'Demo
Database BC'
```

You will be asked to confirm that you want to remove the tables.

> **WARNING**
> Running the **Remove-NAVApplication** cmdlet is not reversible. When you have removed the application tables from the database, you cannot import them again. Make sure that you have a full backup available.

At the end of the process, you have two databases. In the example earlier in this topic, the databases are as follows.

DATABASE NAME	DATABASE TYPE	DESCRIPTION
Demo Database BC	Business data database	Contains the data from the original database.
Business Central App	Application database	Contains the tables that define the application.

7. Clear the `DatabaseName` setting in the Business Central Server instance configuration file by using the Set-

NAVServerConfiguration cmdlet:

```
Set-NAVServerConfiguration –ServerInstance <server instance name> –element appSettings –KeyName
'DatabaseName' –KeyValue ''
```

8. Start the Business Central Server instance by using the Start-NAVServerInstance cmdlet:

```
Start-NAVServerInstance –ServerInstance <server instance name>
```

9. Mount the application database on the Business Central Server instance by using the Mount-NAVApplication cmdlet:

```
Mount-NAVApplication –ServerInstance <server instance name> –DatabaseServer <server name\instance
name> –DatabaseName <application database name>
```

10. Mount the business data database (tenant) on the server instance by using the Mount-NAVTenant cmdlet:

```
Mount-NAVTenant –ServerInstance <server instance name> -Id <tenant name> –DatabaseServer <server
name\instance name> -DatabaseName <business data database> -OverwriteTenantIdInDatabase
```

11. Reinstall extensions.

    In this step, you reinstall the same extensions that were installed on the tenant before. To install an extension, you use the Install-NAVApp cmdlet.

    a. It can be helpful to get a list of the published extensions and their version. To get a list of all published extensions, use the Get-NAVAppInfo cmdlet:

    ```
    Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
    ```

    b. If your solution uses the System Application, install this first.

    ```
    Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version
    <extension version>
    ```

    Replace `<extension version>` with the exact version of the published System Application.

    c. Install the Base Application.

    ```
    Install-NAVApp -ServerInstance <server instance> -Name "Base Application" -Version <extension
    version>
    ```

    Replace `<extension version>` with the exact version of the published Base Application.

    d. Install the Application extension. This extension is required to install Microsoft extensions.

    ```
    Install-NAVApp -ServerInstance <server instance> -Name "Application" -Version <extension
    version>
    ```

    Replace `<extension version>` with the exact version of the published Application extension.

    e. Install other extensions, including Microsoft and third-party extensions.

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version
<extension version>
```

For more information, see Business Central Windows PowerShell Cmdlets.

Example

The following code example illustrates how you can manually write commands in the Business Central administration shell. The commands create an application database based on an existing Business Central database.

The sample commands are assumed to run in the Business Central administration shell based on the CRONUS International Ltd. demonstration database on a local computer.

```
Stop-NAVServerInstance –ServerInstance 'businesscentral_server_instance'
Export-NAVApplication –DatabaseServer 'MyServer' –DatabaseInstance 'BCDEMO' –DatabaseName 'Demo Database BC'
–DestinationDatabaseName 'Business Central App'| Remove-NAVApplication –DatabaseName 'Demo Database BC' -
Force
Set-NAVServerConfiguration –ServerInstance 'businesscentral_server_instance' –element appSettings –KeyName
'DatabaseName' –KeyValue ''
Start-NAVServerInstance –ServerInstance 'businesscentral_server_instance'
Mount-NAVApplication –ServerInstance 'businesscentral_server_instance' –DatabaseServer 'MyServer\BCDEMO' –
DatabaseName 'Business Central App'
Mount-NAVTenant –ServerInstance 'businesscentral_server_instance' -Id tenant1 –DatabaseServer
'MyServer\BCDEMO' -DatabaseName 'Demo Database BC' -OverwriteTenantIdInDatabase
```

In the example, the commands stop the Business Central Server service, creates the application database, clears the default database name in the server configuration, and then restarts the service. Then, the application database and the tenant database are mounted, and the configuration is saved in the Tenants.config file on the server. As a result, you have an application database and a single-tenant deployment. When you try to open the Dynamics NAV Client connected to Business Central, an error displays because you have not specified a tenant. So in the **Select Server** window, in the **Server Address** field, add the tenant ID to the address. In this example, the address is **localhost:7046/BC160/tenant1**.

TIP

For an example of how you can automate the process of transferring user accounts from the original database to the new application database, see the HowTo-ExportNAVApplicationDatabase.ps1 sample script. This and other sample scripts are in the **...\Windows PowerShell\Multitenancy** folder on the Business Central product media. The ExportNAVApplicationDatabase.ps1 sample script can be run in the context of the NAVUpgradeSamples.psm1 script module file. When you call a script such as this, it will export the application tables to a new application database and copy all accounts and SQL Server user roles to the application database. To only transfer the account that the Business Central Server instance uses, use the *–ServiceAccount* parameter for the **Export-NAVApplication** cmdlet. In the examples in this topic, this parameter has not been specified. As a result, the default account, NT AUTHORITY\NETWORK SERVICE, is set up with the required user roles.

See Also

Migrating to Multitenancy
Business Central Windows PowerShell Cmdlets

Installation Considerations for Microsoft SQL Server and Business Central

2/17/2021 • 7 minutes to read • Edit Online

This article describes the requirements for installing and configuring Microsoft SQL Server to work with Business Central.

Dynamics NAV can run on Microsoft SQL Server and Microsoft Azure SQL Database. For a list of supported editions of SQL Server, see SQL Server Requirements.

Using Microsoft SQL Server

Storage

Use different disks or disk partitions for the following:

- Windows operating system.
- Data files for the system databases.
- Log files for system and user databases.
- Data and log files for the TempDB database.

For optimal read/write performance, make sure that disks that are used for SQL Server data files are formatted using 64 KB block size.

Virus scanning

To help you decide which kind of antivirus software to use on the computers that are running Microsoft SQL Server in your environment, see How to choose antivirus software to run on computers that are running SQL Server.

Memory

For optimal read performance, maximize the available memory on the server according to the version and edition of SQL Server used. Refer to the SQL Server documentation for maximum values.

SQL Server components

If you are installing Microsoft SQL Server for use with Business Central, then install the following components:

- Database Engine Services
- Client Tools Connectivity
- Management Tools - Complete

Setup options for Microsoft SQL Server

When you are running Microsoft SQL Server Setup, you must provide additional information. Your responses can affect how you use SQL Server with Dynamics NAV.

TempDB database configuration

For servers with less than 8 cores, create as many data files for the TempDB database as the number of cores. For servers with more than 8 cores, start with 8 data files, and increment with 4 files at a time, if needed.

Make sure that all data files for the TempDB database are of the same size.

Consider putting data and log files for TempDB on a local SSD drive if you are using SAN storage.

Data file and log file configuration

Auto-growth of the database and/or transaction log files in production can degrade performance as all transaction must queue up and wait for SQL Server to grow the file before it can begin to process transactions again. This can create bottlenecks. We strongly recommend growing data and log files during off-peak periods and by 10% to 25% of the current size. We do not recommend disabling "Auto-Grow", as in an emergency it is still better to have SQL Server to auto-grow files than to run out of disk space and bring the database down.

Max degree of parallelism (MAXDOP)

The SQL queries generated by Dynamics NAV is of OLTP type (many, small transactions). It is therefore recommended to run Dynamics NAV with `MAXDOP` set to the value 1.

On SQL Server 2014, `MAXDOP` can only be set on the instance level, changing an advanced server configuration option. On SQL Server 2016, `MAXDOP` can be set on the database level, changing a database scoped configuration.

Both advanced server configuration options and database scoped configurations can be set by using SQL Server Management Studio, see the SQL Server documentation for details.

> **NOTE**
> If you are running SQL Server Enterprise Edition, index maintenance can be done in parallel. If you run maintenance jobs to do this work in off-peak hours, you might want to set `MAXDOP` back to 0 while running these jobs. On SQL Server 2016, it is possible to set MAXDOP directly in the Rebuild Index Task wizard.

Instance configuration

If you plan on installing the Business Central Demo database, and you want Business Central Setup to use an already installed version of SQL Server (and not to install SQL Server Express), you must create a SQL Server instance named **BCDEMO** in SQL Server before you run Setup. Otherwise, Setup will install SQL Server Express automatically, even if there is a valid version of SQL Server already on the computer. If you do not plan to install the Demo database, or if you have no objection to using SQL Server Express, you are free to use the **default instance** and **Instance ID** on the **Instance Configuration** page, or to specify any instance name.

Database engine service

Each SQL Server instance is run by its own windows service. The following two things are important to configure for these services

Startup options

Enable trace flags 1117 and 1118 as startup options for SQL Server 2014. For SQL Server 2016, these trace flags are enabled by default.

Startup options can be set by using SQL Server Configuration Manager, see the SQL Server documentation for details.

Service account

We recommend that you use dedicated domain user accounts for the Windows services running your Business Central Server instances and your SQL Server instances, instead of a Local System account or the Network Service account.

The Business Central Server account must have privileges on the SQL Server instances and on the Dynamics NAV database(s). See Provisioning the Server Service Account for details.

For installations on SQL Server 2014, consider adding the service account for the SQL Server engine to the **Perform Volume Maintenance Tasks** security policy. For SQL Server 2016, it is possible to do this from the installer.

Database configurations

After Dynamics NAV has been installed, it is important to check a few settings on the Dynamics NAV database(s).

This is especially important for databases, which have been upgraded from previous versions of SQL Server.

Statistics

The databases used by Business Central should have set the options AUTO_CREATE_STATISTICS and AUTO_UPDATE_STATISTICS to the value ON (this is the default behavior and should not be changed)

SQL Server (2014 and earlier) uses a threshold based on the percent of rows changed before triggering an update of the statistics for a table regardless of the number of rows in the table. It is possible to change this behaviour by setting trace flag 2371 as a startup option for the instance. See Knowledge Base article ID 2754171, https://support.microsoft.com/en-gb/help/2754171/controlling-autostat-auto-update-statistics-behavior-in-sql-server for more information about when to set this trace flag.

SQL Server (starting with 2016 and under the compatibility level 130) uses a threshold that adjusts according to the number of rows in the table. With this change, statistics on large tables will be updated more often.

Even with "Auto Update Statistics" enabled, we still strongly recommend running a periodic SQL Agent job to update statistics. This is because "Auto Update Statistics" will only be triggered according to the rules described above. On large tables with tens of millions of records (such as Value Entry, Item Ledger Entry and G/L Entry), a small percentage of data in a given statistic such as [Entry No.] can change and have a material effect on the overall data distribution in that statistic. This can cause inefficient query plans, resulting in degraded query performance until any threshold is reached. We recommend using the T-SQL procedure "sp_updatestats" to update statistics, as it will only update statistics where data has been changed. We recommend creating a SQL Agent Job that runs daily or weekly (depending on transaction volume) during off-peak hours to update all statistics where data has changed.

Index fragmentation

Another important administration task that helps to reduce data size and improve performance is to reduce fragmentation for tables and non-clustered indexes. This article in the SQL Server documentation is a good starting point to learn more: Resolve index fragmentation by reorganizing or rebuilding indexes.

Other database options

We recommend to set the database option PAGE_VERIFY to the value CHECKSUM for all databases (including TEMPDB) as this is the most robust method of detecting physical database corruption. This is the default setting for new installations.

Backup

Remember to setup backup of both system and user databases. Remember also to test restore procedures regularly.

Using Microsoft Azure SQL Database

You can deploy a Business Central database to Azure SQL Database. Azure SQL Database is a cloud service that provides data storage as a part of the Azure Services Platform.

To optimize performance, we recommend that the Business Central Server instance that connects to the database is also deployed on a virtual machine in Azure. Additionally, the virtual machine and SQL Database must be in the same Azure region.

For development and maintenance work on Business Central applications, if the Dynamics NAV Development Environment is installed on the same virtual machine in Azure as the Business Central Server, then you can connect to the Azure SQL database from the development environment.

For more information, see Deploying a Business Central Database to Azure SQL Database.

Data Encryption between Business Central Server and SQL Server

When SQL Server and Business Central Server are running on different computers, you can make this data

channel more secure by encrypting the connection with IPSec. (Other encryption options are not supported.) For information on how to do this, see Encrypting Connections to SQL Server, which is part of SQL Server 2008 Books Online in MSDN library.

See Also

Data Access
Troubleshooting: SQL Server Connection Problems
Deployment

Configuring the Business Central Database

2/17/2021 • 7 minutes to read • Edit Online

For a Business Central Server instance to connect to and access a database in SQL Server, the server instance must be authenticated by the database. As in SQL Server, Business Central supports two authentication modes for database communication: Windows Authentication and SQL Server Authentication. When you set up Business Central, you must ensure that database authentication is configured correctly on both the server instance and database, otherwise the server instance will not be able to connect to the database. By default, Windows Authentication is configured on the server instance and database. With Windows Authentication the Business Central Setup does the work for you.

This article describes how to configure SQL Server Authentication. You perform the configuration in two places: on the databases in SQL Server and on the Business Central Server instance. The procedure is different when the Business Central Server instance is configured as a multitenant server instance than when it is not a multitenant server instance.

Set Up an Encryption Key

When using SQL Server authentication, Business Central requires an encryption key to encrypt the credentials (user name and password) that the Business Central Server instance uses to connect to the Business Central database in SQL Server. The encryption key must be installed on the computer where the Business Central Server is installed and also in the database in SQL Server. In a multitenant deployment, the encryption key must be installed in the application database.

To set up an encryption key, you can use one of the following methods:

- You can create and import your own encryption key by using Business Central Administration Shell cmdlets, as described in the following procedure.
- If you are configuring SQL Server authentication on a Business Central Server instance for the first time, you can use the Business Central Server Administration tool which can automatically create and install a system encryption key. If you decide to use this method, no action is required.

Create and import encryption key

1. In the Business Central Administration Shell, run the New-NAVEncryptionkey cmdlet.

   This creates a file that contains an encryption key. If you already have an encryption key file, you can skip this step.

2. Run the Import-NAVEncryptionkey cmdlet to install the encryption key on the Business Central Server instance and database.

On the computer running the Business Central Server instance, the encryption key file has the name BC160.key and is stored in the `%systemroot%\ProgramData\Microsoft\Microsoft Dynamics NAV\[version]\Server\Keys`. In the database, the encryption key is registered in the dbo.ndopublicencryptionkey table. In a multitenant deployment, the encryption key is registered in the application database.

Configure SQL Authentication on the Database

This section describes how to configure a Business Central database to use SQL Server Authentication with a Business Central Server instance. You can complete the steps in this procedure by using SQL Server Management Studio or Transact-SQL.

> **IMPORTANT**
> In a deployment where the Business Central Server instance is configured as a multitenant server instance, you must complete the following procedure on the application database and tenant database.

Configure SQL Server Authentication on the database in SQL Server

1. Configure the SQL Server instance (Database Engine) that hosts the Business Central database to use SQL Server Authentication.

   To use SQL Server authentication, you configure the database instance to mixed authentication mode (SQL Server and Windows Authentication). For more information, see Change Server Authentication Mode.

2. In the SQL Server instance, create a login that uses SQL Server authentication.

   For more information, see Create a Login.

3. Map the login to a user in the Business Central database, and give the user the relevant privileges in the Business Central database.

   For more information, see Create a Database User in the SQL Server docs and the Giving the service account database privileges in SQL Server in the current article.

Configure SQL Server Authentication on the Business Central Instance (Non-Multitenant)

You configure the Business Central Server instance with the login credentials (user name and password) of the user account in the Business Central database in SQL Server that you want to use for authentication. You can do this using the Business Central Server Administration tool or Business Central Administration Shell.

Configure SQL Authentication on a server instance using Business Central Server Administration tool

1. Open the Business Central Server Administration tool.
2. In the console tree, which is the left pane, expand the node for the computer that contains the Business Central Server instance, and then select the Business Central Server instance.
3. In the **Actions** pane, choose **Database Credentials**.
4. On the **Database Credentials** page, choose the **Edit** button.
5. Set the **Database Authentication Type** to **SQL Authentication**.
6. In the **Database User Name** field, type the login name for the database user that you want to use to access the Business Central database in SQL Server.
7. In the **Password** field, type the login password for the database user that you want to use to access the Business Central database in SQL Server.
8. Choose the **Save** button, and then on the **Enable Encryption on SQL Server Connections** dialog box, choose the **OK** button.

   Encryption keys are used to help secure the login credentials over the connection between the Business Central Server instance and the Business Central database in SQL Server.

9. On the **Information** dialog box about encryption, choose the **OK** button.

   This dialog box is to inform you to enable encryption on SQL Server connections, which is disabled by default.

10. If you want to enable encryption on SQL Server connections, in the **Action** pane, choose **Configuration**, and then choose the **Edit** button. In the **Database** tab, select **Enable Encryption on SQL Connections**, choose the **Save** button, and then the **OK** button.

11. Restart the server instance.

Configure SQL Authentication on a server instance using Business Central Administration Shell

- If you are modifying an existing Business Central Server instance, run the Set-NAVServerConfiguration cmdlet.

  Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

- If you are creating a new Business Central Server instance, run the New-NAVServerInstance cmdlet.

  Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

Configure SQL Server Authentication on Business Central Instance in a Multitenant Deployment

This section describes how to configure a Business Central database to use SQL Server Authentication with a Business Central Server instance. You can complete the steps in this procedure by using SQL Server Management Studio or Transact-SQL.

To configure a SQL Server Authentication on a Business Central Server instance, you set up the server instance with the login credentials (user name and password) for the user accounts for the application and tenant databases in SQL Server. You can do this using the Business Central Server Administration tool or Business Central Administration Shell.

Configure SQL Authentication using Business Central Server Administration tool

1. Open the Business Central Server Administration tool.
2. In the console tree, which is the left pane, expand the node for the computer that contains the Business Central Server instance, and then select the Business Central Server instance.
3. Configure SQL Server Authentication with the application database as follows:
   a. In the **Actions** pane, choose **Database Credentials**.
   b. On the **Database Credentials** page, choose the **Edit** button.
   c. Set the **Database Authentication Mode** to **SQL Server Authentication**.
   d. In the **Database User Name** field, type the login name for the database user that you want to use to access the Business Central application database in SQL Server.
   e. In the **Password** field, type the login password for the database user that you want to use to access the Business Central database in SQL Server.
   f. Choose the **Save** button, and then on the **Enable Encryption on SQL Server Connections** dialog box, choose the **OK** button.

      Encryption keys are used to help secure the login credentials over the connection between the Business Central Server instance and the Business Central database in SQL Server.

   g. On the **Information** dialog box about encryption, choose the **OK** button.

      This dialog box is to inform you to enable encryption on SQL Server connections, which is

disabled by default.

h. If you want to enable encryption on SQL Server connections, in the **Action** pane, choose **Configuration**, and then choose the **Edit** button. In the **Database** tab, select **Enable Encryption on SQL Connections**, choose the **Save** button, and then the **OK** button.

4. To configure SQL Server Authentication with the tenant database, mount the tenant to the Business Central Server instance and specify the login credentials (user name and password) for the database user that you want to use to access the Business Central tenant database in SQL Server.

   If the tenant is already mounted to the Business Central Server instance, you must dismount the tenant, and mount it again.

   For more information see Mount or Dismount a Tenant.

5. Restart the server instance.

Configure SQL Authentication using Business Central Administration Shell

1. Configure SQL Server Authentication with the application database as follows:

   - If you are modifying an existing Business Central Server instance, run the Set-NAVServerConfiguration cmdlet.

     Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

   - If you are creating a new Business Central Server instance, run the New-NAVServerInstance cmdlet.

     Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

2. To configure SQL Authentication with the tenant database, run the Mount-NAVTenant cmdlet.

   Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the tenant database.

See Also

Installation Considerations for Microsoft SQL Server
Deployment
Installing Business Central Using Setup

Creating Databases in Business Central

2/17/2021 • 5 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

This article describes how to create new databases in Business Central for storing application and business data.

Overview

Application and tenant databases

There are two types of databases: application and tenant.

- The application database stores data that defines the application and its business logic. This data includes descriptions of the objects and the code of your application. Essentially, it includes data that are common to all tenants. It's the application database to which you publish your extensions, including the base application, system application, and other extensions.
- The tenant database is often referred as just the tenant. The tenant database stores the actual business data for specific companies, for example, data that users enter and modify by using the application. It's the tenant database on which you install extensions that are published to the application database.

In a single-tenant deployment architecture, the application data and business (tenant) data are stored in the same database. That is, there's only one database. If you want to set up single-tenant deployment, you only have to create an application database.

If you have a multitenant deployment architecture, you create an application database and one or more tenant databases.

Supported collations

Business Central supports Windows collations only. For a list of Windows collations, see Windows Collations in the SQL Server documentation.

Create an application database

To create an application database, for either a single-tenant or multitenant deployment, you use the New-NAVApplicationDatabase cmdlet. This cmdlet is available in the Business Central Administration Shell. You use the New-NAVApplicationDatabase cmdlet to create either new database or initialize an existing empty database to make it an Business Central application database.

- If you create a new database, the cmdlet will add a database in SQL Server. The database includes the tables and data required for a Business Central application database. The cmdlet creates a master data file (MDF) and log data file (LDF). Using the cmdlet, you can set the database name, the collation, and where to store the data files. Other database options are set for you.
- If you use the cmdlet with an existing database, the cmdlet modifies the existing database to include Business Central application tables and data. You configure a database beforehand, setting options that aren't done by the cmdlet, such as options for the data files (MDF/NDF/LDF) and their filegroups, table partitioning, and more.

To create an application database, complete the following steps:

1. Run the Business Central Administration Shell as an administrator.

Make sure that the Window user that you run as has the appropriate privileges on the SQL Server as described in Assign privileges on the Business Central database-level.

2. Run the New-NAVApplicationDatabase cmdlet to create a new database or initialize an existing database.

```
New-NAVApplicationDatabase [-DatabaseServer <database server name>\<database server instance>] -
DatabaseName <String> [-DatabaseLocation <String>] [[-Collation] <String>] [-
ApplicationDatabaseCredentials <PSCredential>]
```

To create a new application database, set `-DatabaseName` to the name you want to give the database, and the `-Collation` to the wanted collation. If you omit the `-Collation`, then Latin1_General_100_CS_AS is used by default. Optionally, set the `-DatabaseLocation` to the directory path where you want to store the data files; otherwise the database files will be stored in the default SQL Server location.

To use an existing empty database, set the `-DatabaseName` to the name of the existing database. You can't use the `-Collation` parameter to change the current collation of the database.

The following example creates a new database called MyBCApplicationDB. The database is given the collation Latin1_General_100_CS_AS on the SQL Server instance BCDEMO. The database files are stored in the default data directory for the SQL Server instance (for example, C:\Program Files\Microsoft SQL Server\MSSQL13.BCDEMO\MSSQL\DATA).

```
New-NAVApplicationDatabase -DatabaseServer .\BCDEMO -DatabaseName "MyBCApplicationDB"
```

When the database has been successfully created, text similar to the following displays:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : MyBCApplicationDB
DatabaseCredentials :
DatabaseLocation    :
Collation           : Latin1_General_100_CS_AS
```

3. Connect Business Central Server instance to the new database.

For example:

```
Set-NAVServerConfiguration BC160 -KeyName DatabaseName -KeyValue "MyBCApplicationDB"
```

4. Run the Set-NAVApplication cmdlet to set the application version on the database.

To set the application version, use the `-ApplicationVersion` parameter. The value must have the format `major.minor.[build[.revision]]`, such as '15.1', '15.1.0', or 15.1.0.0'. For example:

```
Set-NAVApplication BC160 -ApplicationVersion 15.1.0.0 -Force
```

This step is required to synchronize your tenant and extensions later. This step sets a value to the `applicationversion` column in the **ndodbproperty** table of the application database.

5. Give the Business Central Server service account privileges to the database.

For more information, see Provisioning the Business Central Server Service Account.

If you have a multitenant deployment, go to the next section about creating a tenant database.

6. If you have a single-tenant deployment, you can now synchronize the tenant. For a multitenant

deployment, go to the next section.

Run the Sync-NAVTenant cmdlet to synchronize the tenant. For example:

```
Sync-NAVTenant -ServerInstance BC160
```

This step will create the tenant-related tables in the database.

Create a tenant database

Complete the followings step to create a new tenant database in a multitenant deployment.

1. In SQL Server, create a new database.

   > **IMPORTANT**
   > Set the collation to the same as the application database.

2. Give the Business Central Server service account privileges to the database, like you did with the application database.

   For more information, see Provisioning the Business Central Server Service Account.

3. Mount the database as a tenant to the application.

   To mount the tenant, use the Mount-NAVTenant cmdlet. For example:

   ```
   Mount-NAVTenant -ServerInstance BC160 -DatabaseName "BCTenantDB" -DatabaseServer .\BCDEMO -Tenant
   BCTenant1 -AllowAppDatabaseWrite
   ```

   > **NOTE**
   > For now, we recommend that you use the `-AllowAppDatabaseWrite` parameter. Later, you can dismount and mount the tenant again without the parameter if needed.

4. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet. For example:

   ```
   Sync-NAVTenant -ServerInstance BC160 -Tenant BCTenant1 -Mode Sync
   ```

   This step will create the tenant-related tables in the database.

Next steps

Complete the following steps to get the application up and running on your tenants.

1. Publish the system symbols and extensions that make up your application. For example, publish the system application, base application, Microsoft extensions, and third-party extensions.

   See Publishing and Installing an Extension

2. Synchronize and install extensions on the tenant.

   See Publishing and Installing an Extension

3. Add a company to the database.

   To add a company, run the New-NAVCompany cmdlet.

4. Export and import existing permissions sets.

   A new database won't include any permission sets except for the SUPER permission set. Also, there will only be one user, typically for your account. You can either create permissions sets from scratch or export the sets from an existing database. As a minimum, it's a good idea to export the BASIC permissions set. The BASIC permission set grants the minimum permissions required for any user to access the application.

   For more information, see Export Permission Sets and Special Permission Sets.

See also

Changing Collation of Existing Database
Multitenant Deployment Architecture in Business Central
Creating and Altering Business Central Databases (Spring 2019 and earlier)

Running a Business Central Database on Azure SQL Database

2/17/2021 • 7 minutes to read • Edit Online

This article describes how you can deploy a Business Central database to Microsoft Azure SQL Database.

> **TIP**
> For multi-tenancy mode installations, the procedures outlined here must be performed both for the application database as well as for the tenant databases.

Prerequisites

Make sure that you have the following prerequisites for completing this procedure:

- A Microsoft Azure subscription and access to the Azure portal.
- A Business Central database installed on a SQL Server Database Engine instance.

Create and configure an Azure SQL Database Server

In the Azure portal, create an SQL Database Server for hosting the Business Central database. For more information about how to create and configure an SQL Database server, see Create your first Azure SQL Database.

Here are some important notes when creating the Azure SQL Database:

1. Specify a login name and password for the server. You'll use this information in the next steps when you deploy the Business Central database to Azure SQL and set up the Business Central Server to authenticate with the database.
2. Configure the server to allow for access by Windows Azure Services.
3. Make a note of the SQL Database server name because you'll need it later.

   The name has a format similar to: `mysqldatabaseserver.database.windows.net`.
4. Configure the server firewall to allow for access by the IP address of the computer that you're using to deploy the Business Central database.

   For information, see How to: Configure Firewall Settings (Azure SQL Database).

Prepare the Business Central Database(s)

Make sure the database meets these requirements:

1. Upload a valid Business Central license file to the database.

   For more information, see Uploading a License File for a Specific Database.
2. Delete all users of the database that use Windows authentication.

   Make sure to delete `NT AUTHORITY\NETWORK SERVICE` and `NT AUTHORITY\SYSTEM`. Only users with SQL

authentication are allowed in Azure SQL Database.

3. Delete any deadlock monitors for the Business Central database.

   SQL Server Management Studio, you can run a query similar to the following query:

   ```
   use [Demo Database BC (14-0)]

   DROP VIEW [dbo].[deadlock_report_event_file_view]
   ```

   For more information about the deadlock monitor, see Monitoring SQL Database Deadlocks.

Deploying using BACPAC files

For smaller databases (typically up to 50 GB), you can deploy them using a BACPAC file. This method requires the database to be off-line (the Business Central Server instance cannot be connected to the database).

For multi-tenancy mode installations, complete the steps for both the application database and the tenant databases.

Export Business Central Database to a BACPAC File

When you deploy your application online, you must provide a compressed .zip file that contains the database as data-tier application file, known as BACPAC (.bacpac) file. This article describes how you to create the BACPAC files and zip. You can do this using SQL Server Management Studio or the SqlPackage.exe command-line tool.

1. In SQL Server Management Studio, connect to the server instance that hosts the database.
2. In **Object Explorer**, right-click either the database, choose **Task**, and then choose **Export Data-tier Application**.
3. Follow the steps in the **Export Data-tier Application** wizard to export the database to a BACPAC file on your computer or network.

   You can use any name for the BACPAC file. For more information about exporting databases to BACPAC format, see Export a Data-tier Application.

Import the BACPAC to Azure SQL

In this task, you'll import the BACPAC files to the Azure SQL Database server instance.

1. In SQL Server Management Studio, connect to Azure SQL Database server instance that you created.
   a. Select **File** > **Connect Object Explorer**.
   b. In the **Server Name**, enter the server name assigned to your Azure SQL Database server.

      For example, *mysqldatabaseserver.database.windows.net*. You can get this information from the **Overview** page in the Azure portal.

   c. Set **Authentication** to **SQL Server Authentication**.
   d. Enter the login name and password that you set up in the first task when creating the Azure SQL Database server.
   e. Select **Connect**.
2. Import the BACPAC file that you created for the Business Central Database.

   This step creates a new database on the Azure SQL Database server instance. The database is based on the BACPAC file. It uses the same schema and includes all the data of the original database.

a. In **Object Explorer**, right-click the **Databases** folder for the Azure SQL database, and select **Import Data-tier Application**.

b. Follow the wizard. On the **Import Settings** page, browse for the BACPAC that you create, and choose **Next**.

   During this step, you'll specify a name for the database. You can give it any valid name you like.

c. Review the **Database Settings** page. Make changes if needed, and then choose the **Next** > **Finish**.

Now, you're ready to configure the Business Central Server instance.

Deploying/migrating existing databases

For larger databases (typically bigger than 50 GB), deployments with BACPAC file aren't recommended. Instead, we recommend using the migration tools and options described in the Azure Database Migration Guide.

Configure the Business Central Server instance

The next task is to configure the Business Central Server instance to connect to the databases in Azure SQL Database. You can configure the server instance using the Business Central Server Administration tool or the Set-NAVServerConfiuration cmdlet of Business Central Administration Shell.

1. Set up SQL Server authentication that uses the login account for the application database in Azure SQL Database.

   For more information, see Configuring SQL Server Authentication.

2. Configure the database connection settings to point to the application database:

SETTING (KEY)	VALUE
Database Instance (DatabaseInstance)	Empty
Database Name (DatabaseName)	The name of the database in Azure SQL Database.
Database Server (DatabaseServer)	The name of the Azure SQL database server, for example, *mysqldatabaseserver.database.windows.net*.

   For more information, see Configuring Business Central Server.

3. Restart the Business Central Server instance.

   The Business Central database is now deployed and configured on Azure.

   - If you have a single-tenant server instance, your deployment is ready to use.
   - If you have a multitenant server instance, go to the next step.

4. Mount the tenant database (multitenant-only).

   The final step in a multi-tenant deployment is to mount Azure SQL Database that holds the business data as a tenant on the Business Central Server instance. Mount the tenant using the Business Central Server Administration tool or Mount-NAVTenant cmdlet of Business Central Administration Shell.

   When mounting the tenant, provide the following information:

SETTING (KEY)	VALUE
Database Name (DatabaseName)	The name of the tenant database in Azure SQL Database.
Database Server (DatabaseServer)	The name of the Azure SQL database server, for example, *mysqldatabaseserver.database.windows.net*.
Login name and password (DatabaseCredentials)	The credentials of the login account used of Azure SQL Database server instance.

For more information, see Mount or Dismount a Tenant on a Microsoft Dynamics Server Instance.

Additional information

Changing database login account database

If you want to use a different login account for the database, do the following steps:

1. Create a new login that uses SQL Server authentication.

   For more information, see Create a Login.

2. Map the login to a user in the Business Central database, and add the user to the **db_owner** role of the Business Central database.

   For more information, see Create a Database User.

Colocation of the Business Central Server instance and the database

To minimize network latency between the Business Central Server instance and a database running on Azure SQL and optimize performance, we recommend that the Business Central Server instance that connects to the database is deployed onto a virtual machine in Azure. Additionally, the virtual machine and the database must be in the same Azure region.

Differences between Azure SQL database and SQL Server

When comparing the performance of Azure SQL database and a SQL Server deployed to a virtual machine, a few things are important to take into account:

- Azure SQL database is a high-availability database with a number of secondary replicas where writes must be confirmed for commits to be done. Secondary replicas will affect the performance of write operations (any inserts, updates, or deletes). For high throughput systems, running the database on SQL Server might be a better option. In this case, high-availability can be achieved using delayed durability or async commits. This condition will affect the data volume that you're ready to sacrifice if a database crash occurs.
- Azure SQL database is a scalable and intelligent service that includes features such as high-availability, backups, automatic index tuning, security vulnerability assessment, and advanced threat detection. When choosing a pricing tier for running your production database, make sure that you compare Azure SQL database to a similar setup for SQL Server on a virtual machine (including the cost of administration and maintenance).
- The performance of an Azure SQL database is correlated to the price you pay for usage, and new pricing models evolve over time. Use the scalability options within the service to change the database performance tier for your installation as your workload changes over time.

See Also

Installation Considerations for Microsoft SQL Server
Optimizing SQL Server Performance

Reducing Data Stored in Business Central Databases

2/17/2021 • 2 minutes to read • Edit Online

The article provides an overview of the different ways to reduce the amount of data stored in a Business Central database. Reducing the size of your database serves the following purposes:

- Prevents reaching the size limit when using Business Central online.
- Improves runtime performance.
- Improves database management processes, like backing up and restoring databases.

These measures are typically done by an application administrator or developer.

Delete unused companies

If you have companies that are no longer needed, such as test companies or the Demo company, delete these companies.

Delete documents

Over time, the database will accumulate historical data for documents, like invoiced purchase orders. If these documents are no longer needed, delete them.

For more information, see Manage Storage by Deleting Documents or Compressing Data.

Use retention policies

Retention policies allow you to specify how frequently you want Business Central to automatically delete outdated data in tables that contain log entries and archived records.

For more information, see Define Retention Policies.

Migrate BLOB data types to Media or MediaSet

Data in Media or Media set data types aren't counted in the database limit. As an extension developer, consider migrating data from blobs to the Media or MediaSet datatypes for your own extensions.

For more information, see Working With Media on Records.

Compress tables

Business Central supports data compression of tables. Compressing table data saves space and helps improve performance of I/O-intensive workloads.

> **NOTE**
> With Business Central online, page-level data compression is automatically enabled on tables in a tenant. It's applied to Microsoft extensions and third-party extensions, unless the **CompressionType** property on a table is explicitly set to `None`.

There are two ways to enable or change data compression in Business Central:

- On a table-level, use the CompressionType property on table objects.
- On a database-level (on-premises only), use the Start-NAVDatabaseCompression cmdlet. This cmdlet is only available with Business Central 2020 release wave 1 and later.

For more information about data compression, see Data Access - Using SQL Server data compression.

See Also

Managing Capacity
Creating Databases in Business Central

Introducing the Dynamics 365 Business Central Mobile App

2/17/2021 • 3 minutes to read • Edit Online

The app displaying the Business Central tablet client and Business Central phone client is targeted at users in small and medium sized businesses that want to access data from a tablet or a phone. The main advantages of this offering are portability and flexibility, which allows end users to perform tasks when they are away from their desk. Having a Dynamics 365 Business Central solution that runs on a smaller device also brings it in the hands of many more users and your app is easy to distribute.

> **IMPORTANT**
>
> The Business Central tablet client and Business Central phone client do not replace the Business Central Web client. Instead, they offer a touch interface for a limited set of application scenarios compared to the Business Central Web client.

The Business Central Web client supports more complex business processes and heavier data entry than it is possible on the Business Central tablet client and Business Central phone client. Business Central is also designed for intensive use, and the user can have multiple windows open at the same time, but this is neither possible in Business Central phone client or Business Central tablet client. For more information, see Differences and Limitations When Developing Pages for the Business Central Mobile App.

The design for the Business Central tablet client is optimized for the touch experience and reduced use of the on-screen keyboard. On the other hand, the design for the Business Central phone client is about touch optimization, given its smaller screen size. The Business Central phone client layout is designed to support one-hand and both hands use, which allows the important data and buttons to be available within thumbs reach.

> **NOTE**
>
> In this documentation, you will see mentions of Business Central tablet client, Business Central phone client, and *Dynamics 365 Business Central Mobile App*. Business Central tablet client and Business Central phone client describe the interface tailored to the category of mobile device, which is one of the tools available for developers for designing mobile solutions, whereas *Dynamics 365 Business Central Mobile App* is the common name for the app across all devices; the end result made with these tools.

Considering the user scenarios

When you design your solution for the Business Central tablet client and the Business Central phone client, you must make sure that scenarios are simple enough to be meaningful and usable. The tablet and phone designs are meant for lighter tasks and are useful, for example, for traveling salespeople or service technicians who need a portable, online, easy-to-use app that provides an overview, for example, of daily tasks and items in stock.

Depending on the scenarios that your tablet and phone solution will support, it will either make sense to create a new Role Center for tablet and phone only, or share the same Role Center across all of the client types. In some cases, it can make sense to have a user sign in with two different profiles, one for a desktop client and one for mobile devices. In other cases, duplicating pages and designing specific duplicates to be device-oriented is the best solution.

If you have existing page objects that you want to make available on Business Central tablet client or Business Central phone client, we strongly recommend that you plan time to evaluate carefully which actions, sections,

and fields will be needed for the user scenarios you want to enable. Fields and actions that are not needed should not be visible to users of your app. The UI must be simplified significantly to work well on a small device. For more information, see Designing for Different Screen Sizes on Tablet and Phone.

Supported credential types

Business Central tablet client and Business Central phone client support the same credential types as Business Central Web client. For more information, see Authentication and Credential Types.

See Also

Getting Started Developing for the Dynamics 365 Business Central Mobile App
Differences and Limitations When Developing Pages for the Dynamics 365 Business Central Mobile App

Preparing For and Installing the Microsoft Dynamics 365 Business Central App

2/17/2021 • 2 minutes to read • Edit Online

This article describes how to prepare for and install the Business Central App when you have on-premises solution. If you have a Business Central Online, there is no preparation (see Getting Business Central on Your Mobile Device).

The Business Central App is available for download for devices that use following operating systems (OS):

OS	DESCRIPTION	DOWNLOAD
iOS	Mobile app for iPad and iPhone	App Store
Android	Mobile app for Android phones	Google Play
Windows 10 or Xbox One	A companion desktop app that mimics that Business Central Web client but has the same look-and-feel as mobile apps.	Windows Store

Like the Business Central Web client, the Business Central App relies on a Business Central web server instance that connects to Business Central Server and database. The Business Central App uses the same application code as the Web client, except it renders the application in a different format.

Preparing the environment

To install a working Business Central App, the following requirements must be met:

- You must have a working Business Central on-premises solution that includes a web server instance, server, and database (application and business data).
- The Business Central Web Server instance must be configured for:
  - https (SSL)
  - For the iOS app, the navsettings.json file must include the `"GlobalEndPoints"` parameter with the following endpoints as a minimum: null, ms://businesscentral, and ms://dynamicsnav. For example:

    ```
    "GlobalEndPoints": "null,ms://businesscentral,ms://dynamicsnav"
    ```

    You can set other endpoints, but these endpoints must be included for the app to work.
- You are set up as a user in Business Central.

  If you installed Business Central, then by default, your Windows account has been added as a user.
- The computer on which you install the app must meet the requirements outlined on the download page for the app.

Install the Business Central App

1. Go to the download page from links provided in the preceding table, and select **Get**.

   You can also install the desktop app by running setup.exe from the Dynamics 365 Business Central installation media (DVD). Follow Setup until you come to the **Dynamics 365 Business Central** page, select **Get the Business Central app from the Microsoft Store**, and then **Get**.

2. On the **Welcome** page, select **Connect to a local or hosted service**.

3. In the **Service name** box, enter the URL for your Business Central Web Server instance.

   This is the same URL that you use for opening the Business Central Web client, and has the format: `https://<hostname>:<port>/<web server instance>`, for example, `https://mycomputer:443/BC150`.

   Select the arrow to continue.

4. When prompted, provide a valid user name and password for accessing Business Central.

   For example, if your deployment is using Windows authentication, this would be your Windows user name and password.

   Select the arrow to continue.

   When completed, the Business Central App will open.

See Also

Troubleshooting the Business Central Mobile App On-Premises
Installing Business Central Using Setup
Configuring Business Central Web Server Instances
Components
Deployment
Web Client URL

Troubleshooting the Business Central Mobile App On-Premises

2/17/2021 • 2 minutes to read • Edit Online

This section contains information to help you resolve problems using the Business Central Web client, the Business Central tablet client, and the Business Central phone client when working on-premises.

The first thing to do is to check that the environment meets the prerequisites for the mobile app. For more information, see Preparing the environment.

Troubleshooting articles

Troubleshooting: Icon Font Not Loaded

Troubleshooting: Device Date is Causing Connection Issues

Troubleshooting: Client Returns Wrong CLIENTTYPE

See Also

Getting Started Developing for the Business Central Mobile App

Using HTTPS and Certificates in Business Central Mobile App

2/17/2021 • 2 minutes to read • Edit Online

Business Central Mobile App only works using an HTTPS protocol. This means that you must configure a valid certificate on the server. If a trusted, valid, and signed certificate is configured on the Business Central Server, the end user does not have to do anything. However, if a self-signed certificate is configured on the Business Central Server, the end user must install a certificate on their device. The steps for implementing certificates will vary depending on the platform the user is using.

For more information, see Using Security Certificates with Business Central On-Premises.

> **IMPORTANT**
> It is recommended to *only* use self-signed certificates for testing purposes and never in a production environment.

See Also

Getting Started Developing for the Dynamics 365 Business Central Mobile App
Introducing the Dynamics 365 Business Central Mobile App

Administration of Business Central On-Premises

2/17/2021 • 2 minutes to read • Edit Online

Business Central provides different tools for different administration tasks.

On-premises administration

TO	SEE
Learn about the Business Central Server Administration Tool, a Microsoft Management Console snap-in that you use to create and manage Business Central Server instances.	Business Central Server Administration Tool
Perform administration tasks with the Business Central Windows PowerShell cmdlets.	Business Central PowerShell Cmdlets
Optimize performance when accessing data from SQL Server.	Optimizing SQL Server Performance with Business Central

See Also

Administration of Business Central Online
The Business Central Administration Center
Cloud Solution Provider program - selling in-demand cloud solutions
Deployment

Business Central Server Administration tool

2/17/2021 • 4 minutes to read • Edit Online

The Business Central Server Administration tool is a Microsoft Management Console (MMC) snap-in for creating and managing Business Central Server instances.

> **TIP**
> You can also administrate your Business Central deployment using Windows PowerShell cmdlets. For more information, see Microsoft Dynamics 365 Windows PowerShell Cmdlets.

Install the Business Central Server Administration tool

To install Business Central Server Administration tool, use the Business Central Setup and choose either **Server Option** or **Administration Tool** under the custom options page. For more information, see Installing Business Central Using Setup.

Run the Business Central Server Administration tool

You typically run the Business Central Server Administration tool by choosing **Business Central Administration** from the Start menu. Or, you can open the MMC first and then add the Business Central snap-in. In this case, choose **Run** from the Start menu and then specify the Microsoft Management Console:

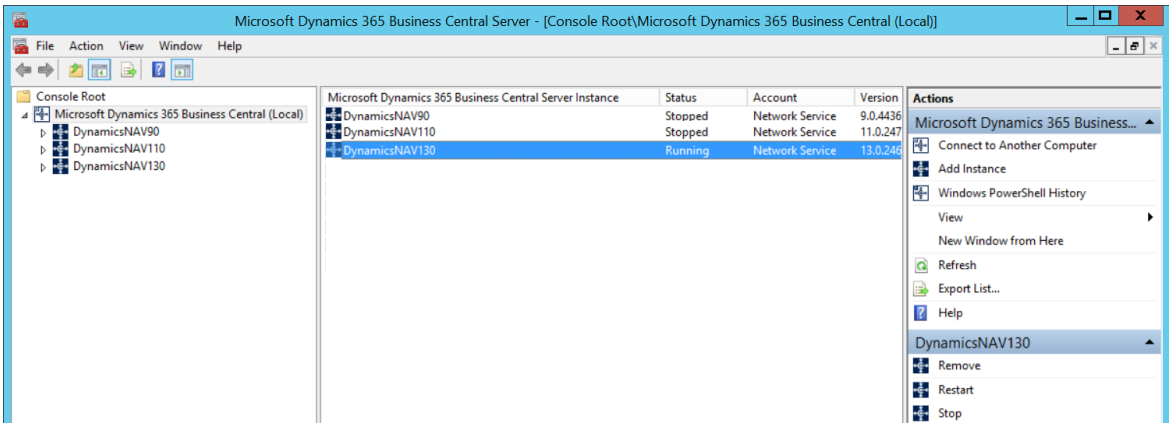
```
mmc
```

> **IMPORTANT**
> Only members of the Administrator group on the computer are able to use the Business Central Server Administration tool.
>
> The Business Central Server Administration tool is not supported for multi-user environments.

Navigating the Business Central Server Administration tool

Business Central Server Administration tool is divided into panes:

- The left pane shows a tree view that lists all Business Central Server computers that you are administering from this computer and all Business Central Server instances on those computers.



- The **center pane** shows information about the item that you have selected in the left pane. When the selected item is a computer running Business Central Server, the center pane shows a list of Business Central Server instances on that computer and the status of each instance (running or stopped), and the name of the account the instance is running under.

  When the item selected in the left pane is a Business Central Server instance, the center pane shows the settings for that instance. For information about a specific setting, see Configuring Business Central Server Instances.

- If the Business Central Server is configured for multitenancy, then you can expand the Business Central Server instance items in the left pane to display a **Tenants** item. Select the **Tenants** item to display all the tenants that are mounted on a Business Central Server instance in the center pane. For more information, see Multitenant Deployment Architecture

- The **right pane** displays available actions for the object that is selected in the left pane. These options differ depending on whether a Business Central Server computer or a Business Central Server instance is selected.

- The **Windows PowerShell History** pane lists the Windows PowerShell commands that the equivalent of the tasks you perform in the Business Central Server Administration tool. You can access the Windows PowerShell History pane from the **Actions** menu and from the right pane. To run a command that is shown in the **Windows PowerShell History** pane, you can copy the command and paste it into the Dynamics NAV Administration Shell, for example.

Connect to remote computers and multiple server instances

You can use the Business Central Server Administration tool to connect to other computers on your network where Business Central Server instances are installed, and then manage those instances.

1. Configure the remote computers to receive Windows PowerShell remote commands by running the Enable-PSRemoting cmdlet on each computer.

2. Next, you can start the Business Central Server Administration tool.

3. If you want to connect to a single remote computer, you can start the Business Central Server Administration tool from the Start menu of your computer.

4. If you want to connect to multiple remote computers, you must start the Business Central Server Administration tool from the **Run** program that you can access in the Start menu,

   a. Enter the command `mmc`.
   b. In the Management Console, on the **File** menu, choose **Add/Remove Snap-in** to open the **Add or remove Snap-ins** dialog box.
   c. In the **Available snap-ins** list, double-click **Microsoft Dynamics 365 Business Central**.

d. In the **Connect to another computer** dialog box, type the name of a Business Central Server computer in the **Server Name** box, and then choose **OK**.
e. Double-click *Microsoft Dynamics 365 Business Central** again, and then enter the name of a different Business Central Server computer in the **Server Name** box. Choose **OK**.

You can also accept the default value, which is **(Local)**, if this is one of the Business Central Server computers that you will be administering.

f. Continue selecting Business Central Server computers as needed. When you are finished selecting computers, choose **OK** to close the **Add or remove Snap-ins** dialog box.

Now you see multiple Business Central Server computers listed in the tree view pane of the Business Central Server Administration tool.

> **TIP**
> When you close MMC, you are prompted to save your settings to a Microsoft Management Console (.msc) file. If you save your settings, then you can use this file to open MMC with your Business Central Server computers already listed

See Also

Configuring Business Central Server Instances
Administration Center API

Authentication and Credential Types for Dynamics 365 Business Central

2/17/2021 • 4 minutes to read • Edit Online

In Business Central online, users are added through the Microsoft 365 admin center. Once users are created in Microsoft 365, they can be imported into the **Users** window in Business Central. For more information, see Managing Users and Permissions in the business functionality content.

Configuring Authentication for On-Premises Deployments

An on-premises deployment of Business Central supports several credential authorization mechanisms for users. When you create a user, you provide different information depending on the credential type that you are using in the current Business Central Server instance.

> **IMPORTANT**
>
> All users of a Business Central Server instance must be using the same credential type. In on-premises deployments, you can specify which credential type is used for a particular Business Central Server instance in the Business Central Server Administration tool.

Credential Types

Business Central on-premises supports the following credential types.

CREDENTIAL TYPES	DESCRIPTION
Windows	With this credential type, users are authenticated using their Windows credentials. You can only specify Windows as the credential type if the corresponding user exists in Windows (Active Directory, local workgroup, or the local computer's users). Because they are authenticated through Windows, Windows users are not prompted for credentials when they access Business Central.
UserName	With this setting, the user is prompted for username/password credentials when they access Business Central. These credentials are then validated against Windows authentication by Business Central Server. There must already be a corresponding user in Windows. Security certificates are required to protect the passing of credentials across a wide-area network. Typically, this setting should be used when the Business Central Server computer is part of an authenticating Active Directory domain, but the computer where the Dynamics NAV Client connected to Business Central is installed is not part of the domain.

CREDENTIAL TYPES	DESCRIPTION
NavUserPassword	With this setting, authentication is managed by Business Central Server but is not based on Windows users or Active Directory. The user is prompted for username/password credentials when they start the client. The credentials are then validated by an external mechanism. Security certificates are required to protect the passing of credentials. This mode is intended for hosted environments, for example, where Business Central is implemented in Azure.
AccessControlService	With this setting, Business Central relies on Azure Active Directory (Azure AD) for user authentication services. Azure AD is a cloud service that provides identity and access capabilities, such as for applications on Azure, in Microsoft Microsoft 365, and for applications that install on-premises. If the Business Central Server instance is configured to use AccessControlService authentication, you can specify an Azure AD account for each user in the **Office 365 Authentication** field so that they can access both the Business Central and their Microsoft 365 site. Also, if you use Business Central in an app for SharePoint, users have single sign-on between the SharePoint site and Business Central. For more information, see Authenticating Users with Azure Active Directory or Authenticating Users with Active Directory Federation Services. Security certificates are required to protect the passing of credentials across a wide-area network.
None	For internal use on system sessions and typically should not be used. If you choose **None**, then the Business Central Server instance cannot start.
ExchangeIdentity and TaskScheduler	For internal use only. Do not use.

IMPORTANT

If Business Central Server is configured to use NavUserPassword or AccessControlService authentication, then the username, password, and access key can be exposed if the SOAP or OData data traffic is intercepted and the connection string is decoded. To avoid this condition, configure SOAP and OData web services to use Secure Socket Layer (SSL). For more information, see How to: Implement Security Certificates in a Production Environment in the ITPro content for Microsoft Dynamics NAV 2018.

Configuring the Credential Type for Client and Server

For on-premises deployment, you must make sure that clients and Business Central Server are configured to use the same credential type.

When you change the credential type for a Business Central Server instance and the relevant client configurations, the changes take effect when you restart the Business Central Server instance and users connect to the instance again.

Server Configuration

To edit the configuration for the Business Central Server instance, you can use either the Business Central Server Administration tool or the Business Central Administration Shell. In the Business Central Server Administration tool, you configure the credential type in the **Credential Type** field on the **General** tab. Alternatively, you can edit the CustomSettings.config file. For more information, see Configuring Business Central Server.

Client Configuration

In the relevant configuration file, find the **ClientServicesCredentialType** parameter and change the value to one of the options listed earlier.

For the Business Central Web client users, you must modify the *navsettings.json* for the Business Central Web Server. The navsettings.json file is a Java Script Object Notification file type that is similar to files that use the XML file format. The file is stored in the physical path of the web server instance, which is by default is *c:\inetpub\wwwroot\BC160*. For more information, see Settings in the navsettings.json.

For each Dynamics NAV Client connected to Business Central user, you must modify the ClientUserSettings.config file. The default location for this file is **C:\Users\<username>\AppData\Roaming\Microsoft\Microsoft Dynamics NAV\130**, where *<username>* is the name of the user. For more information, see Configuring the Microsoft Dynamics NAV Windows Client in the ITPro content for Microsoft Dynamics NAV 2018.

Security Certificates

With UserName, NavUserPassword, and AccessControlService credential types require that you install and configure security certificates on components. For more information, see Using Security Certificates with Business Central On-Premises

See Also

Understanding Users, Profiles, and Role Centers
Configuring Business Central Server

Configuring Business Central Server

2/17/2021 • 65 minutes to read • Edit Online

When you run Business Central Setup and install Business Central Server, you provide information that is then used as the configuration for the default Business Central Server instance. This information is stored in a configuration file for the server instance called CustomSetting.config. The default location of the CustomSettings.config file is *C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service*.

After you install Business Central Server, you can change any of the settings, including other settings that weren't available to you in Setup.

> **NOTE**
> Each Business Central Server instance has its own CustomSettings.config file.

Configuring Business Central Server in Setup

You configure the default instance of Business Central Server by running Business Central Setup and selecting one of the following options:

- Demo Option
- Server Option
- Developer Option
- Customize > Server

After you specify an installation option or customize your component list, the **Specify parameters** pane is displayed in Setup. The list of parameters that you see depends on which components you've selected for configuration. Setup provides a short description for each parameter.

Configuring Business Central Server After Installation

After you install Business Central Server, you can change the configuration settings in the CustomSettings.config file of a Business Central Server instance in the following ways:

- Using the Business Central Server Administration tool.

  For more information, see Settings in the Business Central Server Administration Tool and Business Central Server Administration Tool.

- Using the Set-NAVServerConfiguration cmdlet that is available in the Business Central Administration Shell.

  For more information, see Using Administration Shell Cmdlets to Modify Settings.

- By directly editing CustomSettings.config using a text editor.

  We recommend that you don't directly edit the configuration file. If you make any errors in typing, the server instance might not start.

Restarting Business Central Server after modifications

If you use the Business Central Server Administration tool or modify the CustomSettings.config file directly, you must restart the Business Central Server instance before any changes can take effect.

The Set-NAVServerConfiguration cmdlet doesn't always require restarting the server instance. It depends on the configuration setting that you change. There are several settings that are *dynamically updatable*. *Dynamically updatable* means that a server instance restart isn't necessarily required after modification. For more information, see Modifying dynamically updatable settings.

These settings are indicated by the text **Dynamically Updatable: Yes** in the tables that follow,

Business Central Server Instance Settings

This section describes all the configuration settings for a Business Central Server instance. The settings are grouped according to the tabs under which they appear in the Business Central Server Administration tool.

- The **Setting** column displays the name of the setting as it appears in the Business Central Server Administration tool.
- The **Key Name** column displays the name of the setting as it appears in the CustomSettings.config file. It's

also the name for the setting when you run the Set-NAVServerConfiguration cmdlet.

General Settings

The following table describes fields on the **General** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
AL Function Logging Threshold - Application Insights	ALLongRunningFunctionTracingThresholdForApplicationInsights	Specifies the amount of time (in milliseconds) that an AL function can run before a warning event is recorded in the partner's Application Insights resource trace log. If you don't want a threshold, set the value to -1. To collect this telemetry data, the **Application Insights Instrumentation Key** setting must be configured. For information about analyzing this telemetry, see Analyzing Long Running AL Methods Telemetry. Default: -1 Dynamically Updatable: Yes **APPLIES TO:** Business Central 2020 release wave 2 (version 17.1) and later.
Application Insights Instrumentation Key	ApplicationInsightsInstrumentationKey	Specifies the instrumentation key of the Microsoft Azure Application Insights resource to use for gathering and analyzing telemetry data emitted by the server instance. When this setting is configured, the server instance will send telemetry data to Application Insights for analysis and presentation. This setting only applies to a server instance that is configured as a single-tenant instance (**Multitenant** setting is disabled). For a multitenant server instance, this setting is ignored. You set the Application Insights instrumentation key on a per-tenant basis, when you mount tenants. For more information, see Enable Sending Telemetry to Application Insights. Default: empty Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Build Restriction	ClientBuildRestriction	Specifies what happens when a Business Central client tries to connect to the Business Central Server instance when the client is running a different build version of Business Central than the server instance. Values: **AlwaysConnect** **WarnClient** Before connecting the client to the server instance, a message appears that informs the user that the build versions for the client and server instance are different. The user can choose to continue or cancel the connection. **DoNotAllow** A message appears that informs the user that the client and server instance build versions are different. The client doesn't connect to the server instance. **Note:** With the Business Central Web client and Business Central tablet client, this setting compares the build version of the Business Central Web Server on IIS with the Business Central Server instance. It controls the connection between Business Central Web Server and the server instance. Default: WarnClient Dynamically Updatable: No
Certificate Thumbprint	ServicesCertificateThumbprint	If you use security certificates to protect communications between Business Central Server and client services or web services over an open or wide-area network, you must provide the certificate thumbprint to Business Central Server by updating this setting. For more information, see Using Security Certificates. Default: Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Compile and Load Business Application	CompileBusinessApplicationAtStartup	Specifies whether the server instance compiles business application assemblies and loads them to cache memory when it starts. The assemblies are then retrieved from memory when requested by a Business Central client. This setting reduces the time it takes to load application objects the first time the client requests them after the server instance has started. However, it will also slightly increase the memory usage by the server instance. When the server instance starts for the first time, it compiles the business application assemblies and loads them to the cache memory of the computer. The assemblies, along with metadata such object timestamp information, are also stored to a temporary folder on the computer's file system. When the server instance is restarted, it compares the assemblies stored in memory with corresponding objects in the connected database. An assembly will be reused if the following conditions are met: - The connected database is the same as before, based on the *databasemagic* field in the property table. - The object time stamp that is recorded on the compiled assembly matches the object timestamp in metadata of the connected database. If the conditions aren't met for an assembly or an assembly for an object in the database isn't found in the memory, then a new assembly is built and stored for reuse to cache memory and the file system of the server instance compute for reuse. If you disable this setting, individual assemblies will be compiled on-demand as application objects are requested by the Business Central client. The compiled assemblies won't be reused on later server instance restarts. Notes: • This setting doesn't apply to query objects. • Assembly compilation happens asynchronously. • On average, all application objects will be loaded within the first few minutes that the server instance operates. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Credential Type	ClientServicesCredentialType	Specifies the authentication mechanism for Business Central users of the Business Central Server instance. The options are **Windows**, **Username**, **NavUserPassword**, **AccessControlService**, and **None**. For more information, see Authentication and User Credential Types. If you choose **AccessControlService**, you must specify a federation metadata location for use with Azure AD. If you choose **NavUserPassword** and specify a token signing key, you can use both NavUserPassword and AccessControlService. Notes: • **None** is for internal use on system sessions and typically shouldn't be used. If you choose **None**, then the Business Central Server instance can't start. • **ExchangeIdentity** and **TaskScheduler** are for internal use only, and shouldn't be used. Default: Windows Dynamically Updatable: No
Data Cache Size	DataCacheSize	The contextual size of the data cache. The value must be in the range 1-20. Default: 9 Dynamically Updatable: Yes
Default Client	DefaultClient	Specifies the client type that is used to generate URLs when the client type is set to Default. The options are **Conversational**, **Windows**, **Web**, **SOAP**, and **OData**. Default: Conversational Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Default Language	DefaultLanguage	Specifies which of the installed Business Central languages on the server instance will be used as the default language in the clients. Set the value to a valid language culture name, such en-US or da-DK. In the web and tablet clients, this setting determines the language used if the browser's language setting doesn't match any installed language or a language in the **Supported Languages** setting. In the Business Central Windows client, this value is used if the language setting of the computer doesn't have a match. If there are application-specific configuration settings, this setting will be overridden by the default language setting that is specified in application-specific configuration file. For more information, see Set-NAVServerAppConfiguration cmdlet. Default: en-US Dynamically Updatable: No
Diagnostic Trace Level	TraceLevel	Specifies the lowest severity level of custom telemetry events to be emitted and recorded in the event log for the Business Central Server instance. The events include system telemetry trace events and custom telemetry events. Telemetry events have IDs from 700-706. The setting has the following values, which correspond to the event severity levels (listed from highest to lowest level): **Critical**, **Error**, **Warning**, **Normal** (this value corresponds to the **Information** level), **Verbose**, and **Off**. You use this setting to filter out lower-level events from the log. For example, if you set this setting to **Error**, only **Error** and **Critical** events will be logged. Set to **Off** if you don't want to record telemetry events. When set to **Off**, events aren't emitted. **Note:** Telemetry trace events are recorded in the Business Central Server channel logs, which you can see in Event Viewer, under **Applications and Services Logs** > **Microsoft** > **DynamicsNAV** > **Common** > **Admin**. Default: Normal Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
Diagnostic Trace Level for External Proxies	ExternalTraceLevel	Specifies the lowest severity level of telemetry events from external proxies that you want the Business Central Server instance to emit if an error related to the external system occurs on the server instance. This setting pertains to systems and components that Business Central integrates with, like Dynamics 365 Sales (CRM/Xrm). The server instance listens for event traces from the external proxy. If an error occurs on the server instance, it will emit the last 10 telemetry trace events from the external proxy. The trace events can then be recorded in the Windows event log or picked up by other event trace collection tools. The setting has the following values, which correspond to the event severity levels (listed from highest to lowest level): **Critical**, **Error**, **Warning**, **Information**, **Verbose**, and **Off**. Events that have a lower severity level than the set value won't be emitted. For example, if you set this setting to **Error**, only **Error** and **Critical** events will be emitted. Set to **Off** if you don't want to emit any of these events. Default: Error Dynamically Updatable: Yes
Disable Token-Signing Certificate Validation	DisableTokenSigningCertificateValidation	Specifies whether to enable or disable the validation of the token-signing certificate used by Active Directory Federation Services (AD FS). If the check box is cleared (or the value set to `false`), the validation is enabled. If the check box is selected (or the value is set to `true`), then validation is disabled. Disable token signing certificate validation when configuring Azure Active Directory authentication with single sign-on. Default: Checkbox cleared; set to `false`. Dynamically Updatable: No
Enable AL Function Timing	ALFunctionTimingEnabled	Specifies whether AL function timing is enabled. When enabled, data about AL extension performance will be collected and can be viewed in Application Insights and page inspection. For more information, see Analyzing Long Running AL Methods Telemetry and Inspecting and Troubleshooting Pages. Default: Enables Dynamically Updatable: Yes **APPLIES TO:** Business Central 2020 release wave 2 (version 17.1) and later.
Enable Certificate Validation	ServicesCertificateValidationEnabled	Specifies whether validation is done on the security certificate. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Enable Debugging	EnableDebugging	Specifies whether the Business Central Server instance starts with debugging enabled. If this option is enabled, the following occurs: When the client first connects, all C# files for the application are generated. C# files persist between Business Central Server restarts. Application objects are compiled with debug information. Default: Not enabled Dynamically Updatable: No
Enable Event Logging to Windows Application Log	EnableApplicationChannelLog	Specifies whether to record admin and operational type events in the **Windows Application** log. This setting will record errors, warnings, and information messages that occur on server instance. The events are stored on the computer that is running Business Central Server. Because Business Central Server instance events are always logged to the **Application and Services Logs**, you can disable logging Business Central Server instance events in the Windows Application log and not lose any data. For more information, see Monitoring Business Central Server Events Using Event Viewer and Disable Logging Events to the Windows Application Log. **Important:** If you 're using System Center Operations Manager to monitor Business Central Server instances, don't disable logging to the Windows Application log. If you do, monitoring won't work. Default: Enabled Dynamically Updatable: No
Enable File Access by AL Functions	EnableALServerFileAccess	Specifies whether AL functions of the file data type can access files on the Business Central Server computer. Default: Enabled Dynamically Updatable: No
Enable Full AL Function Tracing	EnableFullALFunctionTracing	Specifies whether full AL function tracing is enabled on Event Tracing for Windows (ETW) sessions. When this setting is enabled, all AL function calls and statements are traced. When this setting is disabled, only root AL function calls are traced. Statements and functions that are called from a function aren't traced. For more information, see Monitoring Business Central Server Events . Default: Not enabled Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
Enable Incremental Company Deletion	UseIncrementalCompanyDelete	Specifies whether to delete companies incrementally. If enabled, when you delete a company, the company record is deleted from the database immediately. But the company data that is stored in the SQL tables will be deleted later by a system task in task scheduler. You can override this setting when using the Remove-NAVCompany cmdlet by setting the -ForceImmediateDataDeletion parameter. Default: Not enabled Dynamically Updatable: Yes
Enable Partial Records	EnablePartialRecords	Specifies whether records can be partially loaded with only some fields. Other fields are loaded as needed, or in other words, just-in-time loaded. For more information, see Using Partial Records. Default: Enabled Dynamically Updatable: No
AllowSessionWhileSyncAndDataUpgrade	AllowSessionWhileSyncAndDataUpgrade	Specifies whether new client sessions can be created while the tenant's state is **OperationalWithSyncPending**. The **OperationalWithSyncPending** state occurs when changes have been made to application tables, but the changes haven't been synchronized with the tenant. In this state, unless you enable client sessions, clients trying to connect with will get an error message similar to: **The tenant 'tenantID' is not accessible**. Default: Not enabled Dynamically Updatable: Yes
Lockout Sign-In Attempts Count	LockoutPolicyFailedAuthenticationCount	Specifies the number of failed sign-in attempts on a user account (within the time window set by the **Lockout Failed Sign-In Attempts Window** setting) at which the user account is disabled. Default: 0 Dynamically Updatable: No
Lockout Failed Sign-In Attempts Window	LockoutPolicyFailedAuthenticationWindow	Specifies time window, in seconds, during which consecutive failed authentication attempts are counted. This setting works in conjunction with the **Account Lockout Max. Sign-In Attempts** setting. When the number of failed sign-in attempts by a user hits the value of the **Account Lockout Max. Sign-In Attempts** setting within this time window, the user account is disabled. Default: 0 Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Max Concurrent Calls	MaxConcurrentCalls	The maximum number of concurrent client calls that can be active on this server instance. Range: 1 - 2,147,483,647 You can also use **MaxValue** as a value to indicate no limit. Default: 40 Dynamically Updatable: No
Max Data Rows Allowed to Send to Excel	MaxRowsToExportToExcel	Specifies the maximum number of rows that can be included in an Excel document that is generated from a list type page in the client. If you don't want to have a limit on rows, set the value to **MaxValue**. **Note:** This setting only pertains to list type pages in the client. For other pages types, like cards, the limit on rows is configured in the client. Default: MaxValue Dynamically Updatable: Yes
Maximum Stream Read Size	MaxStreamReadSize	Specifies the maximum number of bytes that can be read from a stream (InStream object) in a single AL read operation, such a READ or InStream.READTEXT function call. This setting pertains to UTF-8 and UTF-16 text encoding; not MS-DOS encoding. Default: 1000000 Dynamically Updatable: Yes
Multitenant	Multitenant	Specifies if the Business Central Server instance can be used in a multitenant environment. Tenant databases can only be mounted on the Business Central Server instance if this setting is selected. For more information, see Multitenant Deployment Architecture. Default: Not enabled Dynamically Updatable: No
Network Protocol	NetworkProtocol	Specifies the network protocol for accessing the database. Valid values: Default, Named, Sockets, MultiProtocol Default: Default Dynamically Updatable: No
Services Default Company	ServicesDefaultCompany	Specifies the Business Central company that the client services, OData web services, and NAS services use as the default company. If your Business Central database contains only one company, leave the setting blank. Default: Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Services Default Time Zone	ServicesDefaultTimeZone	Specifies the time zone in which web service and NAS services calls are run. Values: **UTC** All business logic for web services and NAS services on the server instance runs in Coordinated Universal Time (UTC). **Server Time Zone** Services use the time zone of the computer that is running Business Central Server. **ID of any time zone recognized by the current version of Windows** Specifies any Windows time zone as defined in the system registry under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones. For example, Romance Standard Time. Default: UTC Dynamically Updatable: No
Services Language	ServicesLanguage	Specifies the global language version to use for text strings with SOAP and OData web services. The value must be valid culture name for a language that is available for the Microsoft Dynamics NAV solution, such as en-US and da-DK. Default: en-US Dynamically Updatable: No
Services Option Text Source	ServicesOptionFormat	Specifies the source of the text strings to use for the option values of an option data type field. Values: **OptionString** Uses the text strings that are specified by the OptionString Property of a field. **OptionCaption** Uses the text strings that are specified by the OptionCaption Property of a field. Default: OptionCaption Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Session Event Table Retain Interval	SessionEventTableRetainInterval	Specifies the amount of time that sessions are stored in the **Session Event** table (ID 2000000111) before they are deleted. Sessions that exceed the time limit are first deleted the next time a purge is run on the **Session Event** table, as determined, in part, by the **SessionEventTablePurgeLookupPeriod** setting. The default is once a day. Time interval format: [dd.]hh:mm:ss[.ff] Where: dd: days hh: hours mm: minutes ss: seconds ff: hundredths of a second Default: 90.00:00:00 (90 days) Dynamically Updatable: No
Non-Interactive Sessions Log Retain Interval	NonInteractiveSessionsLogRetainInterval	Specifies the amount of time that background and web service sessions are stored in the **Session Event** table (ID 2000000111) before they are deleted. Sessions that exceed the time limit are first deleted the next time a purge is run on the **Session Event** table, as determined, in part, by the **SessionEventTablePurgeLookupPeriod** setting. The default is once a day. Time interval format: [dd.]hh:mm:ss[.ff] Where: dd: days hh: hours mm: minutes ss: seconds ff: hundredths of a second Default: 5.00:00:00 (5 days) Dynamically Updatable: No
not available	SessionEventTablePurgeLookupPeriod	Specifies how much time must lapse after the **Session Event** table is purged before it can be purged again. Purging is triggered when a new session of any type is created and the time that has lapsed since last purge exceeds the **SessionEventTablePurgeLookupPeriod** value. When the **Session Event** table is purged, all sessions older than the retain interval (as set by the **Session Event Table Retain Interval** or **Non-Interactive Sessions Log Retain Interval** settings) are deleted from the table. Time interval format: [dd.]hh:mm:ss[.ff] Where: dd: days hh: hours mm: minutes ss: seconds ff: hundredths of a second Default: 1.00:00:00 (1 day) Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Supported Languages	SupportedLanguages	Specifies which of the installed Business Central languages on the server instance will be available for use in the clients. If you don't specify a language, then all installed languages will be available. In the client, users can switch among the supported languages. The setting's value is a semicolon-separated list that contains the language culture names for each language. For example, if you want client users to be able to choose among da-DK, en-US, and en-CA, set the value to da-DK;en-US;en-CA. If you specify any languages in this setting, then you must include the language that you specified in the Default Language setting. If there are application-specific configuration settings, this setting will be overridden by the supported language setting that is specified in application-specific configuration file. For more information, see Set-NAVServerAppConfiguration cmdlet. Default: Dynamically Updatable: No
Token Signing Validation Mode	TokenSigningCertificateValidationMode	Specifies which certificate validation mode to use for token signing validation. This setting is applicable only if the **Enable Certificate Validation** setting is selected. **IssuerNameValidation** validates tokens by verifying the issuer name (tenant) only. **PeerOrChainValidation** validates tokens by verifying that the certificate is either in the Trusted People store or is part of a chain trust to a certification authority in the Trusted Root store. Default: IssuerNameValidation Dynamically Updatable: No
UI Elements Removal	UIElementRemovalOption	Specifies whether UI elements are hidden when the related object isn't accessible according to the license or according to user permissions or both. For more information, see Hiding UI Elements. Default: LicenseFileandUserPermissions Dynamically Updatable: No
Use NTLM Authentication	ServicesUseNTLMAuthentication	Specifies whether NTLM authentication is enabled for web services. To require Kerberos authentication, disable this option. Default: Not enabled Dynamically Updatable: No
not available	XmlMetadataCacheSize	For internal use only. Default: 500

Database Settings

The following table describes fields on the **Database** tab in the Business Central Server Administration tool.

NOTE

If the Business Central Server instance is configured as a multitenant server instance, then except for the **Database Name**, **Database Instance**, and **Database Server** settings, the settings apply to both the application database and the tenant database.

SETTING	KEY NAME	DESCRIPTION
Database Instance	DatabaseInstance	The name of the SQL Server database instance to connect to. If the value is a null string (""), Business Central Server instance connects to the default database instance of SQL Server. If the Business Central Server instance is configured as a multitenant server instance, then this setting specifies the SQL Server database instance that hosts the application database. Default: NAVDEMO Dynamically Updatable: No
Database Name	DatabaseName	The name of the Business Central database in SQL Server. If the Business Central Server instance is configured as multi-tenant server instance, then this setting specifies the application database. Default: "Demo Database BC (14-0)" Dynamically Updatable: No
Database Server	DatabaseServer	A valid network name for the computer that is running SQL Server. If the Business Central Server instance is configured as multi-tenant server instance, then this setting specifies the computer that hosts the application database. Default: The computer that you selected in Business Central Setup. Dynamically Updatable: No
Disable SmartSQL	DisableSmartSql	Specifies whether the SmartSQL performance optimization feature is disabled. - If the check box is selected, SmartSQL is disabled. - If the check box is cleared, SmartSQL is enabled. When SmartSQL is enabled, Business Central Server converts FIND calls and FlowField calculations into a single SQL statement. This can improve performance when running pages that contain FlowFields. However, it can be helpful to disable SmartSQL when troubleshooting database queries because statements are separated into more discrete statements. For more information, see Troubleshooting: Long Running SQL Queries Involving FlowFields by Disabling SmartSQL. Default: SmartSQL performance optimization is enabled (check box is cleared) Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
Disable SQL Query Hint FORCE ORDER	DisableQueryHintForceOrder	Specifies whether the FORCE ORDER Query Hint is used in queries. FORCE ORDER instructs the query optimizer to preserve the join order that is indicated by the query syntax. If you clear the check box (`false`), the FORCE ORDER is used in queries. For more information, see Configuring Query Hints for Optimizing SQL Server Performance. Default: FORCE ORDER is disabled (check box is selected) Dynamically Updatable: Yes
Disable SQL Query Hint LOOP JOIN	DisablQueryHintLoopJoin	Specifies whether the LOOP JOIN Query Hint is used in queries. LOOP JOIN instructs the query optimizer to use LOOP JOIN for all join operations in the whole query. If you clear the check box (`false`), the LOOP JOIN hint is used in queries. For more information, see Configuring Query Hints for Optimizing SQL Server Performance. Default: LOOP JOIN is disabled (check box is selected) Dynamically Updatable: Yes
Disable SQL Query OPTIMIZE FOR UNKNOWN	DisableQueryHintOptimizeForUnknown	Specifies whether the OPTIMIZE FOR UNKNOWN Query Hint is used in queries. OPTIMIZE FOR UNKNOWN instructs the query optimizer to use statistical data instead of the initial values for all local variables when the query is compiled and optimized, including parameters created with forced parameterization. If you clear the check box (`false`), the OPTIMIZE FOR UNKNOWN hint is used in queries. For more information, see Configuring Query Hints for Optimizing SQL Server Performance. Default: OPTIMIZE FOR UNKNOWN is enabled (check box is cleared) Dynamically Updatable: Yes
Enable Buffered Insert	BufferedInsertEnabled	Specifies whether to buffer rows that are being inserted into a SQL Server database table. When this parameter is enabled, up to 5 rows will be buffered in the table queue before they are inserted into the table. To optimize performance in a production environment, you should enable this parameter. In a test environment, you can disable this parameter to help debug failures that occur when you insert rows in an SQL database table. For more information, see Bulk Inserts. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Enable Encryption on SQL Server Connections	EnableSqlConnectionEncryption	Specifies whether the SQL connect string should request encryption when connecting to SQL Server services. Default: Not enabled Dynamically Updatable: No
Enable SQL Read-Only Replica Support	EnableSqlReadOnlyReplicaSupport	Specifies whether the server instance is allowed to connect to an SQL Server secondary read-only replica of an Always On availability group. Using read-only replicas helps balance workloads and improve performance. To take advantage of this setting, the database must be set up with read-scale out support. For more information, see Using Read Scale-Out for Better Performance Default: Not enabled Dynamically Updatable: No
Enable Trust of SQL Server Certificate	TrustSQLServerCertificate	Specifies whether Business Central Server should trust the SQL Server certificate. Default: Not enabled Dynamically Updatable: No
Search Timeout	SearchTimeout	Specifies the time (in seconds) that a search operation on lists in the client will continue until it's stopped. When the limit is reached, the following message displays in the client: **Searching for rows is taking too long. Try to search or filter using different criteria.** Time format: hh:mm:ss Default: 00:00:10 Dynamically Updatable: Yes
SQL Bulk Import Batch Size	SqlBulkImportBatchSize	Specifies how many SQL memory chunks that a data import must be distributed across. Lowering the value increases the number of network transfers and decreases performance, but also lowers the amount of memory that the server instance consumes. If the database is on SQL Server 2016 or later, a low value can lead to large data files. If you don't want to use batching, specify 0. Default: 448 Dynamically Updatable: No
SQL Command Timeout	SqlCommandTimeout	Specifies the contextual time-out for a SQL command. Default: 0:30:00 Dynamically Updatable: No
SQL Connection Idle Timeout	SqlConnectionIdleTimeout	Specifies the time that a SQL connection can remain idle before being closed. The value has the format hh:mm:ss. Default: 00:05:00 Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
SQL Connection Timeout	SqlConnectionTimeout	Specifies the time to wait while trying to connect to the database before stopping the attempt and generating an error. This setting also applies to begin, rollback, and commit of transactions. The value has the format hh:mm:ss. Default: 00:01:30 Dynamically Updatable: Yes
SQL Management Command Timeout[1]	SqlManagementCommandTimeout	Specifies the timeout for SQL commands related to management operations, for example schema synchronization and company management operations. This setting enables you to set a different timeout for management operations than for normal, day-to-day, runtime operations like a Record.FINDSET, which are controlled by the **SQLCommand Timeout** setting. When a negative value is specified for this setting, the **SQL Command Timeout** will be used. The value has the format hh:mm:ss. Default: -1 Dynamically Updatable: Yes
Enable SQL Parameters by Ordinal	SqlParametersByOrdinal	Specifies whether parameters in SQL statements are referenced by their ordinal number. Enabling this setting improves performance when using buffered inserts. Default: Enabled Dynamically Updatable: No
SQL Query Logging Threshold	SqlLongRunningThreshold	Specifies the amount of time (in milliseconds) that an SQL query can run before a warning event is recorded in the application log for the server instance. If this threshold is exceeded, the following event is logged: Action completed successfully, but it took longer than the given threshold. Default: 1000 Dynamically Updatable: Yes

Client Services Settings

The following table describes fields on the **Client Services** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION

SETTING	KEY NAME	DESCRIPTION
Allowed File Types	ClientServicesAllowedFileTypes	Specifies the file types that can be stored by the server when requested by the client. The value is a semicolon-separated list of the file name extensions. The server won't store other file types. Example values: • Blank or empty string (`""`): The setting is disabled. File types will be limited based on **Prohibited File Types** setting instead. • Asterisk (`*`): Specifies that all file types are allowed. • List of file type extensions separated by semi-colons, for example `txt;xml;pdf` : Specifies that only .txt, .xml, and .pdf file types can be stored. Trailing semicolons are ignored. Default: blank Dynamically Updatable: Yes
Chunk Size	ClientServicesChunkSize	The default size for a chunk of data that is transferred between Business Central Server and the Dynamics NAV Client connected to Business Central or Business Central Web Server, in kilobytes. The range of values is from 4 to 80. Default: 28 Dynamically Updatable: No
Compression Threshold	ClientServicesCompressionThreshold	The threshold in memory consumption at which Business Central Server starts compressing datasets, in kilobytes. Default: 64 Dynamically Updatable: No
Enable Client Services	ClientServicesEnabled	Specifies whether client services are enabled for this server instance. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Exchange Auth. Metadata Location	ExchangeAuthenticationMetadataLocation	Specifies the URLs for the Microsoft Exchange authentication metadata document of the services or authorities that are trusted to sign Exchange identity tokens. This setting is used for setting up the Office Add-Ins for Outlook. For more information about the Office Add-ins, see Setting Up the Office Add-Ins for Outlook Integration The value is URL that is used to confirm the identity of the signing authority when using Exchange Authentication. The URL is compared to the Exchange authentication metadata document URL in the Exchange identity token. The scheme and host part of the two URLs must match to pass authentication. Paths in the URLs require only a partial match. With a multitenant server instance, the **Exchange Auth. Metadata Location** setting (if any) on the tenant will overrule the value of this setting. Value: - One or more valid URLs. A URL must include the scheme, such as https:// or https://, and the host name. - Separate multiple URLs with a comma. - Wildcards (*) in URLs are supported. Default: https://outlook.office365.com/ Dynamically Updatable: No
Idle Client Timeout	ClientServicesIdleClientTimeout	The interval of time that a Business Central Server client session can remain inactive before the session is dropped. Time interval format: [dd.]hh:mm:ss[.ff] Where: dd: days hh: hours mm: minutes ss: seconds ff: hundredths of a second Set **Idle Client Timeout** to equal or lower than the **Keep Alive Interval**, to enable **Idle Client Timeout**. You can also use **MaxValue** as a value to indicate no time-out. Default: MaxValue Dynamically Updatable: Yes
Keep Alive Interval	ClientServicesKeepAliveInterval	Specifies the time interval between keep-alive messages that are sent from the Dynamics NAV Client connected to Business Central to the server instance. This setting is used to keep inactive sessions alive until the time that is specified by the **Idle Client Timeout** setting expires. You should use a time interval that is less than the **Idle Client Timeout** setting, to hold the session constantly alive. For more information, see Understanding Session Timeouts. Time interval format: [dd.]hh:mm:ss[.ff] Default: 120 Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Max Concurrent Connections	ClientServicesMaxConcurrentConnections	Specifies the maximum number of concurrent client connections that the current Business Central Server instance accepts. You can use **MaxValue** as a value to indicate no limit. Default: 500 Dynamically Updatable: No
Max Items in Object Graph	ClientServicesMaxItemsInObjectGraph	The maximum number of objects to serialize or deserialize. Default: 512 Dynamically Updatable: No
Max Number of Orphaned Connections	ClientServicesMaxNumberOfOrphanedConnections	Specifies the maximum number of orphaned connections to be kept alive at the same time for the time that is specified by **ReconnectPeriod**. A connection is orphaned when the client is involuntarily disconnected from Business Central Server. You can also use **MaxValue** as a value to indicate no limit. Default: 20 Dynamically Updatable: No
Max Upload Size	ClientServicesMaxUploadSize	The maximum size of files that can be uploaded to or downloaded from Business Central Server, in megabytes. Use this setting to avoid out-of-memory errors. Default: 150 Dynamically Updatable: No
Operation Timeout	ClientServicesOperationTimeout	The maximum time that Business Central Server can take to return a call from the client. Time span format: [dd.]hh:mm:ss[.ff] Where: dd: days hh: hours mm: minutes ss: seconds ff: hundredths of a second You can also use **MaxValue** as a value to indicate no time-out. Default: MaxValue Dynamically Updatable: No
Port	ClientServicesPort	The listening HTTP port for client services. Valid range: 1 - 65535 Default: 7046 Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Prohibited File Types	ClientServicesProhibitedFileTypes	Specifies the file types that can't be stored by the server when requested by the client. The value is a semicolon-separated list of the file name extensions. This setting is ignored if the **Allowed File Types** setting has a value. Example values: • Asterisk (`*`): All file types are prohibited. • Blank or empty string (`""`): The default value is used. • Whitespace string (`" "`): All file types are allowed. • List of file types separated by a semicolon, for example `txt;xml;pdf` : Prohibits the file types txt, xml and pdf. Trailing semi-colons will be ignored. Default: ade;adp;app;asp;bas;bat;chm;cmd;com;cpl; csh;exe;fxp;gadget;hlp;hta;inf;ins;isp;its;js;jse; ksh;lnk;mad;maf;mag;mam;maq;mar;mas;mat;mau; mav;maw;mda;mdb;mde;mdt;mdw;mdz;msc;msi; msp;mst;ops;pcd;pif;prf;prg;pst;reg;scf;scr; sct;shb;shs;url;vb;vbe;vbs;vsmacros;vss;vst;vsw;ws;wsc;wsf;wsh Dynamically Updatable: Yes
Protection Level	ClientServicesProtectionLevel	Specifies the security services for protecting the data stream between clients and Business Central Server. All Dynamics NAV Client connected to Business Central clients connecting to the Business Central Server instance must have the same ProtectionLevel value in their ClientUserSettings.config files. For more information, see Configuring the Windows Client in the Dev and IT Pro Help for Microsoft Dynamics NAV 2018. For background information about transport security, see Understanding Protection Level (links to MSDN Library). Values: EncryptAndSign, Sign, None Default: EncryptAndSign Dynamically Updatable: No
Reconnect Period	ClientServicesReconnectPeriod	The time during which a client can reconnect to a running instance of Business Central Server. Time span format: [dd.]hh:mm:ss[.ff] Where: dd: days hh: hours mm: minutes ss: seconds ff: hundredths of a second You can also use **MaxValue** as a value to indicate no time limit. Default: 00:10:00 Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Token Signing Key	ClientServicesTokenSigningKey	Specifies the signing information that you obtain from the Microsoft Azure portal. The parameter value is a 256-bit symmetric token signing key for use with Azure Access Control service (ACS). This parameter is relevant only when **Credential Type**, on the **General** tab, is set to **AccessControlService**. Default: EncryptAndSign Dynamically Updatable: No
Use the Simplified Filters	UseSimplifiedFilters	Specifies how the search on list pages behaves for plain text search filters. Pain text search filters don't use search symbols like @ or *. If you enable this setting, the search uses a case sensitive and accent sensitive search to find fields that start with the provided filter text. For example, the search on **man** returns all records that include a field that starts with *man* (lowercase m), and the search on **Man** returns all records that include a field that starts with *Man* (uppercase M). Notice that you can get the same results by entering **man*** and **Man*** respectively. If the setting is disabled (which is default), the search on **man** and **Man** return the same results, which are all records that include fields that contain the text *man*, regardless of the case. For more information about the search, see Sorting, Searching, and Filtering Lists. Default: Not enabled Dynamically Updatable: No
Web Client Base URL	PublicWebBaseUrl	Specifies the root of the URLs that are used to open hyperlinks to pages and reports in the Business Central Web client. For example, you can change the value if you want to change the externally facing endpoint. The base URL must have the following syntax: `http[s]://[hostname]:[port]/[webserverinstance]` This field maps to the `PublicWebBaseUrl` setting in the CustomSettings.config file for the Business Central Server instance. Default: The URL of the Web client Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Windows Client Base URL	PublicWinBaseUrl	Specifies the root of the URLs that are used to open hyperlinks to pages and reports in the Dynamics NAV Client connected to Business Central. For example, you can change the value if you want to change the externally facing endpoint. The base URL must have the following syntax: `DynamicsNAV://[hostname]:[port]/[instance]/` This field maps to the `PublicWinBaseUrl` setting in the CustomSettings.config file for the Business Central Server instance. Default: The URL of the Windows client Dynamically Updatable: No

SOAP Services Settings

The following table describes fields on the **SOAP Services** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Enable SOAP Services	SOAPServicesEnabled	Specifies whether SOAP web services are enabled for this server instance. Default: Not enabled Dynamically Updatable: No
Enable SSL	SOAPServicesSSLEnabled	Specifies whether SSL (https) is enabled for the SOAP web service port. For more information, see Using Security Certificates with Business Central On-Premises. Default: Not enabled Dynamically Updatable: No
Max Connections	SOAPMaxConnections	Specifies the maximum number of simultaneous SOAP requests on the server instance (for all tenants). When the limit is exceeded, an error occurs. If you don'tt want a limit, set the value **0**. Default: 0 Dynamically Updatable: Yes
Max Connections Per Tenant	SOAPMaxConnectionsPerTenant	Specifies the maximum number of simultaneous SOAP requests per tenant. When the limit is exceeded, an error occurs. If you do not want a limit, set the value **0**. Default: 0 Dynamically Updatable: Yes
Max Concurrent Requests[1]	SOAPMaxConcurrentRequests	Specifies the maximum number of SOAP requests per tenant that the server instance can process at the same time. If you don't want a limit, set the value `0`. Default: 5 Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
Max Queued Requests[1]	SOAPRequestQueueSize	Specifies the maximum number of pending SOAP connections per tenant waiting to be processed. When the limit is exceeded, a 429 (Too Many Requests) error occurs. If you don't want a limit, set the value `0`. Default: 95 Dynamically Updatable: Yes
Max Message Size	SOAPServicesMaxMsgSize	Specifies the maximum permitted size of a SOAP web services request, in kilobytes. When a request exceeds this limit, a 413: Request Entity Too Large error occurs. **Important:** This setting also pertains to OData web services. Default: 65536 Dynamically Updatable: No
Port	SOAPServicesPort	The listening HTTP port for SOAP web services. Valid range: 1 - 65535 Default: 7047 Dynamically Updatable: No
SOAP Base URL	PublicSOAPBaseUrl	Specifies the root of the URLs that are used to access SOAP web services. For example, you can change the value if you want to change the externally facing endpoint. The base URL must have the following syntax: http[s]://*hostname.port*/*instance*/WS/ This field maps to the `PublicSOAPBaseUrl` setting in the CustomSettings.config file for the Business Central Server instance. Default: The SOAP URL for the server instance Dynamically Updatable: No
Timeout	SOAPServicesOperationTimeout	Specifies the maximum amount of time that the server can allocate to a single SOAP request. When the limit is exceeded, a timeout error occurs. Time span format: hh:mm:ss If you do not want a timeout, set the value to **MaxValue**. Default: 00:10:00 Dynamically Updatable: Yes

OData Services Settings

The following table describes fields on the **OData Services** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Enable Add-in Annotations	ODataEnableExcelAddInAnnotations	Specifies whether Excel add-in annotations should be provided in OData metadata. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Enable API Services	ApiServicesEnabled	Specifies whether API web services are enabled for this server instance. Default: Not enabled Dynamically Updatable: No
Enable OData Services	ODataServicesEnabled	Specifies whether OData web services are enabled for this Business Central Server instance. Default: Enabled Dynamically Updatable: No
Enable SSL	ODataServicesSSLEnabled	Specifies whether SSL (https) is enabled for the OData web service port. For more information, see Using Security Certificates with Business Central On-Premises. Default: Not enabled Dynamically Updatable: No
Enable V3 Endpoint	ODataServicesV3EndpointEnabled	Specifies whether the ODataV3 service endpoint will be enabled. Default: Enabled Dynamically Updatable: No
Enable V4 Endpoint	ODataServicesV4EndpointEnabled	Specifies whether the ODataV4 service endpoint will be enabled. Default: Enabled Dynamically Updatable: No
Max Connections	ODataMaxConnections	Specifies the maximum number of simultaneous OData requests on the server instance (for all tenants). When the limit is exceeded, a 429 (Too Many Requests) error occurs. If you don't want a limit, set the value `0`. OData requests consume server instance resources, which can affect the performance of the clients if the number of requests gets too large. This setting enables you to control the resources allocated for OData requests. Default: 0 Dynamically Updatable: Yes
Max Connections Per Tenant	ODataMaxConnectionsPerTenant	Specifies the maximum number of simultaneous OData requests per tenant. When the limit is exceeded, a 429 (Too Many Requests) error occurs. If you don't want a limit, set the value 0. If the server isn't configured for multitenancy or only has a single tenant, then this setting does the same as the **Max Connections** (ODataMaxConnections) setting. OData requests consume server instance resources and can affect the performance of the clients if the number of requests gets too large. This setting enables you to control the resources allocated for OData requests. Default: 0 Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
Max Page Size	ODataServicesMaxPageSize	Specifies the maximum number of entities returned per page of OData results. For more information, see Server-Driven Paging in OData Web Services. Default: 20000 Dynamically Updatable: No
OData Base URL	PublicODataBaseUrl	Specifies the root of the URLs that are used to access OData web services. For example, you can change the value if you want to change the externally facing endpoint. The base URL must have the following syntax: http[s]://*hostname*:*port*/*instance*/OData/ This field maps to the `PublicODataBaseUrl` setting in the CustomSettings.config file for the Business Central Server instance. Default: The OData URL for the server instance Dynamically Updatable: No
Port	ODataServicesPort	The listening HTTP port for Business Central OData web services. Valid range: 1 - 65535 Default: 7048 Dynamically Updatable: No
V3 Max Concurrent Requests[1]	ODataV4MaxConcurrentRequests	Specifies the maximum number of OData V3 requests per tenant that the server instance can process at the same time. Requests that exceed the limit will wait in the queue until a time slot becomes available. If you don't want a limit, set the value `0`. Default: 5 Dynamically Updatable: Yes
V3 Max Queued Requests[1]	ODataV4MaxRequestQueueSize	Specifies the maximum number of pending OData V3 requests per tenant waiting to be processed. When the limit is exceeded, a 429 (Too Many Requests) error occurs. If you don't want a limit, set the value `0`. Default: 95 Dynamically Updatable: Yes
V4 Max Concurrent Requests[1]	ODataV4MaxConcurrentRequests	Specifies the maximum number of OData V4 requests per tenant that the server instance can actively process at the same time. Requests that exceed the limit will wait in the queue until a time slot becomes available. If you don't want a limit, set the value `0`. Default: 5 Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
V4 Max Queued Requests[1]	ODataV4MaxRequestQueueSize	Specifies the maximum number of pending OData V4 requests per tenant waiting to be processed. When the limit is exceeded, a 429 (Too Many Requests) error occurs. If you don't want a limit, set the value `0`. Default: 95 Dynamically Updatable: Yes
Timeout	ODataServicesOperationTimeout	Specifies the maximum amount of time that the server instance can allocate to a single OData request. When the limit is exceeded, a 408 (Request Timeout) error occurs. If you don't want a limit, set the value to `MaxValue`. Time format: hh:mm:ss Default: 00:08:00 Dynamically Updatable: Yes

IMPORTANT

The maximum permitted size of an OData web services request is specified by the **Max Message Size** option on the **SOAP Services** tab.

NAS Services Settings

The following table describes fields on the **NAS Services** tab in the Business Central Server Administration tool.

NOTE

Instead of using NAS services, we recommend that you use the Task Scheduler (see Task Scheduler. If you decide to use NAS, and want to read more about its configuration, see Configuring NAS Services in the Dev and IT Pro Help for Microsoft Dynamics NAV 2018.

SETTING	KEY NAME	DESCRIPTION
Enable Debugging	NASServicesEnableDebugging	Specifies if the Business Central Debugger must attach to the NAS Services session. When this is enabled, the NAS Services session waits 60 seconds before the first AL statement is run. Default: Not enabled Dynamically Updatable: No
Run NAS Services with Admin Rights	NASServicesRunWithAdminRights	Specifies whether NAS services run operations with administrator rights instead of the rights granted to the Business Central Server service account. If you select this setting, NAS services will have full permissions in Business Central, similar to the permissions that are granted by the SUPER permission set. The Business Central Server service account isn't required to be set up as a user in Business Central. If you clear this setting, the Business Central Server service account must be added as a user in Business Central and assigned the permissions that are required to perform the operations. Default: Not enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Startup Argument	NASServicesStartupArgument	Specifies a string argument that will be used when NAS services start. The argument typically specifies an application type, sometimes with additional configuration information. Example value: `"OSYNCH"` Default: Dynamically Updatable: No
Startup Codeunit	NASServicesStartupCodeunit	Specifies the codeunit that contains the method that will be called by the **NASStartupMethod** setting. Example values: **0** When **NASStartupCodeunit** is set to 0, NAS Services don't start. This value is the default value. **55** When NAS services start, they run the trigger specified by the NAS Startup Method in codeunit 55. **Note:** When the codeunit specified by **NASStartupCodeunit** is a single instance codeunit, the NAS service session will remain alive even after you run all code in the specified **NASStartupMethod**. Default: Dynamically Updatable: No
Startup Method	NASServicesStartupMethod	Specifies the method that will be called in the **NASStartupCodeunit**. Example values: **""** If no start method is specified (null string), the OnRun trigger is called. **StartNAS** NAS services run the StartNAS method in the NAS Startup Codeunit. Default: Dynamically Updatable: No

Management Services Settings

The following table describes fields on the **Management Services** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Enable Management Services	ManagementServicesEnabled	Specifies whether Business Central Server Administration tool is enabled for this server instance. Default: Enabled Dynamically Updatable: No
Port	ManagementServicesPort	The listening TCP port for the Business Central Server Administration tool. Valid range: 1 - 65535 Default: 7045 Dynamically Updatable: No

Azure Key Vault Client Identity Tab Settings

The following table describes fields on the **Azure Key Vault Client Identity** tab in the Business Central Server Administration tool.

These settings are used when you want to use Azure Key Vaults to store extension secrets and data encryption keys. For more information, see Setting up App Key Vaults and Data Encryption.

SETTING	KEY NAME	DESCRIPTION
Client Certificate Store Location	AzureKeyVaultClientCertificateStoreLocation	Specifies the location of the certificate store where the key vault reader certificate is stored. **LocalMachine** specifies that the certificate is stored in a certificate store for the computer that the Business Central Server is running on. **CurrentUser** specifies that the certificate is stored in a certificate store for your account on the computer that the Business Central Server is running on. Default: LocalMachine Dynamically Updatable: No
Client Certificate Store Name	AzureKeyVaultClientCertificateStoreName	Specifies the certificate store where the key vault reader certificate is stored. Default: My Dynamically Updatable: No
Client Certificate Thumbprint	AzureKeyVaultClientCertificateThumbprint	Specifies the thumbprint of the certificate used by the key vault reader application in Azure. Default: My Dynamically Updatable: No
Client ID	AzureKeyVaultClientId	Specifies the application (client) ID of the key vault reader application in Azure. The value is a GUID. Default: 00000000-0000-0000-0000-000000000000 Dynamically Updatable: No

Azure Key Vault Extension Secrets Tab Settings

The following table describes fields on the **Azure Key Vault Extension Secrets** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Enable Publisher Validation	AzureKeyVaultAppSecretsPublisherValidationEnabled	Specifies whether extensions can only use key vaults that belong to their publishers. • **true** turns on validation. Extensions can only use key vaults that belong to their publishers. This setting blocks attempts in AL to read secrets from another publisher's key vault. • **false** turns off validation. **Important** For security reasons, we recommend that you don't turn this setting off, unless you trust all etxtensions that might get installed on the tenant. An extension publisher's identity is specified when the extension is published. However, the validation is done at runtime. This setting doesn't affect extensions that don't use app key vault secrets. For more information, see App Key Vaults - Security Considerations. Default: true Dynamically Updatable: No

Azure Active Directory (Azure AD) Settings

The following table describes fields on the **Azure Active Directory (Azure AD)** tab in the Business Central Server Administration tool.

The settings in this tab configure the Business Central Server instance to use Azure AD authentication. The settings are only relevant when the server instance is configured Access Control Service, that is, when the **Credential Type** is set to **AccessControlService**. For more information about authenticating using Azure AD, see Authenticating Users with Azure Active Directory.

SETTING	KEY NAME	DESCRIPTION
Application Client Certificate Thumbprint	AzureActiveDirectoryClientCertificateThumbprint	Specifies the thumbprint of the x509 certificate that is used with the Azure AD application client for authentication. A public certificate file (.cer) must be installed on the application client and associated with an Azure AD service principal. A private certificate file (.pfx) must be installed on the computer on which the Business Central Server instance is installed. The server instance service account must have access to the private key of that certificate. Default: Dynamically Updatable: No
Application Client ID	AzureActiveDirectoryClientId	Specifies the ID of the application tenant. The ID is used when accessing data in Azure AD. The authentication token for communicating with Azure AD should be retrieved by specifying the **Application Client Certificate Thumbprint**, with a fallback to use the **Application Client Secret**. Default: 00000000-0000-0000-0000-000000000000 Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Application Client Secret	AzureActiveDirectoryClientSecret	Specifies the secret to use with **Application Client ID** for Azure AD authentication. Default: Dynamically Updatable: No
Azure AD App ID URI	AppIdUri	Specifies the App ID URI that is registered for Business Central in the Microsoft Azure Active Directory (Azure AD). You use this setting to configure Business Central web services for OAuth authentication, specifically when the *Credential Type* setting is **AccessControlService**. it's used to validate the security tokens that the server instance receives in SOAP and OData calls. The App ID URI is a logical identifier and doesn't have to represent a valid location, although it's common practice to use the physical URL of the Business Central web service. The App ID URI is typically the same as the value of *wtrealm* parameter of the **ACSUri** setting that is included in the ClientUserSettings.config file for the Dynamics NAV Client connected to Business Central. An example of an App ID URI is *https://localhost:7047/*. For more information about how to use the Azure Active Directory App ID URI with OAuth authentication, see Using OAuth to Authenticate Web Services (Odata and SOAP). Default: Dynamically Updatable: No
Enable Membership Entitlement	EnableMembershipEntitlement	Configures the server instance to use membership entitlement for controlling access the Business Central. This setting is typically only used for software as a service (SaaS) solutions. Default: Dynamically Updatable: No
Excel add-in AAD client app ID	ExcelAddInAzureActiveDirectoryClientId	This setting is used to set up the Excel Add-in that enables users to use Excel to modify and update Business Central data. The setting specifies the client ID of the Azure AD tenant that is used for the Excel add-in. The Excel add-in requires a separate Azure AD tenant. For more information, see Configuring the Excel Add-In. Default: Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Extended Security Token Lifetime	ExtendedSecurityTokenLifetime	Specifies the number of hours that are added to the lifetime of Azure AD security tokens, which are used to authenticate client users. When the lifetime expires, the client is disconnected from the server instance. An event with a message such as "The SAML2 token is invalid because its validity period ended." is recorded in the event log for the server instance. In general, the lifetime of security tokens is 1 hour. Valid range: 0 to 24 hours Default: 0 Dynamically Updatable: No
Valid Audiences	ValidAudiences	Specifies the allowed audiences for Azure AD authentication. This setting is used to authenticate other Azure AD applications that will communicate with the server instance. The value is a semicolon-separated list of audiences. You specify an audience by using the App URI ID or App ID that is assigned to the application in Azure AD. Default: Dynamically Updatable: No
WS-Federation Login Endpoint	WSFederationLoginEndpoint	Specifies the URL for the federation sign-on page that Business Central redirects to when configured for single sign-on. You must specify a URL in the following format: `https://login.microsoftonline.com/[AADTENANTID]/wsfed?wa=wsignin1.0%26wtrealm=...%26wreply=....` The placeholder [AADTENANTID] represents the GUID of your Azure AD tenant. If the server instance has to support multiple Azure AD tenants, then the **Azure AD Tenant ID** parameter that is specified when mounting a tenant replaces the placeholder. Default: Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
WS-Federation Metadata Location	ClientServicesFederationMetadataLocation	Specifies the URL for the federation metadata document that describes the configuration information for your Azure AD tenant. The federation metadata document is used to validate the security tokens that the Business Central Web client and Business Central tablet client receive, and to establish a trust relationship with between Business Central and an application that you've added to Azure AD. You must specify a URL in the following format: `https://login.microsoftonline.com/[AADTENANTID]/Federat 06/FederationMetadata.xml` The placeholder [AADTENANTID] represents the GUID of your Azure AD tenant. If the server instance has to support multiple Azure AD tenants, then the Azure AD Tenant ID parameter that is specified when mounting a tenant replaces the placeholder. This parameter is relevant only when **Credential Type**, on the **General** tab, is set to **AccessControlService**. For more information, see Authenticating Users with Azure Active Directory. Default: Dynamically Updatable: No

Data Encryption Settings

The following table describes fields on the **Data Encryption** tab in the Business Central Server Administration tool.

For more information about data encyrption, see Data Encryption.

SETTING	KEY NAME	DESCRIPTION
Encryption Key Provider	EncryptionProvider	Specifies where the encryption key used to encrypt data in the database is stored, either **LocalKeyFile** or **AzureKeyVault** values. If you use **AzureKeyVault**, see the **Azure Key Vault Encryption Provider** tab settings. Default: LocalKeyFile Dynamically Updatable: No
Key URI	AzureKeyVaultKeyUri	Specifies the URI of the key in the Key Vault encryption provider setup. Default: Dynamically Updatable: No

Task Scheduler Settings

The following table describes fields on the **Task Scheduler** tab in the Business Central Server Administration tool.

The task scheduler processes jobs and other processes on a scheduled basis. For more information about task scheduler, see Task Scheduler.

SETTING	KEY NAME	DESCRIPTION

SETTING	KEY NAME	DESCRIPTION
Enable Task Scheduler	EnableTaskScheduler	Specifies whether the server instance starts with the task scheduler enabled. If this option is enabled, the server instance will process scheduled tasks. Default: Enabled Dynamically Updatable: No
Maximum Concurrent Running Tasks	TaskSchedulerMaximumConcurrentRunningTasks	Specifies the maximum number of tasks that can run simultaneously on the server instance. The value that you specify will depend on the hardware (CPUs) of the deployment environment and what you want to prioritize: client performance or scheduled tasks (such as job queue entries). The setting is particularly relevant when the server instance is used for both scheduled tasks and client services. If there are many jobs running at the same time, you might experience that the response time for clients gets slower. In which case, you could decrease the value. However, if the value is too low, it might take longer than desired for scheduled tasks to process. When you've a dedicated server instance for scheduled tasks, this setting is less important with respect to client performance. Default: 10 Dynamically Updatable: Yes
System Task Start Time	TaskSchedulerSystemTaskStartTime	Specifies the time of day after which system tasks can start. The time is based on the time zone of the computer that is running the server instance. The value has the format hh:mm:ss. Default: 00:00:00 Dynamically Updatable: Yes
System Task End Time	TaskSchedulerSystemTaskEndTime	Specifies the time of day after which system tasks can't start. The time is based on the time zone of the computer that is running the server instance. The value has the format hh:mm:ss. Default: 23:59:59 Dynamically Updatable: Yes
not available	XmlMetadataCacheSize	For internal use only. Default: 500

Asynchronous Processing Settings

APPLIES TO: Business Central 2019 release wave 2 and later

The following table describes fields on the **Asynchronous Processing** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION

SETTING	KEY NAME	DESCRIPTION
Child Sessions Max Concurrency	ChildSessionsMaxConcurrency	Specifies the maximum number of child sessions that can run concurrently per parent session of a page background task. When the value is exceeded, additional child sessions will be queued and run when a slot becomes available as other child sessions are finished. Default: 5 Dynamically Updatable: Yes **INTRODUCED IN:** Version 15.2
Child Sessions Max Queue Length	ChildSessionsMaxQueueLength	Specifies the maximum number of child sessions that can be queued per parent session of a page background task. If the value is exceeded, an error occurs. Default: 100 Dynamically Updatable: Yes **INTRODUCED IN:** Version 15.2
Page Background Task Default Timeout	PageBackgroundTaskDefaultTimeout	Specifies the default amount of time that page background tasks can run before being canceled. Page background tasks can be also given a timeout value when enqueued at runtime. The **Page Background Task Default Timeout** setting is used when no timeout is provided when the page background task is enqueued. The value has the format hh:mm:ss. Default: 00:02:00 Dynamically Updatable: Yes
Page Background Task Max Timeout	PageBackgroundTaskMaxTimeout	Specifies the maximum amount of time that page background tasks can run before being canceled. Page background tasks can be also given a timeout value when enqueued at runtime. If a page background task is enqueued with a timeout greater than the **Page Background Task Max Timeout** setting, the **Page Background Task Max Timeout** setting is used instead. The value has the format hh:mm:ss. Default: 00:10:00 Dynamically Updatable: Yes

For more information about page background tasks, see Page Background Tasks.

Reports Settings

The following table describes fields on the **Reports** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION

SETTING	KEY NAME	DESCRIPTION
Default Max Documents[1]	ReportDefaultMaxDocuments	Specifies the number of documents that can be merged when using WordMergeDataItem property on reports. If exceeded, the report will be canceled by the server. Developers can override this setting by using the MaximumDocumentCount property of a report. Client users can do the same when running a report from the report request page. If you don't want a limit, set the value to **MaxValue**. Default: 200 Dynamically Updatable: Yes
Default Max Rendering Timeout [1]	ReportDefaultTimeout	Specifies the maximum execution time that it can take to generate a report. If exceeded, the report will be canceled by the server. Developers can override this setting by using the ExecutionTimeout property of a report. Client users can do the same when running a report from the report request page. If you don't want a limit, set the value to **MaxValue**. For more information about how reports are canceled, see Report Generation and Cancellation Flow. Timeout format: [dd.]hh:mm:ss[.ff] Default: 06:00:00 Dynamically Updatable: Yes
Default Max Rows [1]	ReporDefaultMaxRows	Specifies the maximum number of rows that can be processed in a report by default. If exceeded, the report will be canceled by the server. Developers can override this setting by using the MaximumDataSetSize property of a report. Client users can do the same when running a report from the report request page. If you don't want a limit, set the value to **MaxValue**. For more information about how reports are canceled, see Report Generation and Cancellation Flow. Default: 500000 Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
Enable Application Domain Isolation	ReportAppDomainIsolation	Specifies whether application domain isolation is used for rendering custom RDLC layouts. This setting pertains to on-premise installations only. Enabling application domain isolation provides a more secure and reliable environment for processing custom RDLC layouts. However, it can considerably increase the time it takes to render reports. Disabling application domain isolation can improve the rendering time but might have a negative impact on security and reliability. Default: Enabled Dynamically Updatable: No
Enable Save as Excel on Request Pages of RDLC-layout Reports	EnableSaveToExcelForRdlcReports	Specifies whether users can open or save a report as an Excel document if the report uses an RDLC layout. If you clear this check box, the **Excel** option is removed from the **Print** menu on the request page. Default: Enabled Dynamically Updatable: No
Enable Save as Word on Request Pages of RDLC-layout Reports	EnableSaveToWordForRdlcReports	Specifies whether users can open or save a report as a Microsoft Word document if the report uses an RDLC layout. If you clear this check box, the **Word** option is removed from the **Print** menu on the request page. Default: Enabled Dynamically Updatable: No
Enable Save from Report Preview	EnableSaveFromReportPreview	Specifies whether users can save a report as a PDF, Microsoft Word, or Microsoft Excel document from the report preview window. If you clear this check box, the **Save As** icon is removed from the report preview window. Default: Enabled Dynamically Updatable: No
Enforce Cloud Print Support	ReportCloudPrintingEnforced	Specifies whether cloud printing is supported for reports. This setting must be enabled to send print jobs to any printer that is set up by an extension that subscribes to the OnAfterDocumentPrintReady event. If you disable this setting, the OnAfterDocumentPrintReady event is never raised, and print jobs are directed to the default printing on the server. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Max Documents (hard limit)[1]	ReportMaxDocuments	Specifies the maximum number of documents that can be merged when using WordMergeDataItem property on reports using a Word layout. If exceeded, the report will be canceled by the server. To turn off this limit set the value to **MaxValue**. Timeout format: [dd.]hh:mm:ss[.ff] Default: 500 Dynamically Updatable: Yes
Max Rendering Timeout (hard limit)	ReportTimeout	Specifies the maximum execution time that it can take to generate a report. If exceeded, the report will be canceled by the server. If you don't want a limit, set the value to **MaxValue**. For more information about how reports are canceled, see Report Generation and Cancellation Flow. Timeout format: [dd.]hh:mm:ss[.ff] Default: 12:00:00 Dynamically Updatable: Yes
Max Rows (hard limit)	ReportMaxRows	Specifies the maximum number of rows that can be processed in a report. If exceeded, the report will be canceled by the server. You can also use MaxValue to indicate no limit. If you don't want a limit, set the value to **MaxValue**. For more information about how reports are canceled, see Report Generation and Cancellation Flow. Default: 1000000 Dynamically Updatable: Yes
Report PDF Font Embedding	ReportPDFFontEmbedding	Specifies whether fonts are embedded in PDF files that are generated for reports when the report uses an RDLC report layout at runtime. This setting applies when reports are run and saved as PDF files on the client (from the report request page or print preview window) or on the server instance (by the SAVEAS function or SAVEASPDF function in AL code). **Note:** This setting doesn't apply when a report uses a Word report layout at runtime. You embed fonts in a PDF of a report to make sure that the PDF will use the same fonts as the original file, no matter where the PDF is opened and which fonts are installed on the computer. However, embedding fonts can significantly increase the size of the PDF files. Disabling font embedding decreases the size of the report PDF files. **Note:** This setting is a global setting for font embedding in report PDF files. You can override this setting on a report basis by the specifying the PDFFontEmbedding property. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
not available	CalculateBestPaperSizeForReportPrinting	Determines the paper size to use when printing reports from the client. If set to `true`, the system calculates which of the available paper sizes on the printer is best suited for printing, and then uses that paper size. If set to `false`, the printer's default paper size is used. Default: true

[1] **APPLIES TO:** Business Central 2020 release wave 2 and later

Query Settings

The following table describes fields on the **Query** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Max Execution Timeout	QueryTimeout	Specifies the maximum execution time that it can take to generate a query. If exceeded, the query will be canceled by the server. If you don't want a limit, set the value to **MaxValue**. Timeout format: [dd.]hh:mm:ss[.ff] Default: MaxValue Dynamically Updatable: Yes
Max Rows	QueryMaxRows	Specifies the maximum number of rows that can be processed in a query. If exceeded, the query will be canceled by the server. You can also use MaxValue to indicate no limit. If you don't want a limit, set the value to **MaxValue**. Default: MaxValue Dynamically Updatable: Yes

Extensions Settings

The following table describes fields on the **Extensions** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Enable Profile Cache Synchronization	EnableProfileCacheSynchronization	Specifies whether profile cache synchronization across multiple server instances is enabled. This setting is relevant when there's more than one server instance connected to the same database or tenant. When not enabled, changes done to profile configurations on one server instance won't show on other server instances until the first server instance is restarted. When enabled, profile configuration changes will be automatically synchronized and propagated across all server instances. However, enabling this setting will lower the tenant performance. For more information about profiles and configurations, see Designing Profiles. Default: Not enabled (false) Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Overwrite Existing Translations	OverwriteExistingTranslations	Specifies whether to overwrite existing text translations in the base application with text translations included in extensions. Default: Enabled Dynamically Updatable: No
Required Extensions	RequiredExtensions	Specifies a list of required extensions that can't be uninstalled by using the **Extension Management** page in the client. The extensions can still be uninstalled by using the Uninstall-NAVApp cmdlet of the Business Central Administration Shell. You specify an extension by its AppID (which is a GUID). When you've more than one extension, separate each AppID with a semicolon. The AppID for the BaseApp extension is 437dbf0e-84ff-417a-965d-ed2bb9650972 and System Application extension is 63ca2fa4-4f03-4f2b-a480-172fef340d3f. Default: Blank or "" (empty string) Dynamically Updatable: Yes **APPLIES TO:** Business Central 2019 release wave 2 (version 17.0) and later
Solution Version Extension	SolutionVersionExtension	Specifies the ID of the extension whose version number will show as the Application Version on the client's Help and Support page. Typically, you'd use the extension considered to be your solution's base application. If your solution uses the Microsoft Base Application, set the value to `437dbf0e-84ff-417a-965d-ed2bb9650972`. Default: 00000000-0000-0000-0000-000000000000 Dynamically Updatable: Yes

Development Settings

The following table describes fields on the **Development** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION

SETTING	KEY NAME	DESCRIPTION
Allowed Extension Target Level	ExtensionAllowedTargetLevel	Specifies the allowed target level when publishing extensions. The options are **Internal**, **Extension**, **Solution**, **Personalization**, **Cloud**, and **OnPrem**. **Note:** The **Internal**, **Extension**, **Solution**, and **Personalization** options have been deprecated and replaced with **Cloud** and **OnPrem**. - If you specify the **OnPrem** option, the allowed compilation target is set to everything on-premises. The **OnPrem** setting allows using all restricted APIs. The `target` setting in the `app.json` file must also be set to **OnPrem**. For more information, see JSON Files. - If you specify the **Cloud** option, the allowed extension target level is set to SaaS. - By adding the setting `"target":"Cloud"` in the manifest enables you to submit the extension to AppSource. **Note:** It's recommended to use either **OnPrem** or **Cloud** options to set Allowed Extension Target Level. Default: Internal Dynamically Updatable: No
Debugger – Long Running SQL Statements Threshold	LongRunningSqlStatementsInDebuggerThreshold	Specifies the amount of time (in milliseconds) that an SQL query can run before it's logged in debugger telemetry. Default: Enabled Dynamically Updatable: Yes
Debugger - Number of SQL Statements to Show	AmountOfSqlStatementsInDebugger	Specifies the number of SQL statements used in the debugger. The higher number you choose, the more data will be sent to the debugger. Default: Enabled Dynamically Updatable: Yes
Debugger - Show Long Running SQL Statements	EnableLongRunningSqlStatementsInDebugger	Specifies whether long running SQL statements will be shown in the debugger. Default: Enabled Dynamically Updatable: Yes
Debugger - Show SQL Statements	EnableSqlInformationDebugger	Specifies whether the debugger should collect the last used SQL statements and show them in the debugger. Default: Enabled Dynamically Updatable: Yes
Debugging Allowed	DebuggingAllowed	Specifies whether AL debugging is allowed for the Business Central Server instance. Default: Enabled Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Enable Debugging	EnableDebugging	Specifies whether the Business Central Server instance starts with debugging enabled. If this option is enabled, the following occurs: When the client first connects, all C# files for the application are generated. C# files persist between Business Central Server restarts. Application objects are compiled with debug information. Default: Not enabled Dynamically Updatable: No
Enable Developer Service Endpoint	DeveloperServicesEnabled	Specifies whether the Developer service endpoint is enabled. This setting must be enabled to publish extensions and download symbols. - If the check box is selected, the extension can be published to the allowed extension target level. - If the check box isn't selected, the extension can't be published. - This setting can also be controlled from the .xml file. Default: Not enabled (check box is cleared) Dynamically Updatable: No
Enable Loading Application Symbol References at Server Startup	EnableSymbolLoadingAtServerStartup	Specifies whether application symbol references should be loaded at server startup. This setting must be enabled to allow any symbol generation. If the setting isn't enabled, the **generatesymbolreference** setting doesn't have any effect. For more information, see Running C/SIDE and AL Side-by-Side. Default: Not enabled (check box is cleared) Dynamically Updatable: No
Enable Multithreaded Compilation of Published Extensions Service Endpoint	EnableMultithreadedCompilation	Specifies whether to use multiple threads for compiling AL extensions that are published to the server instance. Using multiple threads can make compilation faster, but might impact the responsiveness of other sessions that are running on the server instance. Default: Enabled Dynamically Updatable: No **APPLIES TO:** Business Central 2019 release wave 2 (version 17.0) and later.

SETTING	KEY NAME	DESCRIPTION
Enable SSL	DeveloperServicesSSLEnabled	Specifies whether SSL (HTTPS) is enabled for the developer web service port. SSL (Secure Sockets Layer) secures the connection for the web services. - If the check box is selected, then SSL is enabled. - If the check box isn't selected, the developer web service port can't establish a secure connection. Default: Not enabled (check box is cleared) Dynamically Updatable: No
Enable Test Automation	TestAutomationEnabled	Specifies whether test automation is enabled. For more information about test automation, see Testing the Application Overview. Default: Enabled Dynamically Updatable: No
HttpClient AL Function Maximum Timeout	NavHttpClientMaxTimeout	Specifies the maximum allowed timeout value that can be set for the HttpClient Timeout AL method. The value has the format HH:MM:SS. Default: 00:05:00 Dynamically Updatable: Yes
HttpClient AL Function Response Size	NavHttpClientMaxResponseContentSize	Specifies the maximum size in megabytes of a response buffer used by the HttpClient AL function. The maximum allowed extension size can be adjusted based on the HttpClient AL Function Maximum Timeout setting. Default: 15 Dynamically Updatable: Yes
Port	DeveloperServicesPort	The listening HTTP port for Microsoft Dynamics NAV Developer web services. Valid range: 1 - 65535 Default: 7049 Dynamically Updatable: No

Compatibility Settings

The following table describes settings that you can adjust for compatibility with other systems. In most cases, we don't recommend that you change these settings from their default values.

SETTING	KEY NAME	DESCRIPTION

SETTING	KEY NAME	DESCRIPTION
Enable Client Callbacks in Write Transactions	AllowSessionCallSuspendWhenWriteTransactionStarted	Specifies whether callbacks from the server to clients are allowed during write transactions. Client callbacks, for example, are used when displaying a dialog that requires user input. The setting applies to production and test environments. However, one of the main reasons for testing, to catch cases where a transaction is kept open during the suspension of AL execution. Default: Enabled (check box is selected) Dynamically Updatable: No
Enable Cloud Replication Maintenance	EnableCloudReplicationMaintenance	Specifies whether to keep the cloud replication status in the database up to date. When enabled, synchronize operations on tenants and extensions will update records in the **Intelligent Cloud Status** table, and set change tracking on tables that are configured to replicate data. Enabling this setting isn't required for migrating most on-premises solutions to the cloud, and you'll improve synchronization and upgrade performance by disabling it. Default: Disabled (check box is cleared) Dynamically Updatable: Yes **APPLIES TO:** Business Central 2020 release wave 2 (version 17.4) and later.
Security Protocol	SecurityProtocol	Specifies the default security protocol level for the server instance. This setting also controls the HttpClient. Values: **Ssl3**, **Tls**, **Tls11**, **Tls12**, **SystemDefault**. For more information about these values, see SecurityProtocolType Enum. Default: **Tls12** Dynamically Updatable: No
Unsupported Language IDs[1]	UnsupportedLanguageIds	Specifies which installed Business Central languages aren't available for use in the clients. Use this setting to prevent including languages that have the same name but different culture identifiers (LCIDs). A culture identifier is a standard international numeric abbreviation. For example, the LCID for *Spanish - Spain (Traditional)* is 1034, and *English - United States* is 1033. The setting's value is a semicolon-separated list that contains the LCIDs for each language. Default: **1034** Dynamically Updatable: No
Use Client Timestamp For Report Execution Timestamp	ReplaceReportExecutionTimeWithClientTime	Specifies whether to replace the report execution timestamp with the client timestamp instead of the server instance timestamp. Default: Enabled (check box is selected) Dynamically Updatable: No

SETTING	KEY NAME	DESCRIPTION
Use FIND('-') to Populate Pages Instead of FIND('=><')	UseFindMinusWhenPopulatingPage	Specifies whether pages are initially populated by using FIND('-') instead of FIND('=><'). This setting is relevant to pages that display lists in descending order. Enabling this setting ensures that the first record, instead of the last record, is in focus when the page opens. Pages that use the OnFindRecord trigger will ignore this setting and always use FIND('=><'). Default: Enabled (check box is selected) Dynamically Updatable: No

[1] Introduced in Business Central 2020 release wave 1, update 16.4.

> **IMPORTANT**
> Starting March, 2020, Business Central only supports Transport Layer Security (TLS) version 1.2 or later. This applies to Business Central 2019 release wave 2 online and on-premises (version 15.x). Upgrade to Business Central 2019 release wave 2 or later to remove support for earlier versions of TLS. If your solution or an add-on uses TLS 1.0 or 1.1, you must update that configuration or add-on to TLS 1.2 or later as soon as possible. For more information, see Blog post: Update to TLS 1.2 or later in Dynamics 365 Business Central in 2019 release wave 2.

Upgrade Settings

APPLIES TO: Business Central 2019 release wave 2 and later

The following table describes fields on the **Upgrade** tab in the Business Central Server Administration tool.

SETTING	KEY NAME	DESCRIPTION
Integration Records Table (ID)	IntegrationRecordsTableId	Specifies the ID of the application table that is used to store integration record IDs. This setting only applies when you upgrade from Business Central Spring 2019 (version 14) to Business Central 2019 release wave 2 (version 15). It ensures that current integration record IDs that are used in the application can be used as-is after upgrade. When a database is upgraded, a **SystemId** column will be added to the **Integration Records** table, and each record in the table will be assigned a **SystemId** value. The base application looks to this column to identify integration records. If you specify the **Integration Records Table (ID)** setting, the data upgrade process will set the **SystemId** to the same value as integration record ID; otherwise, the value is randomly assigned by the system. If you aren't upgrading to the Business Central 2019 release wave 2 base application, you can either skip this upgrade step by setting the value to 0. This is the case, when you've a fully customized application or you're only doing a technical upgrade. Or, if you 're using a different integration record table than **5151 Integration Record**, set the value to ID of your custom integration record table. The custom integration record table must include the following columns as a minimum (with exact naming): Integration ID, Record ID, Table ID. Default: 5151 Dynamically Updatable: Yes

SETTING	KEY NAME	DESCRIPTION
not available	DestinationAppsForMigration	Specifies the extensions that Microsoft and 3rd-party extensions have a dependency on. The dependencies typically include the base application, system application, and test application. Only use this setting when upgrading from Business Central Spring 2019. The setting is used during the data upgrade process. If configured, when an extension is published, the server instance will automatically modify the manifest of the extension to include the required dependencies. Default: Dynamically Updatable: No

Using Business Central Administration Shell to Modify Server Instance Settings

The Business Central Administration Shell includes several `Set-` cmdlets that enable you to create and modify Business Central Server instances.

The main cmdlet for configuring a server instance is the Set-NAVServerConfiguration cmdlet. You can use this cmdlet to change any of the configuration settings that are listed in the previous sections. To change a configuration setting, you set `-KeyName` parameter to the **Key Name** that corresponds to the setting, and set the `-KeyValue` parameter to the new value. For example, you can change the value for `DatabaseServer` to `DatabaseServer.Domain.Com` for the server instance named `MyInstance` by executing this cmdlet:

```
Set-NAVServerConfiguration -ServerInstance "MyInstance" -KeyName "DatabaseServer" -KeyValue
"DatabaseServer.Domain.Com"
```

Modifying dynamically updatable settings

For dynamically updatable settings, use the `-ApplyTo` parameter to specify how to apply the change. The change can be written directly to the configuration file (CustomSettings.config) and applied to the current server instance state. The option you choose will determine whether a server instance restart is required for the change to take effect. The parameter has three options, as described in the following table:

OPTION	DESCRIPTION
ConfigFile	Saves the change to the configuration file of the server instance. The change won't take effect until the server instance is restarted.
Memory	Applies the change only to the server instance's current state. The changes take effect immediately, without having to restart the server instance. The change is stored in memory. The next time the server instance is restarted, it reverts to the setting in the configuration file.
All	Applies the change to the server instance's current setting state (in memory) and to the configuration file. The changes take effect immediately, without having to restart the server instance.

For example, the following command sets the value for the `MaxStreamReadSize` key to `42424242`, without having to restart the server instance.

```
Set-NAVServerConfiguration -ServerInstanceMyInstance -KeyName MaxStreamReadSize -KeyValue 42424242 -ApplyTo
Memory
```

For more information about running the Business Central Administration Shell, see Microsoft Dynamics NAV Windows PowerShell Cmdlets.

See Also

Business Central Server Administration Tool

Configuring Business Central Web Server Instances

2/17/2021 • 11 minutes to read • Edit Online

Accessing Business Central from the Business Central Web client or App requires a Business Central Web Server instance on IIS. You create a Business Central Web Server instance for the Business Central Web Server by using the Business Central Setup to install the Business Central Web Server or by running the New-NAVWebServerInstance cmdlet. When you set up a web server instance, you are configuring the connection from the Business Central Web Server to the Business Central Server instance. The connection settings, along with several other configuration settings, are saved in a configuration file for the web server instance.

About the configuration file

The configuration file for the web server instances is a .json file type called **navsettings.json**. The navsettings.json file is a Java Script Object Notification file type that is similar to files that use the XML file format.

After installation, you can change the configuration by modifying the navsettings.json. There are two ways to modify this file: directly or using PowerShell.

Where to find the navsettings.json file

The navsettings.json file is stored in the physical path of the web server instance, which is by default is *%systemroot%\inetpub\wwwroot\[WebServerInstanceName]*.

[WebServerInstanceName] corresponds to the name (alias) of the web server instance in IIS, for example, *c:\inetpub\wwwroot\BC160*.

Modify the navsettings.json file directly

1. Open the navsettings.json in any text or code editor, such as Notepad or Visual Studio Code.

   Each setting is defined by a key-value pair.

   - In the navsettings.json file, a setting has the format:

     ```
     "keyname": "keyvalue",
     ```

   The `keyname` is the name of the configuration setting and the `keyvalue` is the value.

   For example, the configuration setting that specifies the Windows credential type for authenticating users is:

   ```
   "ClientServicesCredentialType":  "Windows",
   ```

   Include values in double quotes.

2. Find the configuration settings that you want to change, and then change the values.

   See the Settings section for a description of each setting.

3. When you are done making changes, save the file.

4. Restart the Business Central Web Server instance for the changes to take effect.

For example, in IIS Manager, in the **Connections** pane, select website node for Business Central Web Server, and then in the **Actions** pane, choose **Restart**. Or, from your desktop, run `iisreset`.

Modify the navsettings.json file by using the Set-NAVWebServerInstanceConfiguration cmdlet

The PowerShell script module **NAVWebClientManagement.psm1** includes the Set-NAVWebServerInstanceConfiguration cmdlet that enables you to configure a web server instance.

1. Depending on your installation, run the Dynamics NAV Development Shell or Windows PowerShell as an administrator.

   For more information, see Get started with the Business Central Web Server cmdlets.

2. For each setting that you want to change, at the command prompt, run the following command:

   ```
   Set-NAVWebServerInstanceConfiguration -Server [MyComputer] -ServerInstance [ServerInstanceName] -
   WebServerInstance [MyBCWebServerInstance] -KeyName [Setting] -KeyValue [Value]
   ```

   Replace:

   - `[MyComputer]` with the name of the computer that is running the Business Central Server
   - `[ServerInstanceName]` with the name of the server instance, such as **BC160**.
   - `[MyBCWebServerInstance]` with the name of the web server instance for the Business Central Web Server.
   - `[KeyName]` with the name of the setting. Refer to the next section in this article.
   - `[KeyValue]` with the new value of the setting.

Settings in the navsettings.json

The navsettings.json has the following structure, where settings are included under one of two root elements: `NAVWebSettings` and `ApplicationIdSettings`:

```
{
  "NAVWebSettings": {
     "//ServerInstance":  "Name of the Business Central Server instance to connect to (for client) or listen
on (for server).",
     "ServerInstance":  "BC150",
      [...more keys]
 },
  "ApplicationIdSettings": {
      "BlogLink": "https://myBCBlog.com",
      [...more keys]
  }
}
```

`//` indicates a comment that provides help for the setting, and has no effect on the Web Server instance.

The following table describes the settings that are available in the navsettings.json for each root element. If you do not see a setting in the file, this is because some settings are not automatically included as a key in the file. For these settings, you can add the key manually. If you do not add the key, the default value of the setting is used.

> **IMPORTANT**
> If modifying the file directly, place values in double quotes `""`.

`NAVWebSettings` element settings

SETTING/KEYNAME	DESCRIPTION
AllowedFrameAncestors	Specifies the host name of any web sites in which the Business Central Web client or parts of are embedded. By default, the Business Central Web Server will not allow a website to display it inside an iframe unless the website is hosted on the same web server. This value of this setting is a comma-separated list of host names (URIs). Wildcard names are accepted. For example: `https:mysite.sharepoint.com, https:*.myportal.com`
AllowNtlm	Specifies whether NT LAN Manager (NTLM) fallback is permitted for authentication. To require Kerberos authentication, set this value to `false`. Values: `true`, `false` Default value: `true`
AuthenticateServer	Specifies whether to authenticate the server by enabling service identity checks on protocol between the Web server and the Business Central Server instance. Values: `true`, `false` Default value: `true`
ClientServicesCredentialType	Specifies the authorization mechanism that is used to authenticate users who try to connect to the Business Central Web Server. For more information, see Authentication and User Credential Type. The credential type must match the credential type configured for the Business Central Server instance that the Business Central Web Server connects to. For information about how to set up the credential type on Business Central Server, see Configuring Business Central Server. Values: Windows, UserName, NavUserPassword, AccessControlService Default value: Windows

SETTING/KEYNAME	DESCRIPTION
ClientServicesChunkSize	Sets the maximum size, in kilobytes, of a data chunk that is transmitted between Business Central Web Server and Business Central Server. Data that is transmitted between Business Central Web Server and Business Central Server is broken down into smaller units called chunks, and then reassembled when it reaches its destination. Values: From 4 to 80. Default value: 28
ClientServicesCompressionThreshold	Sets the threshold in memory consumption at which Business Central Web Server starts compressing data sets. This limits amount of consumed memory. The value is in kilobytes. Default value: 64
ClientServicesPort	Specifies the TCP port for the Business Central Server. This is part of the Business Central Server's URL. Values: 1-65535 Default value: 7046
ClientServicesProtectionLevel	Specifies the security services used to protect the data stream between the Business Central Web Server and Business Central Server. This value must match the value that is specified in the Business Central Server configuration file. For more information, see Configuring Business Central Server. Values: EncryptAndSign, Sign, None Default value: EncryptAndSign
DefaultRelativeHelpPath	Specifies the default Help article to open if no other context-sensitive link is specified. Default value: none
Designer	Specifies whether the in-client designer is enabled in the Business Central Web client. Set to `true` to enable designer. For more information, see Using Designer.

SETTING/KEYNAME	DESCRIPTION
DnsIdentity	Specifies the subject name or common name of the service certificate for Business Central Server. This parameter is only relevant when the ClientServicesCredentialType is set to UserName, NavUserPassword, or AccessControlService, which requires that security certificates are used on the Business Central Web Server and Business Central Server to protect communication. **Note:** You can find the subject name by opening the certificate in the Certificates snap-in for Microsoft Management Console (MMC) on the computer that is running Business Central Web Server or Business Central Server. For more information, see Authentication and User Credential Type. Value: The subject name of the certificate. Default value: none
GlobalEndPoints	Specifies the comma-separated list of global endpoints that are allowed to call this website. The values must include http scheme and fully qualified domain name (FDQN), such as `https://businesscentral.microsoft.com`. "GlobalEndPoints": "null,ms://businesscentral,ms://dynamicsnav" Default value: none
HelpServer	Specifies the name of the Business Central Help Server if the deployment uses Help Server. If the deployment uses an online library, remove this setting. Default value: none
HelpServerPort	Specifies the TCP port on the specified Business Central Help Server if the deployment uses Help Server. If the deployment uses an online library, remove this setting. Default value: none
LoadScriptsFromCdn	Specifies whether to load scripts from Content Distribution Networks (CDNs). This only applies to scripts that are available from a CDN, like jQuery. If set to `false`, scripts will be loaded from the Web server, which is useful in, for example, an intranet scenario where there is no internet access. Default value: `true` **DISCONTINUED AFTER:** Business Central Spring 2019
ManagementServicesPort	The listening TCP port for the Business Central management endpoint. Valid range: 1-65535 Default value: 7045

SETTING/KEYNAME	DESCRIPTION
OfficeSuiteShellServiceClientTimeout	Defines the time Business Central will wait for the Office Suite Shell Service to respond. **Important:** This setting has been deprecated in Business Central, and it has no effect on the Web Server instance. Default value: 10
PersonalizationEnabled	Specifies whether personalization is enabled in the Business Central Web client. Set to `true` to enable personalization. For more information, see Managing Personalization.
ProductName	Specifies the full name of the application.
ProductNameShort	Specifies the short name of the application.
ProductNameMarketing	Specifies the marketing name of the application.
RequireSsl	Specifies whether SSL (https) is required. If the value is set to `true`, all cookies will be marked with a `\u0027secure\u0027` attribute. If SSl is enabled on the Web server, you should set this to `true`. Values: `true`, `false` Default value: `false`
Server	Specifies the name of the computer that is running the Business Central Server. Default value: localhost
ServerInstance	Specifies the name of the Business Central Server instance that the Business Central Web Server connects to. Default value: BC160
ServicePrincipalNameRequired	If this parameter is set to `true`, then the Business Central Web Server can only connect to a Business Central Server instance that has been associated with a service principal name (SPN). If this parameter is set to `false`, then the Business Central Web Server attempts to connect to the configured Business Central Server service, regardless of whether that service is associated with an SPN. For more information about SPNs, see Configure Delegation. Default: `false`

SETTING/KEYNAME	DESCRIPTION
SessionTimeout	Specifies the maximum time that a connection between the Business Central Web Server and the Business Central Server can remain idle before the session is stopped. In the Business Central Web Server, this setting determines how long an open Business Central page or report can remain inactive before it closes. For example, when the SessionTimeout is set to 20 minutes, if a user does not take any action on a page within 20 minutes, then the page closes and it is replaced with the following message: **The page has expired and the content cannot be displayed.** The time span has the format [dd.]hh:mm:ss[.ff]: - dd is the number of days - hh is the number of hours - mm is the number of minutes - ss is the number of seconds - ff is fractions of a second Default value: 00:20:00
ShowPageSearch	Specifies whether to show the  **Tell me what you want to do** icon in the Business Central header. This feature lets users find Business Central objects, such as pages, reports, and actions. If you do not want to show the **Tell me what you want to do** icon, then set the parameter to `false`. Default value: `true`

SETTING/KEYNAME	DESCRIPTION
UnknownSpnHint	Specifies whether to use a server principal name when establishing the connection between the Business Central Web Server server and Business Central Server. This setting is used to authenticate the Business Central Server, and it prevents the Business Central Web Server server from restarting when it connects to Business Central Server for the first time. You set values that are based on the value of the ServicePrincipalNameRequired key. Value: The value has the following format. (net.tcp://BCServer:Port/ServerInstance/Service)=NoSpn|SPN - BCServer is the name of the computer that is running the Business Central Server. - Port is the port number on which the Business Central Server is running. - ServerInstance is the name of the Business Central Server instance. - NoSpn|SPN specifies whether to use an SPN. If the ServicePrincipalNameRequired key is set to `false`, then set this value to NoSpn. If the ServicePrincipalNameRequired key is set to `true`, then set this value to Spn. Default value: (net.tcp://localhost:7046/BC160/Service)=NoSpn If you set this key to the incorrect value, then during startup, the Business Central Web Server will automatically determine a correct value. This will cause the Business Central Web Server to restart. **Note:** For most installations, you do not have to change this value. Unlike the Dynamics NAV Client connected to Business Central, this setting is not updated automatically. If you want to change the default value, then you must change it manually.
UseAdditionalSearchTerms	Specifies whether **Tell me** uses the additional search terms that are defined on pages and reports. The additional search terms are specified by the AdditionalSearchTerms and AdditionalSearchTermsML properties. If you set this to `false`, the additional search terms are ignored. Default value: true

`ApplicationIdSettings` element settings

SETTING/KEYNAME	DESCRIPTION
BaseHelpUrl	Specifies the link to the online Help library that the deployment uses, such as *https://mysite.com/{0}/mylibrary/*. Default value: none For more information, see Configuring the Help Experience.

SETTING/KEYNAME	DESCRIPTION
BlogLink	Specifies the target of the **blog** link on the **Help & Support** page. Use this link to give users access to your end-user blog. Value: a valid URL Default value: https://go.microsoft.com/fwlink/?linkid=2076643|
ComingSoonLink	Specifies the target of **coming soon** link on the **Help & Support** page. Use this link to give users access to information about your roadmap or any upcoming features and fixes. Value: A valid URL. Default value: https://go.microsoft.com/fwlink/?linkid=2047422
CommunityLink	Specifies the URL to a community or resource for sharing information. Value: a valid URL Default value: https://go.microsoft.com/fwlink/?LinkId=287089|
ContactSalesLink	Specifies the target of the contact sales link on the **Help & Support** page. Use this link to give users access to your sales-focused web page where users can engage with your sales process. **Note:** This setting and link are not used for Business Central on-premises.
SigninHelpLink	Specifies the URL to open if the user selects help on the sign-in page box. Value: a valid URL Default value: none

See Also

Setting up Multiple Business Central Web Server Instances
Install Business Central Components
Business Central Web Server Overview
Configuring Business Central Server
Configuring the Help Experience

Setting Up Multiple Business Central Web Server Instances Using PowerShell

2/17/2021 • 5 minutes to read • Edit Online

Although you can use the Business Central Setup to install the Business Central Web Server components and create a single web server instance in IIS for client connection, there may be scenarios when you want to set up multiple instances. For example, you could set up a separate Business Central Web Server instance for the different companies of a business. For this scenario, you can use the Business Central Web Server PowerShell cmdlets, which are outlined in the following table.

CMDLET	DESCRIPTION
New-NAVWebServerInstance	Creates a new Business Central Web Server instance and binds this instance to a Business Central Server instance.
Set-NAVWebServerInstanceConfiguration	Specifies configuration values for a named Business Central Web Server instance.
Get-NAVWebServerInstance	Gets the information about the Business Central Web Server instances that are registered on a computer.
Remove-NAVWebServerInstance	Removes an existing Business Central Web Server instance.

Get started with the Business Central Web Server cmdlets

The Business Central Web Server cmdlets are contained in the PowerShell script module **NAVWebClientManagement.psm1**, which is available on the Business Central installation media (DVD).

The module is installed with the Business Central Server or the Business Central Web Server components.

There are different ways to launch this module and start using the cmdlets:

- If you are working on the computer where the Business Central Server was installed, run the Business Central Administration Shell as an administrator.

  For more information, see Business Central PowerShell Cmdlets.

- If you installed the Business Central Web Server components, just start Windows PowerShell as an administrator.

- Otherwise, start Windows PowerShell as an administrator, and use the Import-Module cmdlet to import the **NAVWebClientManagement.psm1** file:

  ```
  Import-Module -Name [filepath]
  ```

  For example:

  ```
  Import-Module -Name "C:\Program Files\Microsoft Dynamics 365 Business
  Central\130\Service\NAVWebClientManagement.psm1"
  ```

For more information, see the Windows PowerShell docs.

Creating Business Central Web Server instances

Get Access to the WebPublish folder

To create a new web server instance, you need access to the **WebPublish** folder that contains the content files for the Business Central Web Server components.

- This folder is available on the Business Central installation media (DVD) and has the path "DVD\WebClient\Microsoft Dynamics NAV\13x\Web Client\WebPublish".
- If you installed the Business Central Web Server components, this folder has the path "C:\Program Files\Microsoft Dynamics 365 Business Central\160\Web Client\WebPublish".

You can use either of these locations or you can copy the folder to more convenient location on your computer or network.

Decide on the site deployment type for the instance

When you create a new Business Central Web Server instance, you can choose to create either a RootSite or SubSite type. Each instance type has a different hierarchical structure in IIS, which influences its configuration and the URLs for the accessing from the Business Central Web client.

RootSite

A *RootSite* instance is a root-level web site that is complete with content files for serving the Business Central Web client. It is configured with its own set of bindings for accessing the site, such as protocol (http or https) and communication port. The structure in IIS looks like this:

```
- Sites
  - BusunessCentralWebSite (web site)
    + nn-NN (language versions)
    + www (content)
    navsettings.json
    ...
```

The Business Central Web client URL for the RootSite instance has the format:

`https://[WebserverComputerName]:[port]`

For example: `https://localhost:8080`.

SubSite

A *SubSite* instance is a web application that is under a container web site. The container web site is configured with a set of bindings, but the site itself has no content files. The content files are contained in the application (SubSite). The SubSite inherits the bindings from the container web site. This is the deployment type that is created when you install Business Central Web Server components in the Setup wizard. Using the New-NAVWebServerInstance cmdlet, you can add multiple SubSite instances in the container web site. The structure in IIS for two instances looks like this in IIS:

```
- Sites
  - BusunessCentralWebSite (web site)
    - BusinessCentralWebInstance1 (application)
      + nn-NN (language versions)
      + www
      navsettings.json
      ...
    - BusinessCentralWebInstance2 (application)
      + nn-NN (language versions)
      + www
      navsettings.json
      ...
```

The Business Central Web client URL of a SubSite instance is generally longer than a RootSite because it also contains the application's alias (or virtual path) for the instance, which you define. The URL for a SubSite instance has the format:

`https://[WebserverComputerName]:[port]/[WebServerInstance]`

For example: `https://localhost:8080/BusinessCentralWebInstance1` and `https://localhost:8080/BusinessCentralWebInstance2`.

Run the New-NAVWebServerInstance cmdlet

At the command prompt, run the New-NAVWebServerInstance cmdlet. The following are simple examples for creating a RootSite and SubSite deployment type.

RootSite example:

```
New-NAVWebServerInstance -WebServerInstance MyBCWebsite -Server MyBCServer -ServerInstance
MyBCServerInstance -SiteDeploymentType RootSite -WebSitePort 8081 -PublishFolder "C:\Web Client\WebPublish"
```

SubSite example:

```
New-NAVWebServerInstance -WebServerInstance MyWebApp -Server MyBCServer -ServerInstance MyBCServerInstance -
SiteDeploymentType Subsite -ContainerSiteName MySiteContainer -WebSitePort 8081 -PublishFolder
"C:\WebClient\WebPublish"
```

- Susbtitute *MyBCWebsite* with the name that you want to give the web application in IIS for the web server instance. If you are creating a SubSite deployment type, this name will become part of the URL for opening the Business Central Web client application, for example, https://MyWebServer:8081/MyWebApp.
- Susbtitute *MyBCServer* to the name of the computer that is running the Business Central Server to which you want to connect.
- Susbtitute *MyBCServerInstance* with the name of the Business Central Server instance to use.
- Substitute *MySiteContainer* with name of the container web site under which you want to add the instance. If you specify a name that does not exist, then a new container web site will be created, which contains the new web server instance.
- Susbtitute *8081* with the port number that you want to bind the instance to. If you do not specify a port number, then port 80 is used.
- Substitute *C:\WebClient\WebPublish* with the path to your WebPublish folder. By default, the cmdlet looks in the'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Web Client' folder. So if you are working on a computer where the Business Central Web Server components are installed, you do not

have to set this parameter.

> **NOTE**
>
> This command only sets the required parameters of the NAVWebServerInstance cmdlet. The cmdlet has several other parameters that can use to configure the web server instance. For more information about the syntax and parameters, see New-NAVWebServerInstance.

Modifying a Business Central Web Server instance

After you create the web server instance, if you want to change its configuration, you can use the Set-NAVWebServerInstanceConfiguration cmdlet. Or, you can modify the configuration file (navsettings.json) of the instance directly. For more information, see Configuring Web Server Instances.

See Also

Business Central Web Server Overview
Configuring Web Server Instances

Configuring the Business Central Database

2/17/2021 • 7 minutes to read • Edit Online

For a Business Central Server instance to connect to and access a database in SQL Server, the server instance must be authenticated by the database. As in SQL Server, Business Central supports two authentication modes for database communication: Windows Authentication and SQL Server Authentication. When you set up Business Central, you must ensure that database authentication is configured correctly on both the server instance and database, otherwise the server instance will not be able to connect to the database. By default, Windows Authentication is configured on the server instance and database. With Windows Authentication the Business Central Setup does the work for you.

This article describes how to configure SQL Server Authentication. You perform the configuration in two places: on the databases in SQL Server and on the Business Central Server instance. The procedure is different when the Business Central Server instance is configured as a multitenant server instance than when it is not a multitenant server instance.

Set Up an Encryption Key

When using SQL Server authentication, Business Central requires an encryption key to encrypt the credentials (user name and password) that the Business Central Server instance uses to connect to the Business Central database in SQL Server. The encryption key must be installed on the computer where the Business Central Server is installed and also in the database in SQL Server. In a multitenant deployment, the encryption key must be installed in the application database.

To set up an encryption key, you can use one of the following methods:

- You can create and import your own encryption key by using Business Central Administration Shell cmdlets, as described in the following procedure.
- If you are configuring SQL Server authentication on a Business Central Server instance for the first time, you can use the Business Central Server Administration tool which can automatically create and install a system encryption key. If you decide to use this method, no action is required.

Create and import encryption key

1. In the Business Central Administration Shell, run the New-NAVEncryptionkey cmdlet.

   This creates a file that contains an encryption key. If you already have an encryption key file, you can skip this step.

2. Run the Import-NAVEncryptionkey cmdlet to install the encryption key on the Business Central Server instance and database.

On the computer running the Business Central Server instance, the encryption key file has the name BC160.key and is stored in the `%systemroot%\ProgramData\Microsoft\Microsoft Dynamics NAV\[version]\Server\Keys`. In the database, the encryption key is registered in the dbo.ndopublicencryptionkey table. In a multitenant deployment, the encryption key is registered in the application database.

Configure SQL Authentication on the Database

This section describes how to configure a Business Central database to use SQL Server Authentication with a Business Central Server instance. You can complete the steps in this procedure by using SQL Server Management Studio or Transact-SQL.

> **IMPORTANT**
> In a deployment where the Business Central Server instance is configured as a multitenant server instance, you must complete the following procedure on the application database and tenant database.

Configure SQL Server Authentication on the database in SQL Server

1. Configure the SQL Server instance (Database Engine) that hosts the Business Central database to use SQL Server Authentication.

   To use SQL Server authentication, you configure the database instance to mixed authentication mode (SQL Server and Windows Authentication). For more information, see Change Server Authentication Mode.

2. In the SQL Server instance, create a login that uses SQL Server authentication.

   For more information, see Create a Login.

3. Map the login to a user in the Business Central database, and give the user the relevant privileges in the Business Central database.

   For more information, see Create a Database User in the SQL Server docs and the Giving the service account database privileges in SQL Server in the current article.

Configure SQL Server Authentication on the Business Central Instance (Non-Multitenant)

You configure the Business Central Server instance with the login credentials (user name and password) of the user account in the Business Central database in SQL Server that you want to use for authentication. You can do this using the Business Central Server Administration tool or Business Central Administration Shell.

Configure SQL Authentication on a server instance using Business Central Server Administration tool

1. Open the Business Central Server Administration tool.
2. In the console tree, which is the left pane, expand the node for the computer that contains the Business Central Server instance, and then select the Business Central Server instance.
3. In the **Actions** pane, choose **Database Credentials**.
4. On the **Database Credentials** page, choose the **Edit** button.
5. Set the **Database Authentication Type** to **SQL Authentication**.
6. In the **Database User Name** field, type the login name for the database user that you want to use to access the Business Central database in SQL Server.
7. In the **Password** field, type the login password for the database user that you want to use to access the Business Central database in SQL Server.
8. Choose the **Save** button, and then on the **Enable Encryption on SQL Server Connections** dialog box, choose the **OK** button.

   Encryption keys are used to help secure the login credentials over the connection between the Business Central Server instance and the Business Central database in SQL Server.

9. On the **Information** dialog box about encryption, choose the **OK** button.

   This dialog box is to inform you to enable encryption on SQL Server connections, which is disabled by default.

10. If you want to enable encryption on SQL Server connections, in the **Action** pane, choose **Configuration**, and then choose the **Edit** button. In the **Database** tab, select **Enable Encryption on SQL Connections**, choose the **Save** button, and then the **OK** button.

11. Restart the server instance.

Configure SQL Authentication on a server instance using Business Central Administration Shell

- If you are modifying an existing Business Central Server instance, run the Set-NAVServerConfiguration cmdlet.

  Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

- If you are creating a new Business Central Server instance, run the New-NAVServerInstance cmdlet.

  Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

Configure SQL Server Authentication on Business Central Instance in a Multitenant Deployment

This section describes how to configure a Business Central database to use SQL Server Authentication with a Business Central Server instance. You can complete the steps in this procedure by using SQL Server Management Studio or Transact-SQL.

To configure a SQL Server Authentication on a Business Central Server instance, you set up the server instance with the login credentials (user name and password) for the user accounts for the application and tenant databases in SQL Server. You can do this using the Business Central Server Administration tool or Business Central Administration Shell.

Configure SQL Authentication using Business Central Server Administration tool

1. Open the Business Central Server Administration tool.

2. In the console tree, which is the left pane, expand the node for the computer that contains the Business Central Server instance, and then select the Business Central Server instance.

3. Configure SQL Server Authentication with the application database as follows:

   a. In the **Actions** pane, choose **Database Credentials**.

   b. On the **Database Credentials** page, choose the **Edit** button.

   c. Set the **Database Authentication Mode** to **SQL Server Authentication**.

   d. In the **Database User Name** field, type the login name for the database user that you want to use to access the Business Central application database in SQL Server.

   e. In the **Password** field, type the login password for the database user that you want to use to access the Business Central database in SQL Server.

   f. Choose the **Save** button, and then on the **Enable Encryption on SQL Server Connections** dialog box, choose the **OK** button.

      Encryption keys are used to help secure the login credentials over the connection between the Business Central Server instance and the Business Central database in SQL Server.

   g. On the **Information** dialog box about encryption, choose the **OK** button.

      This dialog box is to inform you to enable encryption on SQL Server connections, which is

disabled by default.

h. If you want to enable encryption on SQL Server connections, in the **Action** pane, choose **Configuration**, and then choose the **Edit** button. In the **Database** tab, select **Enable Encryption on SQL Connections**, choose the **Save** button, and then the **OK** button.

4. To configure SQL Server Authentication with the tenant database, mount the tenant to the Business Central Server instance and specify the login credentials (user name and password) for the database user that you want to use to access the Business Central tenant database in SQL Server.

   If the tenant is already mounted to the Business Central Server instance, you must dismount the tenant, and mount it again.

   For more information see Mount or Dismount a Tenant.

5. Restart the server instance.

Configure SQL Authentication using Business Central Administration Shell

1. Configure SQL Server Authentication with the application database as follows:

   - If you are modifying an existing Business Central Server instance, run the Set-NAVServerConfiguration cmdlet.

     Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

   - If you are creating a new Business Central Server instance, run the New-NAVServerInstance cmdlet.

     Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the application database.

2. To configure SQL Authentication with the tenant database, run the Mount-NAVTenant cmdlet.

   Use the *DatabaseCredentials* parameter to provide the login credentials of the database user that you want to use to access the tenant database.

See Also

Installation Considerations for Microsoft SQL Server
Deployment
Installing Business Central Using Setup

Encrypting Data in Dynamics 365 Business Central

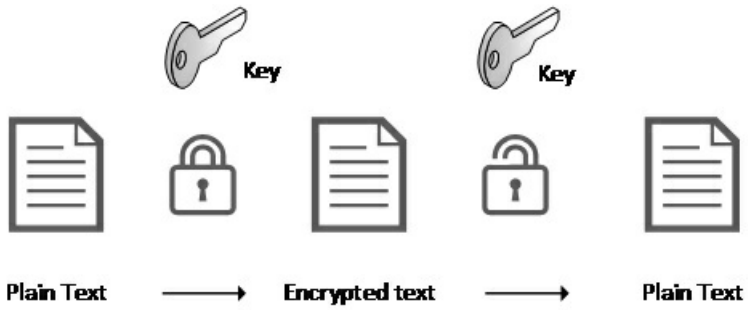
2/17/2021 • 2 minutes to read • Edit Online

This article gives an overview of data encryption and how to use it to add security Dynamics 365 Business Central.

> **NOTE**
> This applies only to on-premises versions of Business Central. For online versions, encryption is always enabled and you cannot turn it off.

Cryptography overview

The methods that support cryptography provide services that enable developers manage encrypting and decrypting data. Each tenant supports a single encryption key which is used for encrypting and decrypting data stored in the database. Additional methods are provided to assist building robust solutions when working with encryption and for managing the encryption keys.



Encryption keys are stored in a secure location, and retrieved at runtime when needed. Additional functionality is provided to export and import keys, which is important when moving solutions from one location to another.

Encryption key management

The process of encrypting and decrypting data requires a key. An encryption key is typically a random string of bits generated specifically to scramble and unscramble data. Encryption keys are created by using algorithms designed to make sure that each key is unique and unpredictable. The keys that are used by Dynamics 365 Business Central are generated by the .NET Framework Data Protection API.

Each tenant supports having a single encryption key. To use the encryption methods, a key must be created. There are two ways of doing this; either by importing a key or by creating a key. The **CREATEENCRYPTIONKEY** method will create an encryption key in a system that does not have a key present. Alternatively, if a key exists, you can use the **IMPORTENCRYPTIONKEY** method to introduce a key to a keyless system.

> **WARNING**
> **CREATEENCRYPTIONKEY** will fail if the key already exists, you must then call **DELETEENCRYPTIONKEY** to clear the state. **IMPORTENCRYPTIONKEY** will throw a warning if a key already exists, regardless of if the key is present on the system or not.

Best practices

These are some best practices we recommend that you follow:

- Make sure to always backup your key and store it securely. Use the **EXPORTENCRYPTIONKEY** method and keep the output file in a secure location.
- Use the Dynamics 365 permission system to restrict access to encryption key logic.
- Be aware of the difference between the **ENCRYPTIONKEYEXISTS** and **ENCRYPTIONENABLED** methods.

ENCRYPTIONKEYEXISTS versus ENCRYPTIONENABLED

The encryption key is stored in a file in a directory that the Dynamics 365 Business Central service has access to. When a key is created or imported, data is recorded in the tenant table registering that encryption has now been enabled. Any subsequent calls to **ENCRYPTIONENABLED** will return true after the tenant table has been updated with this information. However, if the encryption file is deleted, then **ENCRYPTIONENABLED** will continue to return true. Use the **ENCRYPTIONKEYEXISTS** method to perform a file system check to see whether the key is present.

See Also

Application Security in Business Central
Data Encryption

Setting up the Excel Add-In for Editing Business Central Data

2/17/2021 • 9 minutes to read • Edit Online

You can set up the Business Central deployment to support an Excel add-in that enables users in the Business Central client to work with data from list pages in Excel. Users can get fresh data from Business Central into Excel and push updated data from Excel to Business Central.

Without this add-in, users can open a list page in Excel from the **Open in Excel** action on the page. But the **Open in Excel** action doesn't allow them to push changed data back to Business Central. When this add-in is set up, the **Open in Excel** action is replaced by the **Edit in Excel** action. The add-in is based on the Microsoft Dynamics Office Add-In.

> **NOTE**
> The Excel add-in is not available in the mobile apps.

This article provides guidance for how to configure Business Central so that users can edit data in Excel.

> **NOTE**
> This article does not apply to the older Excel add-in that was available for installation with the Dynamics NAV Client connected to Business Central client by using the Business Central Setup in versions older than 2019 release wave 2.

Deploy the Excel add-in for Business Central online

For Business Central online, the administrator can deploy the add-in for all users. But users can also install the add-in themselves, provided they have permission to configure their Office experience.

> **TIP**
> In some organizations, administrators cannot deploy add-ins centrally. For more information, see Determine if Centralized Deployment of add-ins works for your organization.

To deploy the Excel add-in for all users

1. As the administrator, sign in to the Microsoft commercial website and find the add-in at https://appsource.microsoft.com/product/office/WA104379629.
2. Choose the **Get it now** button.

   You'll be redirected to the Microsoft 365 admin center.
3. In the left panel, go to **Settings**, and then choose **Add-ins**.
4. In the **Configure add-in** pane, specify which users to grant access to the add-in.
5. Save your changes.

When users now choose the **Edit in Excel** action, the add-in will launch as a pane in Excel. Each user will be automatically logged in and connected to their Business Central, but if a user cannot connect automatically, you

can unblock them by asking them to follow the steps in the To configure the connection section.

To add the Excel add-in locally

1. Open Excel, and then open any Excel workbook.
2. On the **Insert** menu, choose **Office Add-ins**, and then choose **Admin managed** or **Store** as appropriate.
3. Search for *Dynamics Office Add-In*, and then install the add-in.

When the add-in is installed, it shows up as a panel in Excel. Next, you must configure the connection.

To configure the connection

1. In the Dynamics 365 Excel add-in, choose **Add server information**, and then in the **Server URL** field, enter `https://exceladdinprovider.smb.dynamics.com`.
2. Choose the OK button, and then confirm that the app reloads.
3. When prompted, sign in with your Azure Active Directory account.
4. Optionally, choose the environment and company that you want to connect to.

The add-in is now connected to your Business Central, and you can edit data and publish the changes to Business Central.

> **TIP**
> If the workbook is not automatically saved to the user's OneDrive, then recommend them to save all workbooks that they export from Business Central. When they open the workbook again, the connection is still available, so they do not have to configure the connection again.

> **NOTE**
> In certain deployments, the administrator must configure network access to unblock the Excel add-in. For more information, see Preparing Your Network for the Excel Add-In.

Troubleshooting

Sometimes, users run into problems with the Excel add-in. In this section, we provide tips for how to unblock users in certain circumstances.

ISSUE	SOLUTION OR WORKAROUND	COMMENTS
The add-in doesn't start	Check if the add-in is deployed centrally, or if the user is blocked from installing it locally.	The admin can configure Office so that users cannot acquire add-ins. In those cases, the admin must deploy the add-in centrally. For more information, see Deploy add-ins in the admin center.
Data does not load into Excel	Test the connection by opening another list in Excel from Business Central. Alternatively, open the workbook in Excel in a browser.	If the user has specified a company name that contains special characters, the add-in might not be able to connect.
Data can't publish back to Business Central.	Test the connection by opening the workbook in Excel in a browser.	Sometimes an extension can block the publishing job. If the page is extended or customized, remove the extensions, and then try again.

ISSUE	SOLUTION OR WORKAROUND	COMMENTS
The dates are wrong	Excel might show times and dates in a different format than Business Central. This does not make them wrong, and the data in Business Central will not get messed up.	

Deploy the Excel add-in for Business Central on-premises

Your on-premises deployment must meet the following prerequisites:

- Azure Active Directory (Azure AD) used to authenticate users.

  The Business Central Server instance, clients, and users must be configured for Azure Active Directory (Azure AD) authentication.

  For more information, see Authenticating Users with Azure Active Directory.

- OData enabled and uses Secure Sockets Layer (SSL) for authentication.

  For more information, see Using Security Certificates with Business Central On-Premises.

- Business Central Web Server installed and configured to use SSL (https).

  For more information, see Install Business Central and Configure SSL to Secure the Connection to Web Client.

- If your deployment is multitenant, Business Central Web client must accept host names for tenants.

  For more information, see Configure the Web Server to Accept Host Names for Tenants.

- Business Central client computers have a supported version of Excel.

  For more information, see System Requirements.

Expose the Business Central application Web API in Azure AD

APPLIES TO: Business Central On-Premises

When Business Central is configured for Azure AD authentication, the Business Central application is registered as an application in an Azure AD. Before the Excel add-in can be configured, you must configure the Business Central application in Azure AD to expose its Web API.

For information about how to expose the Web API, see Quickstart: Configure an application to expose web APIs.

Register and configure an Azure AD application for the Excel Add-in in Microsoft Azure

APPLIES TO: Business Central On-Premises

When Azure AD authentication was set up for your Business Central deployment, an Azure AD tenant was created in Microsoft Azure, and an application for Business Central was registered in the tenant. The Excel add-in requires that you add (register) a separate Azure AD application in the Azure AD tenant.

1. Register an Azure AD application for the Excel add-in.

   You add the Azure AD application by using the Azure portal. For guidelines, see Register your application with your Azure Active Directory tenant.

   When you add an application to an Azure AD tenant, you must specify the following information:

SETTING	DESCRIPTION
Name	The name of your application as it will display to your users, such as *Excel Add-in for Business Central*.
Supported account types	Specifies which accounts that you would like your application to support. For purposes of this article, select **Accounts in this organizational directory only**.
Redirect URI	Specifies the type of application that you're registering and the redirect URI (or reply URL) for your application. Select the type to **Web**, and in the redirect URL box, enter URL for signing in to the Business Central Web client, for example `https://localhost:443/BC170/SignIn`. The URI has the format `https://<domain or computer name>/<webserver-instance>/SignIn`, such as `https://cronusinternationltd.onmicrosoft.com/BC170/SignIn` or `https://MyBcWebServer/BC170/Signin`. **Important** The portion of the reply URL after the domain name (in this case `BC170/SignIn`) is case-sensitive, so make sure that the web server instance name matches the case of the web server instance name as it is defined on IIS for your Business Central Web Server installation.

When completed, the **Overview** displays in the portal for the new Excel Add-in application.

2. Grant the Excel add-in application permission to access the Business Central application Web API.

   Give the Azure AD application for the Excel add-in delegated permission to access the Business Central application Web API in Azure AD (which you exposed earlier in this article). This permission allows users of the Excel add-in to access the OData web services to read and write data.

   a. From the application's **Overview**, select **API Permissions**.
   b. Select the **Add a permission**
   c. On the **APIs my organization uses**, select the Business Central application.
   d. Select **Delegated permission**.
   e. Select the permission from the list, and then select **Add Permission**.

   For information, see Quickstart: Configure a client application to access web APIs.

3. Configure OAuth2 authentication in the Excel add-in.

   The Excel add-in requires OAuth2 implicit grant flow to be enabled on the Excel Add-in application. You configure OAuth2 in the manifest file for the Excel Add-in application. From the application's **Overview**, select **Manifest**, and then set `"oauth2AllowIdTokenImplicitFlow"` and `"oauth2AllowImplicitFlow"` to `true`:

   ```
   "oauth2AllowIdTokenImplicitFlow": true,
   "oauth2AllowImplicitFlow": true,
   ```

4. In the manifest, add the following URL entry to the `"replyUrlsWithType":`:

```
{
    "url": "https://az689774.vo.msecnd.net/dynamicsofficeapp/v1.3.0.0/*",
    "type": "Web"
}
```

Remember to add a comma before or after this entry, depending on where you add it in the list.

5. Copy the **Application (Client) ID** that is assigned to Excel add-in application.

   You can get this value from the **Overview** page for the application. You'll need it to configure the Business Central Server instance.

This completes the work you have to do in the Azure portal. The final configuration is to add the Excel add-in to the Business Central Server instances.

Configure the Business Central Server Instances

APPLIES TO: Business Central On-Premises

You add the Excel add-in to the Business Central Server instances in your deployment. You can use either the Business Central Server Administration tool or Set-NAVServerConfiguration cmdlet in the Business Central Administration Shell.

1. In the Business Central Server Administration tool, in the **Azure Active Directory** section, set the **Excel add-in AAD client ID** field to the application (client) ID for the Excel add-in application that you copied from the Azure portal.

   If you use the Set-NAVServerConfiguration cmdlet, set the `ExcelAddInAzureActiveDirectoryClientId` key.

2. In the **Client Services** section, set the **Web Client Base URL** field to the base URL of the Business Central Web client.

   This value is the root portion of all URLs that are used to access pages in the web client. The value must have the format `https://[hostname:port]/[instance]`, such as `https://MyBCWebServer/BC` or `https://cronusinternationltd.onmicrosoft.com/BC170`.

   If using the **Set-NAVServerConfiguration** cmdlet, set the `PublicWebBaseUrl` key.

3. In the **OData Services** section, set the **OData Base URL** field to the public URL for accessing OData services.

   The URL must have the following format `https://<hostname>:<port>/<instance>/ODataV4/`, such as `https://localhost:7048/BC170/ODataV4/`.

   With the **Set-NAVServerConfiguration** cmdlet, set the `PublicODataBaseUrl` key.

Prepare devices and network for the Excel Add-In

Network services such as proxies or firewalls must allow routing between each client device on which the Add-In is installed and a number of service endpoints. For a list of endpoints, see Preparing your network for the Excel Add-In.

Use the Excel Add-In

Your users can now use the Excel add-in. When a list page shows the **Edit in Excel** action, then users can open lists, such as the **Customers** page, in Excel and work with the data there. They use the add-in to update data in Business Central and get fresh data from the database. For more information, see Viewing and Editing in Excel From Business Central.

See Also

Configuring Business Central Server
Authenticating Users with Azure Active Directory

Setting up App Key Vaults for Business Central On-premises

2/17/2021 • 7 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Business Central extensions can be developed to get secrets from Azure Keys Vaults. This article describes the tasks required to set up Azure Keys Vaults for storing extension secrets and configure them in your Business Central deployment.

For more information about developing extensions with key vaults, see Using Key Vault Secrets in Business Central Extensions.

Prerequisites

To complete the tasks in this article, you need:

- An Azure subscription with an Active Directory tenant.

  You sign up for an Azure subscription at https://azure.microsoft.com. For information about getting an Azure AD tenant, see How to get an Azure Active Directory tenant.

- A security certificate

  Later, you'll have to register and configure an application in Azure AD for reading key vaults. This step requires a certificate. The certificate is used to prove the application's identity when requesting upon request. For a production environment, obtain a certificate from a certification authority or trusted provider.

  In a test environment, if you don't have a certificate, then you can create your own self-signed certificate. For example, on the Business Central server computer, start Windows PowerShell as an administrator. Then at the prompt, run the following commands:

  ```
  $cert = New-SelfSignedCertificate -Subject "BusinessCentralKeyVaultReader" -Provider "Microsoft
  Strong Cryptographic Provider"
  $cert.Thumbprint
  Export-Certificate -Cert $cert -FilePath c:\certs\BusinessCentralKeyVaultReader.cer
  ```

  These commands add a certificate called BusinessCentralKeyVaultReader to the computer's **LocalMachine** > **Personal (My)** certificate store.

Create the Azure Key Vault with secrets

Now, you create one or more key vaults in Azure, and add the secrets that you want to make available to your extensions. An extension can be set up with one or two key vaults.

There are different ways to create an Azure key vault. For example, you can use the Azure portal, Azure CLI, and more.

- The easiest way is to use the Azure portal. For instructions, see Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal.

  After you create the key vault, add the secrets.

- For using other methods, see Azure Key Vault Developer's Guide.

Register a key vault reader application in Azure AD

Next, register an application on your Azure AD tenant for reading secrets from the key vaults. When Azure AD authentication was set up, an Azure AD tenant was created in Azure. Reading key vaults requires a separate application registration with the Azure AD tenant. You can use an existing application.

The steps in this task are done from the Azure portal.

1. Sign in to Azure portal at portal.azure.com and set the portal to your Azure Active Directory tenant.
2. Register an Azure AD application for the reading key vault.

   You add the new application by using the Azure portal. For guidelines, see Register your application with your Azure Active Directory tenant.

   When you add an application to an Azure AD tenant, you must specify the following information:

SETTING	DESCRIPTION
Name	The name of your application as it will display to your users, such as *Business Central Key Vault Reader*.
Supported account types	Specifies which accounts that you would like your application to support. For purposes of this article, select **Accounts in this organizational directory only**.

   When completed, the **Overview** displays in the portal for the new application.

   Copy the **Application (client) ID**. You'll use this information later.
3. Upload the security certificate to the registered application.

   In this step, you upload the certificate file that you obtained as part of the prerequisites.

   Go to the key vault reader application overview page. Select **Certificates & secrets** > **Upload certificate**. Follow the instructions to locate and upload the certificate.

Grant the key vault reader application permission to key vaults

In this task, you grant the key vault reader application permission to read secrets from your key vaults.

The steps in this task are done from the Azure portal.

1. Open the key vault in the portal.
2. Select **Access policies**, then **Add Access Policy**.
3. Set **Secret Permissions** to **Get**.
4. Choose **Select principal**, and on the right, search for either **Application (client) ID** or display name for the key vault reader application.
5. Select **Add**.
6. Select **Save**.

At this point, the work in Azure is finished.

Configure the Business Central Server to use the Apps Key Vault feature

Next, you configure the Business Central Server instance to use the key vault reader application and its certificate, which you registered in Azure AD, for authenticating to the key vaults.

If you're running a container-based environment, you have two options for configuring the server instance. You can either do it manually or use the Set-BcContainerKeyVaultAadAppAndCertificate script. Using the Set-BcContainerKeyVaultAadAppAndCertificate script is simpler and recommended.

Configure a container-based Business Central Server instance

If you are running a container-based environment, use the Set-BcContainerKeyVaultAadAppAndCertificate.ps1 script that is available in the NAV Container Helper GitHub repository at https://github.com/microsoft/navcontainerhelper/blob/master/ContainerHandling/Set-BcContainerKeyVaultAadAppAndCertificate.ps1.

Manually configure a Business Central Server instance

To complete this task, you'll need the user name of the service account that runs the Business Central Server.

1. If not already done, import your key vault certificate and its private keys to the local certificate store for the Business Central server computer.

   You can import the certificate either using the MMC snap-in or Import-PfxCertificate cmdlet from a Windows PowerShell prompt.

   For example, the following PowerShell command installs a certificate to the local machine's personal store:

   ```
   Import-PfxCertificate -FilePath "C:\certificates\BusinessCentralKeyVaultReader.pfx" -Password
   (ConvertTo-SecureString -String "pfxpassword" -AsPlainText -Force) -CertStoreLocation
   Cert:\LocalMachine\My\
   ```

2. Give the service account used by the Business Central Server instance permission to access the certificates private key.

   To do this using the MMC:

   a. Open the MMC snap-in for certificates. See How to: View Certificates with the MMC Snap-in.

   b. Expand the **Certificates (Local Computer)** node, expand the **Personal** node, and then select the **Certificates** subfolder.

   c. In the right pane, right-click the certificate, select **All Tasks**, and then choose **Manage Private Keys**.

   d. In the **Security** dialog box, choose **Add**.

   e. In the **Select Users, Computers, Service Accounts, or Groups** dialog box, enter the name of the dedicated domain user account that is associated with Business Central Server, for example, NETWORK SERVICE. Then, choose the **OK** button.

   f. In the **Full Control** field, select **Allow**, and then choose the **OK** button.

3. Make a note of the certificate thumbprint because you'll need it in the next step. See How to: Retrieve the Thumbprint of a Certificate.

4. Configure the Business Central Server instance.

   Now, you'll configure App Key Vault settings on the server instance. The following table describes the settings that you must configure:

SETTING (KEY NAME)	VALUE
Client Certificate Store Location (AzureKeyVaultClientCertificateStoreLocation)	Set to the certificate store location where key vault certificate was stored. Example: LocalMachine
Client Certificate Store Name (AzureKeyVaultClientCertificateStoreName)	Set to the certificate store name where key vault certificate was stored. Example: MY
Client Certificate Thumbprint (AzureKeyVaultClientCertificateThumbprint)	Set to the thumbprint for the key vault certificate. Example: 649419e4fbb87340f5a0f995e605b74c5f6d943e
Client ID (AzureKeyVaultClientId)	Set to the **Application (client) ID** of the key vault reader application registered in your Azure AD tenant. Example: ed4129d9-b913-4514-83db-82e305163bec
Enable Publisher Validation (AzureKeyVaultAppSecretsPublisherValidationEnabled)	Specifies whether extensions can only use key vaults that belong to their publishers. Enabling this setting (`true`) blocks attempts in AL to read secrets from another publisher's key vault. When extensions that use key vault secrets are published, you must provide your Azure AD tenant ID, which is done by using the Publish-NAVApp cmdlet with the `-PublisherAzureActiveDirectoryTenantId` parameter. **Important** We recommend that you only set it to `false` if you trust all extensions that will be installed. For more information, see App Key Vaults - Security considerations. Example: true

You can configure the instance using the Business Central Server Administration tool or Set-NAVServerConfiguration cmdlet.

To use the Set-NAVServerConfiguration cmdlet, start the Business Central Administration Shell as an administrator, and run the following commands one at a time. Replace brackets with your own values.

```
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultClientCertificateStoreLocation -KeyValue <certificate store location>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultClientCertificateStoreName -KeyValue <certifcate store>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultClientCertificateThumbprint -KeyValue <certificate thumbprint>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName AzureKeyVaultClientId -KeyValue
<application ID of key vault reader app in Azure>
Set-NAVServerConfiguration -ServerInstance <serverInstance> -KeyName
AzureKeyVaultAppSecretsPublisherValidationEnabled -KeyValue <true|false>
Restart-NAVServerInstance -ServerInstance <serverInstance>
```

At this point, you can run your extensions that use key vault secrets to read secrets from key vault.

TIP

If your on-premises solution uses the ImportStreamWithUrlAccess method, you must have set up an Azure blob storage account and stored the account name and account keys in the current subscription's Azure KeyVault using the identifiers TEMPORARYDOCUMENTSTORAGEACCOUNT and TEMPORARYDOCUMENTSTORAGEKEY. That way, your users can use the integration with Outlook.

See Also

Using App Key Vaults with Business Central Extensions
Security Considerations With App Key Vaults
Monitoring and Troubleshooting App Key Vaults
Authentication and Credential Types
Configuring Business Central Server

Business Central Performance Counters

2/17/2021 • 7 minutes to read • Edit Online

The following table describes the performance counters that are available in Business Central for monitoring Business Central Server instances.

Client session counters

These counters pertain to sessions from the clients, NAS, and web services, to the server instance.

COUNTER	DESCRIPTION
# Active child sessions	Number of active child sessions on the Business Central Server instance. An active child session is a connection to the server instance from a Business Central client, such as the Dynamics NAV Client connected to Business Central or Business Central Web client, NAS, or Web services. A child session is created by a background task that is run asynchronously on a page. For more information, see Page Background Tasks.
# Active sessions	Number of active sessions on the Business Central Server instance. An active session is a connection to the Business Central Server instance from a Business Central client, such as the Dynamics NAV Client connected to Business Central or Business Central Web client, and Web services (OData and SOAP).
Server operations/sec	Number of operations that have started on the Business Central Server per second. An operation is a call to the Business Central Server instance from a Business Central client to run Business Central objects. Note: OData and SOAP requests are not included.
Average server operation time (ms)	Average duration of server operations in milliseconds.

SQL Server connection counters

These counters pertain to the connection from the server instance to the SQL Server instance and databases.

COUNTER	DESCRIPTION
# Mounted tenants	Number of tenants that are mounted on the Business Central Server instance. This counter is relevant with a multitenant server instance, where tenants are often mounted and dismounted.

COUNTER	DESCRIPTION
# Open connections	The current number of open connections from the Business Central Server instance to Business Central databases on SQL Servers. The value is always equal to the sum of the **# Open tenant connections** counter and the **# Open application connections** counter. -We recommend that you use these counters instead.
# Open application connections	Current number of open application connections from the Business Central Server instance to the Business Central application database on SQL Servers. Because all connections are to only one application database, you will see failures when the total number of connections for all server instances exceeds the maximum number of connections allowed to the database.
# Open tenant connections	Current number of open tenant connections from the Business Central Server instance to Business Central tenant databases on SQL Servers. If there are multiple tenant databases, you cannot see the distribution of opened connections per database (or database pool). With Azure SQL Database, connections are denied if the throttling limit is reached. The limit depends on the database configuration. Be aware that in clusters, other server instances will also have connections to the same database, so the total load on a database requires that you look at multiple server instances.
% Query repositioning rate	Percentage of queries that are re-executed when fetching the query result.
Hard throttled connections	Number of connections that were hard-throttled.
Soft throttled connections	Number of connections that were soft-throttled.
Transient errors	Number of transient errors.
Heartbeat time (ms)	The time that it takes to complete a single write to a system table. Conceptually, this counter measures the time it takes to call the application database server to update the 'last active' field the **dbo.Service Instance** table for the Business Central Server instance. Every 30 seconds, the instance writes a record to indicate that the instance is "alive." You can use this counter to indicate if there is network latency between the Business Central Server and the database.

COUNTER	DESCRIPTION
# Preferred connection total requests	Count of the total number of requests to the preferred connection cache. The preferred connection cache contains requests from the SQL connection pool that was last used by a Business Central user.
% Preferred connection cache hit rate	Percentage of hits in the preferred connection cache, compared to the total number of requests.

Data and caching counters

These counters pertain to the data caching on the server instance.

COUNTER	DESCRIPTION
# Calculated fields cache total requests	Count of the total number of requests to the calculated fields cache. The calculated fields cache contains the results of CALCFIELDS method (Record) calls.
% Calculated fields cache hit rate	Percentage of hits in the calculated fields cache, compared to the total requests to the calculated fields cache.
# Command cache total requests	Count of the total number of requests to the command cache. The command cache contains the results of all SQL commands.
% Command cache hit rate	Percentage of hits in the command cache, compared to the total requests to the command cache.
# Primary key cache total requests	Count of the total number of requests to the primary key cache. The primary key cache contains the results of requests to get a record by using its primary key.
% Primary key cache hit rate	Percentage of hits in the primary key cache, compared to the total requests to the primary key cache.
# Result set cache total requests	Count of the total number of requests to the result set cache. The result set cache contains result sets that are returned from SQL Server.
% Result set cache hit rate	Percentage of hits in the result set cache, compared to the total requests to the result set cache. The value also depends on the usage pattern and which parts of the application are is used. For example, the SELECTLATESTVERSION method will clear the cache, which results in a lower hit rate. In general, reading frequently updated values will lower the hit rate because the cache synchronization across Business Central Server instances will remove stale values, which causes re-reads.
# Rows in all temporary tables	Count of number of rows in all temporary tables.

Scheduled task counters

These pertain to tasks that are run by Task Scheduler.

COUNTER	DESCRIPTION
# Available tasks	Remaining number of tasks that can potentially run simultaneously before the maximum number of tasks is reached. The value of this counter is the value the **Maximum # of tasks** counter minus the value of the **# Running tasks** counter.
# of task errors/sec	Number of errors per second that are caused by running tasks. The task are causing errors in AL or exceptions on the server instance. If the value is greater than zero for an extended period of time, this typically indicates a failing task that keeps getting rescheduled.
# Running tasks	Number of tasks that are currently running on the server instance. The value is limited to the value of the **Maximum # of tasks** counter.
Average task execution time	The average time (in ticks) that tasks have taken to complete. Task execution time is counted regardless of whether the task completed successfully or raised an error. There is no general rule for what the normal operations level is. To analyze this counter, look for large spikes to identify long-running tasks. **Note:** A tick is the smallest unit that the your system uses for time measurements, and it is typically determined by the operating system. For example, in Windows, a single tick represents one hundred nanoseconds, which means that there are 10,000 ticks in a millisecond. Tick durations can differ bewteen systems, so be aware of this fact when comparing absolute values across systems.
Maximum # of tasks	The maximum number of tasks that can run simultaneously. This value is defined by the **Maximum Concurrent Running Tasks** (TaskSchedulerMaxConcurrentRunningTasks) setting in the server instance configuration. Therefore, this value is constant until the server instance setting is changed and the instance is restarted.
Total # Pending tasks	The total number of tasks in the shared task list that are waiting to be picked up by server instances connected to this application database. The tasks counted are those that are ready and have been scheduled to run now or earlier and that are not currently running.
Total # Running tasks	Total number of tasks in the shared task list that are currently running by any server instance connected to this application database.
Time (ms) since the list of running tasks last had capacity for new tasks	The time (ms) since the list of running tasks last had capacity for new tasks.

For more information about task scheduler, see Task Scheduler.

See Also

Set up Performance Counters in Windows Performance Monitor
Create a Data Collector Set From Template
Optimizing SQL Server Performance with Business Central

Monitoring Business Central Server Events

2/17/2021 • 2 minutes to read • Edit Online

You can monitor events on Business Central Server to diagnose conditions and troubleshoot problems that affect operation and performance.

Event Logging Overview

Business Central uses Event Tracing for Windows (ETW), which is a subsystem of Windows operating systems. ETW provides a tracing mechanism for events that are raised by an application or service. ETW enables you to use industry standard tools such as Windows Performance Monitor, PerfView, Event Viewer, and Windows PowerShell to dynamically collect data on trace events that occur on the Business Central Server.

Events that occur on Business Central Server instances are recorded in Windows Event logs on the Business Central Server computer. Dynamics 365 Business Central uses channels on all events. Event channels provide a way to collect and view events from a specific provider, which in this case is Business Central Server, and group the events according to predefined types, such as admin, operational, and debug. For example, in Event Viewer, Business Central Server instance events are collected in the Admin, Operational, and Debug channel logs for Business Central in the Applications and Services Logs.

For more general information about ETW and event channels, see Event Tracing for Windows and Event Logs and Channels in Windows Event Log.

Monitoring Business Central Server Event Traces

Event tracing provides detailed information about what is occurring on the Business Central Server and application when users work with Business Central. This can help you identify and analyze problems or conditions that affect performance. Event tracing enables you to dynamically monitor Business Central Server without having to restart the server or Business Central clients. By using industry-standard tools for event tracing, you can start and stop event tracing sessions, and then view the trace event data from a stored log file.

You can use event tracing to track the following operations on Business Central Server instances:

- Running Business Central reports, queries, and XMLports.
- Execution of SQL statements by Business Central Server.
- Execution of AL functions.
- Telemetry.
- Windows event log events.

Event Trace Monitoring Tools

There are various industry-standard tools that you can use to collect event trace data. The procedures in this section use Windows Performance Monitor, PerfView, Event Viewer, and Windows PowerShell to illustrate how you can collect and view event trace data. For details about how to use these tools and others, refer to the documentation available with the tool. For an overview of some of the tools, see Tools for Monitoring Performance Counters and Events.

Get Started

TASK	FOR MORE INFORMATION, SEE
Review the list of trace events that are available for monitoring Business Central Server instances.	Business Central Trace Events List
Collect event trace data in an event trace log (.etl) file. Use the event trace monitoring tool to start an event trace session.	Use Performance Monitor to Collect Event Trace Data Use PerfView to Collect Event Trace Data Use Logman to Collect Event Trace Data
View event trace data that is contained in an .etl file.	Use PerfView to View Event Trace Data
Use Event Viewer to collect and view events	Monitoring Business Central Server Events by Using Event Viewer
Use Windows PowerShell to view event trace data	Monitoring Business Central Server Events by Using Windows PowerShell
Turn off or limit the amount of telemetry trace events emitted based on the severity level.	Turn Off or Limit Telemetry Trace Events

See Also

Business Central Server Trace Events
Business Central Server Admin and Operational Events

Business Central Server Trace Events

2/17/2021 • 8 minutes to read • Edit Online

This article provides an overview of the trace events that are generated by Business Central server instance.

Overview

- There are two event trace providers that publish different trace events to the event log: **Microsoft-DynamicsNAV-Server** and **Microsoft-DynamicsNAV-Common**. The **Microsoft-DynamicsNAV-Common** provider is strictly for telemetry trace events. All other events use **Microsoft-DynamicsNAV-Server**. You typically need to specify the event trace provider in the monitoring tool that you are using.
- There are several types of trace events for each event trace provider, including: Windows event viewer, SQL traces, service calls, AL function calls, and telemetry. Trace event types are identified by a keyword with a corresponding decimal and hexadecimal value. The keywords and values enable you to collect data on specific trace events. Some tools support the hexadecimal values only and other tools support both hexadecimal and decimal.
- For some trace events, there is separate event for when an operation starts and when it stops. This enables you to determine the duration of the operation. Some monitoring tools, such as PerfView, will automatically return the duration in the stop event.
- Some monitoring tools might interpret and display the collected event trace differently than others. For more information, see Event Trace Data.

Windows Event Viewer Trace Events

Windows Event Viewer trace events track errors, warnings, and information messages that provide information about the condition or state of Business Central Server instances. These events can be viewed in the **DynamicsNAV** channel logs and **Application** log of Event Viewer on the computer that is running Business Central Server. For more information, see Monitoring Business Central Server Events Using Event Viewer.

EVENT TRACE PROVIDER	KEYWORD	DECIMAL VALUE	HEXADECIMAL VALUE
Microsoft-DynamicsNAV-Server	EventViewer	1	0x1

For a list and description of EventViewer trace events, see Business Central Server Admin and Operational Events.

SQL Trace Events

SQL trace events track a specific set of SQL statements that are executed from the Business Central Server instance against the Business Central database on SQL Server.

EVENT TRACE PROVIDER	KEYWORD	DECIMAL VALUE	HEXADECIMAL VALUE
Microsoft-DynamicsNAV-Server	SQLTracing	2	0x2

The event data that is collected includes: session ID, tenant ID, the Business Central user, and the SQL statement.

For more information, see Event Trace Data.

The following table lists the SQL trace events.

ID	EVENT (TASK/OPCODE)	WHAT IS TRACED
1	ExecuteScalar/Start	SQL statements that query a database table and return a single field from a row in the query result.
2	ExecuteScalar/Stop	SQL statements that query a database table and return a single field from a row in the query result.
3	ExecuteNonQuery/Start	SQL statements that return a number of rows from a database table
4	ExecuteNonQuery/Stop	SQL statements that return a number of rows from a database table
5	ExecuteReader/Start	SQL statements that return a set of rows from a database table.
6	ExecuteReader/Stop	SQL statements that return a set of rows from a database table.
7	ReadNextResult/Start	SQL statements that return the next result from a database query.
8	ReadNextResult/Stop	SQL statements that return the next result from a database query.
9	ReadNextRow/Start	SQL statements that return the next row in database table.
10	ReadNextRow/Stop	SQL statements that return the next row in database table.
11	BeginTransaction/Start	SQL statements that start a database transaction.
12	BeginTransaction/Stop	SQL statements that start a database transaction.
13	Prepare/Start	SQL statements that create a prepared version of the command on an instance of SQL Server.
14	Prepare/Stop	SQL statements that create a prepared version of the command on an instance of SQL Server.
15	OpenConnection/Start	SQL statements that open connection to the database from the connection pool.

ID	EVENT (TASK/OPCODE)	WHAT IS TRACED
16	OpenConnection/Stop	SQL statements that open connection to the database from the connection pool.
17	Commit/Start	SQL statements that commit a database transaction.
18	Commit/Stop	SQL statements that commit a database transaction.
19	Rollback/Start	SQL statements that cancel the changes in a pending database transaction.
20	Rollback/Stop	SQL statements that cancel the changes in a pending database transaction.

Service Call Trace Events

Service call trace events track when Business Central clients run the Business Central objects: Queries, Reports, and XMLports.

EVENT TRACE PROVIDER	KEYWORD	DECIMAL VALUE	HEXADECIMAL VALUE
Microsoft-DynamicsNAV-Server	ServiceCall	4	0x4

The event data that is collected includes: session ID, tenant ID, Business Central user, and the Business Central object ID. For more information, see Event Trace Data.

The following table lists the service call trace events.

ID	EVENT (TASK/OPCODE)	WHAT IS TRACED
300	RunQuery/Start	Business Central Query objects that are opened and closed.
301	RunQuery/Stop	Business Central Query objects that are opened and closed.
302	RunReport/Start	Business Central Report objects that are opened and closed.
303	RunReport/Stop	Business Central Report objects that are opened and closed.
310	RunXmlPort/Start	Business Central XMLport objects that are opened and closed.
311	RunXmlPort/Stop	Business Central XMLport objects that are opened and closed.

ID	EVENT (TASK/OPCODE)	WHAT IS TRACED
500	OpenSession	Dynamics NAV Client connected to Business Centrals and Business Central Web clients establish a connection to the Business Central Server instance.
501	CloseSession	Dynamics NAV Client connected to Business Centrals and Business Central Web clients establish a connection to the Business Central Server instance.

AL Methods Trace Events

AL function tracing events track the execution of AL functions and statements on the Business Central Server instance.

EVENT TRACE PROVIDER	KEYWORD	DECIMAL VALUE	HEXADECIMAL VALUE
Microsoft-DynamicsNAV-Server	ALTracing	8	0x8

The event data that is collected includes: session ID, tenant ID, Business Central user, AL function, AL statements, and line number. For more information, see Event Trace Data.

> **IMPORTANT**
>
> If the Business Central Server instance is not configured for full AL function tracing, then only root-level AL function will be traced. Statements and AL functions that are called from functions will not be traced. By default, the Business Central Server instance is not configured for full AL function tracing. For information about how to specify full AL function tracing, see Configuring Business Central Server.

The following table lists the AL function tracing events.

ID	EVENT (TASK/OPCODE)	WHAT IS TRACED				
400	ExecuteALFunction/Start	AL functions that are called.		401	ExecuteALFunction/Stop	AL functions that are called.
402	ExecuteALFunctionFailed	Errors that occur when executing AL functions. The errors can be caused by exceptions or ERROR Function (Dialog) calls.				

ID	EVENT (TASK/OPCODE)	WHAT IS TRACED				
403	ExecuteALFunction	AL statements that are executed. **Important:** This trace event is only traced when the Business Central Server is configured to full AL function tracing.				

Telemetry Trace Events

Telemetry trace events can provide data about operations in the application and how it is being used in production. These events include both system telemetry trace events and user-defined, custom telemetry trace events.

EVENT TRACE PROVIDER	KEYWORD	DECIMAL VALUE	HEXADECIMAL VALUE
Microsoft-DynamicsNAV-Common	TelemetryTracing	32	0x20

Custom telemetry trace events are emitted from the application. These are events that are sent by SENDTRACETAG method calls from inside the application. For more information about custom telemetry trace events, see Instrumenting an Application for Telemetry.

Some of the important event data that is collected for both system and custom telemetry trace events includes: tag, category, message, dataclassification. For more information about this data, see Event Trace Data.

Telemetry events can have one of the following event IDs, based on the data classification and verbosity (or seveity level):

DATA CLASSIFICATION	VERBOSITY	ID
All except CustomerContent and EndUserIdentifiableInformation	Critical	700
	Error	701
	Informational	702
	Informational	703
	Verbose	704
	Warning	705

DATA CLASSIFICATION	VERBOSITY	ID
	Informational	706
CustomerContent or EndUserIdentifiableInformation	Critical	707
	Error	708
	Informational	709
	Informational	710
	Verbose	711
	Warning	712

NOTE

Event IDs 703, 706, and 710 are used only for system telemetry trace events. All other IDs are used for both system and custom events.

Event Trace Data

The following table lists the arguments that make up the data collected for trace events. When viewing event trace data, the way that the arguments are interpreted and displayed can vary depending on the tool that you use.

ARGUMENT	DESCRIPTION	TRACE EVENT TYPE				
category	Specifies the category of the telemetry trace event.	Telemetry (TelemetryData)				
connectionType	Specifies the RoleTailored client that has established the connection to the Business Central server instance. Values include Dynamics NAV Client connected to Business Central and Business Central Web client.	Service calls (ServiceCall)				

ARGUMENT	DESCRIPTION	TRACE EVENT TYPE				
dataclassification	Specifies the RoleTailored client that has established the connection to the Business Central server instance. Values include Dynamics NAV Client connected to Business Central and Business Central Web client.	Service calls (ServiceCall)				
failureMessage	Includes the error message that is returned when a AL function fails.	AL function trace events (ALTracing)				
functionName	Specifies the AL function that was executed.	AL function trace events (ALTracing)				
lineNumber	Specifies the line number of the statement in the AL code of the Business Central object that was executed.	AL function trace events				
message	Specifies the error, warning, or information message text that was issued for a trace event	Windows event log trace events (EventViewer) Telemetry (TelemetryData)				
objectId	Specifies the ID of the Business Central object that was executed in the session.	Service calls trace events (ServiceCall) AL function trace events (ALTracing)				

ARGUMENT	DESCRIPTION	TRACE EVENT TYPE				
objectType	Specifies the Business Central object type that executed by a AL function or statement. Values include the following: CodeUnit, Page, Query, Report, Table, and XMLport.	AL function trace events (ALTracing)				
sessionId	Specifies the ID that is assigned to the session that is used by the operation. Each operation establishes a session with the Business Central Server instance from a connection in the Business Central Server's connection pool.	All				
sqlStatement	Specifies the SQL statement that was executed on the session.	SQL trace events (SQLTracing)				
statement	Specifies the AL statement that was executed on the session.	AL function trace events		tag	Specifies the ID of the telemetry trace event. For system telemetry trace events, this ID is autogenerated. For custom telemetry trace events, it is user-defined.	Telemetry (TelemetryData)

ARGUMENT	DESCRIPTION	TRACE EVENT TYPE				
tenantId	Specifies the ID of the tenant database that is mounted on the Business Central Server instance. If the Business Central Server instance is not configured for multitenancy, then the value is empty. For more information about multinenancy, see Multitenant Deployment Architecture.	All				
userName	Specifies the Business Central user account that is logged on to the session.	All				

See Also

Monitoring Business Central Server Events
Classifying Data

Business Central Server Admin and Operational Events (EventViewer) List

2/17/2021 • 11 minutes to read • Edit Online

Events have the source `BusinessCentralServer$[ServerInstance]`. Each event has a unique ID and is assigned to a task category. The source, IDs, and task categories enable you to filter the events that display in Event Viewer. For a description of the task categories, see Task Categories.

The following table lists the events that are generated by the Business Central Server.

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS					
112	Error	12	Operational	Message: Fatal Sql error. The connection can no longer be used. Remarks: An error occurred on the connection from the Business Central Server instance to the SQL database and the connection could not been established. This					

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				error could be caused by one of the following reasons: - The Business Central Server has been stopped. - The SQL server connection settings are incorrect - A network failure has occurred. - A hardware failure has occurred on the server or on your computer. To resolve this issue, try to restart the Business Central Server or see Troubleshooting: A fatal						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				fatal error occurred. The connection to SQL server cannot be established.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
113	Information	12	Operational	Message: Object Change Listener is listening on SQL Server '{0}'in Database'{1}'. Remarks: Occurs when the Business Central Server instance has established a connection to the SQL database. The Change Listener object listens for changes to application objects in the Business Central database.		200	Error	12	Admin	Remarks: This event ID is used for various errors that occur on Business Central Server instances. These events indicate that the Business Central Server instance is not operating. View the details of each message to determine the cause of the problem.

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
201	Information	12	Operational	Remarks: This event ID is used for various information messages that occur on Business Central Server instances. These events are typical conditions and are for information only.						
202	Warning	12	Admin	Remarks: This event ID is used for various warning messages that occur on Business Central Server instances. These events indicate that						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				an unexpected condition occurred on the Business Central Server instance. In most cases, the Business Central Server instance will still be operational. View the details of each message to determine the cause of the problem.						
205	Information	8	Operational	Message: 'Microsoft Dynamics NAV Data Service' is listening to requests at net.tcp://[Server]:[Port]/[ServerInstance]/OData Remarks:						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				Indicates that the listening port for OData web services has been opened on the Business Central Server instance and it is ready to handle OData requests. Typically, this condition occurs when the Business Central Server instance starts.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
206	Information	11	Operational	Message: 'Microsoft Dynamics NAV Data Service' at net.tcp://[Server]:[Port]/[ServerInstance]/OData has stopped. Remarks: Indicates that the listening port for OData web services on the Business Central Server instance has been closed. Typically, this condition occurs when the Business Central Server instance is stopped.						
207	Information	8	Operational	Messa						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
	ation		ional	ge: Microsoft Dynamics NAV Business Web Services' is listening to requests at net.tcp://[Server]:[Port]/[ServerInstance]/WS/Service Remarks: Indicates that the listening port for SOAP web services has been opened on the Business Central Server instance and it is ready to handle SOAP requests. Typically, this condition occurs when the Business Central Server instanc

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
				e starts.
208	Information	11	Operational	Message: 'Microsoft Dynamics NAV Business Web Services' at net.tcp://[Server]:[Port]/[ServerInstance]/WS/Services has stopped. Remarks: Indicates that the listening port for SOAP web services has been closed. Typically, this condition occurs when the Business Central Server instance is stopped.
209	Information	8	Operational	Message: 'Microsoft Dynamics NAV Service'

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				is listening for requests at net.tcp://[Server]:[Port]/[ServerInstance]/Service Remarks: Indicates that the listening port for Dynamics NAV Client connected to Business Central has opened and it is ready for Dynamics NAV Client connected to Business Central connections. Typically, this condition occurs when the Business Central Server instance starts.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
210	Information	11	Operational	Message: 'Microsoft Dynamics NAV Service' at net.tcp://[Server]:[Port]/[ServerInstance]/Service has stopped. Remarks: The listening port for Dynamics NAV Client connected to Business Central has closed. Typically, this condition occurs when the Business Central Server instance is stopped.		211	Information	12	Operational	Remarks: This event ID is used for various information messages that are generated by Microsoft Dynamics NAV Application Server (NAS) and background sessions. Because NAS does not have a user interface, events are used to provide operational information to administrators or developers.
214	Information	8	Operational	Message: 'Microsoft						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				Dynamics NAV Service' is listening to requests at net.tcp://[Server]:[Port]/[ServerInstance]/ManagementService. Remarks: Indicates that the listening port for Business Central Server Administration tool has been opened on the Business Central Server instance. Typically, this condition occurs when the Business Central Server instance starts.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
215	Information	11	Operational	Message: 'Microsoft Dynamics NAV Service' at net.tcp://[Server]:[Port]/[ServerInstance]/ManagementService has stopped. Remarks: Indicates that the listening port for the Business Central Server Administration tool has closed. Typically, this condition occurs when the Business Central Server instance is stopped.						
216	Error	13	Admin	Remarks: This event						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				ID is used for various errors that occur when the Business Central Server cannot start or establish a connection to the Business Central database on SQL Server. These events are caused by unhandled exceptions that are thrown the Business Central Server instances and indicate that an unrecoverable condition has occurred. The errors can be caused by an incorrect						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				configuration of the Business Central Server or the Microsoft SQL Server connection. To resolve errors, verify the Business Central Server and SQL Server configuration. For more information, see Configuring Business Central Server and Troubleshooting: SQL Server Connection Problems.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
217	Information	13	Operational	Remarks: This event ID is used for various information messages that occur when the Business Central Server starts and establishes a connection to the Business Central database on SQL Server. These events are caused by exceptions that are thrown by the Business Central Server instances. These events are typical conditions and are for						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
218	Warning	13	Admin	Remarks: This event ID is used for various warnings that occur when the Business Central Server starts and establishes a connection to the Business Central database on SQL Server. These events are caused by handled exceptions that are thrown by the Business Central Server instances. Typically, the Business Central Server will continue to

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				operate. But you should address the problem that is described in the event message.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
219	Information	12	Operational	Message: Microsoft Dynamics NAV Application Server for tenant '[TenantID]' is scheduled to start with the following configuration: Company: [CompanyName], Codeunit: [StartupCodeunitID], Method: [StartupMethod], Arguments: [StartupArguments] Remarks: Refers to NAS service only. Indicates that the NAS service is scheduled to start.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
220	Information	12	Operational	Message: Microsoft Dynamics NAV Application Server for tenant '[TenantID]' has completed. Remarks: Refers to NAS service only. Indicates that the NAS service has started successfully.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
221	Error	12	Admin	Message: The Microsoft Dynamics NAV Application Server session for tenant '[TenantID]' has failed and will be restarted. Reason: [Message] Remarks: Refers to NAS service only. Indicates that an exception has occurred and NAS service did not start. NAS service will attempt to start again.						
222	Error	12	Admin	Message: The Microsoft Dynamics NAV						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
			Application Server	session for tenant '[TenantID]' has permanently failed and will not be restarted. Reason: [Message] Remarks: Refers to NAS service only. Indicates that an exception has occurred and NAS service did not start. The NAS service will not be restarted because the maximum number of times that service can attempt to restart has been met. This value is specifie

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				d by the *NASServicesRetryAttemptsPerDay* in the CustomSetting.xml file for the Business Central Server instance. For more information, see Configuring NAS Services.						
223	Information	12	Operational	Message: The service is initializing its configuration. Remarks: Occurs when the Business Central Server instance has been started but is not ready for use. The Business Central Server instance is loading						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				the configuration settings that are specified in the CustomSettings.xml file. For more information about the CustomSettings.xml file, see Configuring Business Central Server.						
224	Information	12	Operational	Message: The service has completed configuration and is ready. Remarks: Occurs when the Business Central Server is started and is ready for use.		225	Information	12	Operational	Message: The service is shutting down. Remarks: Occurs when Business Central Server is stopped.
226	Information	12	Operational	Message: The NAV application was						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
				mounted from database '[DatabaseName]' on database server '[SQLServerInstance]'. Remarks: Refers to Business Central Server instances that are used in a multitenant environment. Indicates that the Business Central Server instance is connected to the Business Central application in the specified application database. Typically, this condition occurs

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				when the Business Central Server instance starts or the Mount-NAVApplication cmdlet is run.						
227	Error	12	Admin	Message: The NAV application could not be mounted for database '[DatabaseName]' on database server '[SQLServerInstance]' due to the following error: [Message]. Remarks: Refers to Business Central Server instances that are configured for multitenancy. Indicat						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				es that the Business Central Server instance cannot connect the Business Central application in the specified application database. Verify that Business Central Server is configured to use the correct application database. For more information, see Migrating to Multitenancy.						
228	Information	12	Operational	Message: Tenant '[Tenant]' was mounted from database '[DatabaseName]' on database server						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				'[SQLServer\Instance] '. Remarks: Refers to the Business Central Server instances that are configured for multitenancy. Indicates that the Business Central Server is connected to the tenant that is in the specified tenant database. Typically, this condition occurs when the Business Central Server instance starts or when the Mount-NAVTenant cmdlet is run.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
229	Error	12	Admin	Message: Tenant '[TenantID]' could not be mounted due to the following error: [Message] Remarks: Refers to Business Central Server instances that are configured for multitenancy. Occurs when the Business Central Server instance cannot connect to the tenant database.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
230	Information	12	Operational	Message: Tenant '[Tenant]' was dismounted. Remarks: Refers only to Business Central Server instances that are used in a multitenant environment. Typically, this condition occurs when the [Dismount-NAVTenant cmdlet is run.						
231	Error	12	Admin	Remarks: This event ID is used for various errors that occur when authenticating a Business Central user who is						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
				trying to log on to the Business Central Server from a RoleTailored client. This event is caused by an error in the authentication system. This event is only relevant when the Business Central Server instance is configured for NavUserPassword or AccessControlService credential types.

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
232	Information	12	Operational	Message: A user successfully authenticated against the server. Remarks: This event occurs when a user successfully logs on to the Business Central Server instance from a RoleTailored client. This event is only relevant when the Business Central Server instance is configured for NavUserPassword or AccessControlService credential types.						
232	Information	12	Operational	Message:						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				A user provided invalid credentials. Authentication was not successful. Remarks: This event occurs when a user provides an invalid user name or password when the user logs on to the Business Central Server instance from a RoleTailored client. This event is only relevant when the Business Central Server instance is configured for NavUserPassword or AccessControlService						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	credential MESSAGE/REMARKS
233	Information	12	Admin	Message: The session attempted to write to table '[Table Name]', but the write operation was rejected because it exceeds the optional table limit of the license. The license only permits writing to [Number] optional tables per session. The session has already written to the following tables: '[Table Name]', '[Table Name]', 'and [Table Name]'. Remarks:

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				This event pertains to the limited user license that is used on the Business Central solution. A limited user license specifies how many optional tables a session can write to. For more information about licensing for Business Central, see Microsoft Dynamics ERP Licensing Guide.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
700-706	Critical, Error, Warning, Information	33	Admin	Telemetry events. You can configure the lowest level of telemetry events to be recorded in the event log by changing the Diagnostic Trace Level setting in the Business Central Server instance configuration. For more information, see Configuring Business Central Server.						
1000	Error	12	Operational	Certificate monitoring has permanently failed and will not be restarted. Reason						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	: IMESSAGE/REMARKS						
				Remarks: An unhandled exception occurred that prevents the certificate from being monitored. **Note:** Event IDs 1000 through 1006 refer to the security certificate that is used by the Business Central Server instance to protect communications with client or web services. For more information, see Using Security Certificates.						
1001	Information	12	Operational	Message: Configuration						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
				setting 'ClientServices CertificateThumbprint' has been updated. It will not take effect until the service is restarted. Remarks: Occurs when the security certificate that is used by Business Central Server has been replaced. The thumbprint is set by the *ClientServicesCertificateThumbprint* parameter in the CustomSetting.xml file for the Business Central Server.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
1002	Error	12	Operational	Message: The service certificate is valid from [Date] to [Date] only. Remarks: Occurs when the security certificate that is used by the Business Central Server is not valid for use on the current date.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
1003	Warning	12	Operational	Message: The service certificate is close to its expiration date. Remarks: Occurs for the first time 30 days before the expiration date of the security certificate that is used on the Business Central Server, and then one time each day until the certificate is replaced or renewed.						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS						
1006	Information	12	Operational	Message: Configuration setting 'ClientServicesCertificateThumbprint' has been updated. It will not take effect until the service is restarted. Remarks: Occurs when a new security certificate is applied on Business Central Server. The thumbprint is set by the *ClientServicesCertificateThumbprint* parameter in the CustomSetting.xml file for the Business						

EVENT ID	LEVEL	TASK CATEGORY	CHANNEL	MESSAGE/REMARKS
				Central Server.

Task Categories

Task categories logically classify events according to the operations that they perform. In Event Viewer, you can sort, include, or exclude events in the Windows Application log based on the task categories. A task category is defined by a decimal number. The following table lists the task categories that are associated with Business Central Server events.

TASK CATEGORY	DESCRIPTION
8	Service connects to the Business Central Server instance.
11	Service disconnects from the Business Central Server instance.
12	Information, warning, or error message from the Business Central Server instance.
13	Exception thrown by Business Central Server.

See Also

Monitoring Business Central Server Events Using Event Viewer
Monitoring Business Central Server Events

Monitoring Business Central Server Events Using Event Viewer

2/17/2021 • 5 minutes to read • Edit Online

Events that occur on the Business Central Server instances can be recorded in event logs on the computer that is running Business Central Server. You can view the events by using Event Viewer.

About Business Central Server Events in Event Viewer

Events that occur on Business Central Server instances are recorded in the event channels specific to Business Central and also in the general Windows Application log. Event channels provide a way to collect and view events from a specific event trace provider. This differs from the Windows Application log which contains system-wide events from multiple publishers (applications and components).

Business Central channel logs

In the Event Viewer console tree, open **Applications and Services Logs** > **Microsoft** > **DynamicsNAV**.

Server folder

The **Server** folder contains events from the event trace provider called **Microsoft-DynamicsNAV-Server**. The events are recorded in the following logs:

LOG	DESCRIPTION
Admin	Includes events that target end users and IT administrators. These events typically indicate a problem that requires action to resolve the problem. An example of an admin event is a tenant database failing to mount on the Business Central Server instance. For a list and description of these events, see Business Central Server Admin and Operational Events.
Operational	Includes events that provide information about an operation that occurred on Business Central Server instances. These events are typically ordinary operating events that do not require any action but can be used to analyze and diagnose a problem. An example of an operational event is the shutting down of the Business Central Server instance. For a list and description of these events, see Business Central Server Admin and Operational Events.
Debug	Includes the trace event types: SQL (SQLTracing), service calls (ServiceCalls), and AL function calls (ALTracing). For more information about the different trace events and others ways to monitor them, see Business Central Server Trace Events and Monitoring Business Central Server Events . **Note:** In Event Viewer, this log is hidden and disabled by default. For information about how to show and enable this log, see Enable Business Central Debug Logs in Event Viewer.

Common folder

The **Common** folder contains telemetry events from the event trace provider called **Microsoft-DynamicsNAV-Common**. This folder contains strictly telemetry events, which have IDs 700-707. The telemetry events are recorded in the following logs:

LOG	DESCRIPTION
Admin	Includes custom telemetry trace events that are emitted from the application. These are events that are sent by SENDTRACETAG method calls from inside the application. For more information, see Instrumenting an Application for Telemetry. **Note** The Business Central Server instance includes a configuration setting called **Diagnostic Trace Level** (`TraceLevel` in the customsettings.config file) that enables you to specify the lowest severity level of telemetry events to be recorded in the event log, or even turn off telemetry event logging altogether. If you do not see the expected events, then verify the Business Central Server instance configuration with an administrator. For information, see Configuring Business Central Server.
Operational	Not applicable.
Debug	Includes system telemetry trace events that occur. **Note:** In Event Viewer, this log is hidden and disabled by default. For information about how to show and enable this log, see Enable Business Central Debug Logs in Event Viewer.

Application log

The Application log includes admin and operational type events (errors, warnings, and information messages) that occur on the Business Central Server instance.

To view the **Application** log, in the console tree, choose **Windows Logs**, **Applications**.

The events in this log are the same events that are recorded in the **Admin** and **Operation** logs in the **DynamicsNAV** > **Server** channel. Therefore, you can consider the **Application** log to be a secondary log for these events. Unless you are using System Center Operations Manager to monitor Business Central Server events, you can disable logging Business Central Server events to the Windows Application log and rely on **Applications and Services Logs** instead. For more information, see Disable Logging Events to the Windows Application Log.

> **NOTE**
> Trace events are not included in this log.

Filtering Dynamics Server Events in Event Viewer

By default, the Business Central Server logs contain events of all levels (error, warning, and information) for all Business Central Server instances. You can use the filtering functionality that is available in Event Viewer to display only Business Central Server instance events that meet specific criteria. For example, if you have several

Business Central Server instances, you can filter logs to show only events from a specific Business Central Server instance. For more information, see the following example.

Example

Your Business Central Server is running several instances that are configured with multiple tenants. In Event Viewer, you want to view only errors that occurred in the last 24 hours on the tenant *MyTenant1* of the Business Central Server instance *MyNavServerInstance1*.

To filter the event log

1. For example, in the console tree of Event Viewer, choose **Applications and Services Logs** > **Microsoft** > **DynamicsNAV** > **Server**.
2. Select the **Admin** log.
3. In the **Action** pane, choose **Filter Current Log**.

   The **Filter Current Log** window opens.
4. On the **Filter** tab, set the **Logged** drop-down list to **Last 24 hours**.
5. In the **Error Level** section, select the **Error** check box.
6. Choose the **XML** tab.

   XML similar to the following is displayed:

   ```
   <QueryList>
     <Query Id="0" Path="Microsoft-DynamicsNAV-Server/Admin">
       <Select Path="Microsoft-DynamicsNAV-Server/Admin">
         *[System[(Level=2) and TimeCreated[timediff(@SystemTime) <= 604800000]]]
     </Query>
   </QueryList>
   ```

   `Microsoft-DynamicsNAV-Server` indicates that Business Central Server is the provider of the events in the log.
7. Select the **Edit** query manually check box, and then choose the **Yes** button.
8. In the `<Select Path="Microsoft-DynamicsNAV-Server/Admin">` element, after `*[System[(Level=2) and TimeCreated[timediff(@SystemTime) <= 86400000]]]`, add the following lines:

   ```
   and
   *[EventData[Data[@Name='tenantId'] and Data  = 'MyTenant1']]
   and
   *[EventData[Data[@Name='serverInstanceName'] and Data='MyNavServerInstance1']]
   ```

   The complete XML should look similar to the following XML:

```
<QueryList>
  <Query Id="0" Path="Microsoft-DynamicsNAV-Server/Admin">
    <Select Path="Microsoft-DynamicsNAV-Server/Admin">
      *[System[(Level=2) and TimeCreated[timediff(@SystemTime) <= 604800000]]]
      and
      *[EventData[Data[@Name='tenantId'] and Data  = 'MyTenant1']]
      and
      *[EventData[Data[@Name='serverInstanceName'] and Data='MyNavServerInstance1']]
    </Select>
  </Query>
</QueryList>
```

9. Choose the **OK** button.

The **Admin** log displays only errors that occurred in the last 24 hours on tenant *Tenant1* and Business Central Server instance *MyNavServerInstance1*. The applied filter can be removed. Alternatively, you can save it as a custom view. For more information about filtering in Event Viewer, see Filter Displayed Events and Advanced XML filtering in the Windows Event Viewer.

See Also

Monitoring Business Central Server Events
Business Central Server Trace Events
Monitoring Business Central Server
Monitoring Business Central Server Using Performance Counters
Windows Event Viewer

How to: Use Performance Monitor to Collect Event Trace Data

2/17/2021 • 3 minutes to read • Edit Online

This topic describes how to use Windows Performance Monitor to collect event trace data for Business Central Server. To collect trace event data, you create a Data Collector Set, and then start the Data Collector Set.

Create a Data Collector Set for collecting Business Central trace event data

1. Start Windows Performance Monitor.
   - Choose **Start**, in the **Search** box, type **perfmon**, and then choose the related link.
2. In the navigation tree, expand **Data Collector Sets**, right-click **User-defined**, choose **New**, and then choose **Data Collector Set**.
3. On the **Create new Data Collector Set Wizard** page, enter a name for the new data collector set, select **Create manually (Advanced)**, and then choose the **Next** button.
4. On the **What type of data do you want to include** page, select the **Event trace data** check box, and then choose the **Next** button.
5. On the **Which event trace providers would you like to enable** page, choose the **Add** button to add a provider.
6. In the **Event Trace Providers** list, select **Microsoft-DynamicsNAV-Server**, and then choose the **OK** button.
7. If you want to collect data for all trace events, choose the **Next** button. If you want to collect data on specific trace events, do the following:
   a. In the **Properties** list, select **Keywords (Any)**, and then choose the **Edit** button.
   b. On the **Property** page, in the **Manual** box, type the keyword decimal value for the trace event. For a list of keyword values for trace events, see Business Central Server Trace Events.

      For example, if you want to collect data on service call trace events, then type **4**. If you want to collect data on more than one trace event, add the keyword values for each trace event and then use the sum in the **Manual** box. For example, if you want to collect data on service calls (keyword decimal value = 4) and AL functions (keyword decimal value = 8), then use the value **12**.

      > **NOTE**
      > Performance Monitor will automatically convert the value to hexadecimal, such as 0x4 or 0xC. You can also enter the keyword hexadecimal values directly.

   c. Choose the **OK** button, and then **Next** button.
8. On the **Where would you like the data to be stored** page, set the **Root directory** box to the folder where you want to save the event trace log file that is generated when you run the Data Collector Set.
9. Choose the **Finish** button to complete the wizard

The new Data Collector Set appears under **User Defined** in the navigation pane.

Complete the next procedure to increase trace buffer settings to make sure that events are not dropped when collecting trace event data.

Change the Data Collector Set trace buffers

1. In the navigation pane, select the new Data Collector Set.
2. In the main pane, right-click the **DataCollector01** item, and then choose **Properties**.
3. In the **Properties** dialog box, choose the **Trace Buffers** tab.
4. Set the following properties.

PROPERTY	RECOMMENDED MINIMUM VALUES
Buffer size	128
Minimum buffers	50
Maximum buffers	50

   You might have to adjust these properties based on the monitoring sessions and expected number of events that will be collected. If a large number of events are collected, then the trace buffer size and count might have to be increased.

5. Choose the **OK** button to save and close the **Properties** dialog box.

To start and stop a Data Collector Set

- To start to collect data, right-click the Data Collector Set, and then choose **Start**.
- To stop collecting data, right-click the Data Collector Set, and then choose **Stop**.

For information about how to schedule a time to start and stop collecting data, see Schedule Data Collection in Windows Performance Monitor.

The collected event trace data is stored in an event trace log (.etl) file in the location that you specified. You can view the data in the log file by using various industry-standard tools, such as PerfView. For information about how to use PerfView to view the event trace data, see Use PerfView to View Event Trace Data.

See Also

Monitoring Business Central Server Events
Use PerfView to View Event Trace Data
Instrumenting an Application for Telemetry

How to: Use PerfView to Collect Event Trace Data

2/17/2021 • 2 minutes to read • Edit Online

This topic describes how to use PerfView to collect event trace data for Business Central Server. When you collect event trace data, the data is stored in an event trace log (.etl) file in a location that you choose.

To install PerfView

- Go to https://go.microsoft.com/fwlink/?LinkID=313428, and then follow the instructions to download and install PerfView.

To collect event trace data

1. Open PerfView.exe.
2. On the **Collect** menu, choose **Collect**.

   The **Collecting data over a user specified interval** dialog box appears.
3. Set the **Data file** field to the path and name of the log file in which to store the trace event data. The file name must have the .etl file name extension.
4. Choose **Advanced options**.

   The upper part of the **Advanced options** area includes check boxes and fields that specify the providers from which to collect event trace data.
5. In the **Additional providers** field, type **Microsoft-DynamicsNAV-Server**.
   - If you want to filter on a specific trace event, include a colon after **Microsoft-DynamicsNAV-Server**, followed by the hexadecimal keyword value for the trace event. For example, to collect trace events data on service call trace events only, then type **Microsoft-DynamicsNAV-Server:0x4**.
   - If you want to collect data on more than one trace event, add the keyword values for each trace event and then use the sum in the field. For example, if you want to collect data on service calls (keyword value = 0x4) and AL function traces (keyword value = 0x8), then type **Microsoft-DynamicsNAV-Server:0xC** in the field. In hexadecimal, the sum of 0x4 and 0x8 is 0xC.
6. Clear the check boxes above the **Additional providers** field for any providers that you do not want to collect data for.
7. To start recording data, choose the **Start Collection** button.
8. To stop recording data, choose the **Stop Collection** button.

The collected event trace data is stored in an event trace log (.etl) file in the location that you specified. You can view the data in the log file by using various industry-standard tools, such as PerfView. For information about how to use PerfView to view the event trace data, see Use PerfView to View Event Trace Data.

To view event trace data from an event trace log file

1. Open PerfView.exe.
2. In PerfView, use the left pane to locate the .etl file that you want to view.

   The left pane displays the current directory and the files that PerfView is set up to browse. To change a

directory, choose a subdirectory from the list or type the directory (for example, c:\PerfLogs) in the text box at the top of the pane.

3. Double-click the .etl file that you want to view.

   Several items appear in the left pane under the .etl file that you selected.

4. To view the event traces, double-click **Events**.

   The **Events** window opens to display the contents of the .etl file. Trace events are listed in the left pane.

5. To view details about a trace event, double-click the trace event.

See Also

Monitoring Business Central Server Events
Business Central Server Trace Events
Instrumenting an Application for Telemetry

How to: Use LogMan to Collect Event Trace Data

2/17/2021 • 2 minutes to read • Edit Online

This article describes how to use logman to collect event trace data for Business Central Server. Logman (logman.exe) comes with the Windows Operating System. You can use it to create and manage event trace session and performance logs from the command prompt.

This article provides a brief introduction to using logman to collect trace event data for Business Central Server and telemetry events. For more detailed information about logman, see Logman.

Collect event trace data

You can collect Business Central Server trace event data from two different trace event providers: **Microsoft-DynamicsNAV-Server** and **Microsoft-DynamicsNAV-Common**. **Microsoft-DynamicsNAV-Server** is used for trace events like SQL traces, AL function traces, and session calls. **Microsoft-DynamicsNAV-Common** is used for telemetry events.

Data that is collected with logman is stored in an event trace log (.etl) file.

The following steps give you an example of how to use logman.

1. Open the command prompt, and change to the directory that contains the `logman.exe` file.

   This is typically `C:\Windows\System32`

2. At the command prompt, run one of the following commands to create a trace data collector.

   For telemetry trace events:

   ```
   logman create trace MyTelemetryTraceData -p Microsoft-DynamicsNAV-Common -o
   c:\perflogs\MyTelemetryTraceData.etl
   ```

   For server trace events:

   ```
   logman create trace MyServerTraceData -p Microsoft-DynamicsNAV-Server -o
   c:\perflogs\MyServerTraceData.etl
   ```

   These commands will create event log files named `MyTelemetryTraceData.etl` and `MyServerTraceData.etl` in the `c:\perflogs` folder of your computer.

3. To start the trace session, run one of the following commands.

   For telemetry trace events:

   ```
   logman start MyTelemetryTraceData
   ```

   For server trace events:

   ```
   logman start MyServerTraceData
   ```

4. To stop the trace session, run one of the following commands.

For telemetry trace events:

```
logman stop MyTelemetryTraceData
```

For server trace events:

```
logman stop MyServerTraceData
```

The data is now stored in an .etl file.

View trace event data

There are various industry tools available for viewing data in .etl files.

For example, from the command line, you can use the tracerpt command to create dump files, summary, and report files. The following code creates files for the MyTelemetryTraceData_000001.etl file:

```
tracerpt c:\perflogs\MyTelemetryTraceData_000001.etl -o c:\perflogs\MyTelemetry-dmp.xml -of XML -summary
c:\perflogs\MyTelemetry-summary.txt -report c:\perflogs\MyTelemetry-rpt.xml
```

You can also use PerView. For more information, see Use PerfView to View Event Trace Data.

See Also

Monitoring Business Central Server Events
Business Central Server Trace Events
Instrumenting an Application for Telemetry

Monitoring Business Central Server Events with PowerShell

2/17/2021 • 3 minutes to read • Edit Online

Events that occur on the Business Central Server instances are recorded in event logs on the computer that is running Business Central Server. You can view the events by using Windows PowerShell as described in this article.

PowerShell Get-WinEvent Cmdlet

You can use the Get-WinEvent cmdlet of Windows PowerShell to view Business Central Server instance events and trace events in the event logs and event tracing log files on the Business Central Server computer. The Get-WinEvent cmdlet retrieves the same events that can be viewed in Event Viewer under **Applications and Services Logs** > **Microsoft** > **DynamicsNAV** (see Monitoring Business Central Server Events Using Event Viewer).

The Get-WinEvent cmdlet includes several parameters that enable you to filter the events that you view and specify how the events are displayed. Windows PowerShell enables you can create scripts that perform complex operations for extracting and displaying specific event data. For more information about the Get-WinEvent cmdlet, see Get-WinEvent.

For more information about installing and getting started with Windows PowerShell, see Getting Started with Windows PowerShell.

To use the Get-WinEvent Cmdlet to view events

1. If you want to view events in a **Debug** log, ensure that the log is enabled. The **Admin** and **Operational** logs are enabled by default.

   For information, see To enable the Business Central Server Debug Log from Windows PowerShell.

2. On the computer that is running Business Central Server, start Window PowerShell.

   For more information, see Starting Windows PowerShell.

3. At the command prompt, enter the `Get-WinEvent` command. The following table provides some simple example commands.

TO VIEW	COMMAND
Events in the all **DynamicsNAV** > **Server** logs	`Get-WinEvent -ProviderName Microsoft-DynamicsNAV-Server`
Events in the all **DynamicsNAV** > **Common** logs	`Get-WinEvent -ProviderName Microsoft-DynamicsNav-Common`
Events in the **DynamicsNAV** > **Server** > **Admin** log	`Get-WinEvent -LogName Microsoft-DynamicsNAV-Server/Admin`
Events in the **DynamicsNAV** > **Common** > **Admin** log	`Get-WinEvent -LogName Microsoft-DynamicsNav-Common/Admin`

TO VIEW	COMMAND
Events in the Business Central Server Operational log	`Get-WinEvent -LogName Microsoft-DynamicsNAV-Server/Operational`
Trace events in the Business Central Server Debug log	`Get-WinEvent -LogName Microsoft-DynamicsNAV-Server/Debug -Oldest`

To enable the Debug Logs from Windows PowerShell

There are two debug logs for Business Central: **Microsoft-DynamicsNAV-Server/Debug** and **Microsoft-DynamicsNav-Common/Debug**.

1. On the computer that is running Business Central Server, start Window PowerShell as an administrator.
2. At the command prompt, run the following commands:

```
wevtutil.exe set-log "Microsoft-DynamicsNAV-Server/<Debug>" /q:true /e:true
```

```
wevtutil.exe set-log "Microsoft-DynamicsNav-Common/<Debug>" /q:true /e:true
```

TIP

You can also enable the Debug log from Event Viewer. For more information, see Enable Analytic and Debug Logs.

Filtering Business Central Server Events

You can filter the events that you view in a Business Central Server log by setting the *FilterXpath* parameter of the Get-WinEvent cmdlet. The following examples illustrate how you can use the *FilterXpath* parameter to filter the Business Central Server events.

Example 1

The following example uses the Get-WinEvent cmdlet to view errors in the Business Central Server Admin log for the tenant *MyTenant1* on the server instance *MyNavServerInstance1*.

```
Get-WinEvent -LogName 'Microsoft-DynamicsNAV-Server/Admin' -FilterXPath "*[System[(Level=2)]] and *
[EventData[Data[@Name='tenantId'] and (Data = 'MyTenant1')]] and *
[EventData[Data[@Name='serverInstanceName'] and Data='MyNavServerInstance1']]" | Format-List -Property
Message-
```

Example 2

The following is an example of a Windows PowerShell script that you can create and run to view trace events in the Business Central Server Debug log. The script returns the start and stop AL function trace events that take more than four seconds to execute on the tenant *MyTenant1* of the server instance *MyNavServerInstance1*.

```
$maxAllowedSeconds = 4

$xPath = "*[System[(EventID = 400 or EventID = 401)]] and " +
         "*[EventData[Data[@Name='tenantId'] and (Data  = 'MyTenant1')]] and " +
         "*[EventData[Data[@Name='serverInstanceName'] and Data='MyNavServerInstance1']]"

$events = Get-WinEvent -LogName 'Microsoft-DynamicsNAV-Server/Debug' -FilterXPath $xPath -Oldest -MaxEvents
10000

Write-Host "List of AL functions that took more than $maxAllowedSeconds  seconds to execute :" -
ForegroundColor DarkYellow

for($i = 0; $i -lt  $events.Length; $i+=2)
{
   $seconds = ($events[$i + 1].TimeCreated - $events[$i].TimeCreated).Seconds

   if ($seconds -ge $maxAllowedSeconds )
   {
     Write-Host $events[$i].Message `r`n -ForegroundColor Magenta
   }
}
```

You can create the script by using, for example, Notepad or Windows PowerShell Integrated Scripting Environment (ISE). You save the script as .ps1 file type, and then run it from the Windows PowerShell.

See Also

Monitoring Business Central Server Events
Business Central Server Trace Events
Monitoring Business Central Server
Monitoring Business Central Server Using Performance Counters
Event Viewer

Turn Off or Limit Telemetry Trace Events

2/17/2021 • 2 minutes to read • Edit Online

The application and platform can emit many telemetry trace events, which can be collected using various event trace tools. For example, telemetry trace events are recorded in the Business Central Server channel logs, which you can see in Event Viewer, under **Applications and Services Logs** > **Microsoft** > **DynamicsNAV** > **Common** > **Admin**.

The number of events can place a large demand on the logging resources on the computer running the Business Central Server instance. To help alleviate this demand, the Business Central Server instance includes a configuration setting called **Diagnostic Trace Level** (`TraceLevel` in the customsettings.config file) that enables you to specify the lowest severity level of customer telemetry trace events that are emitted from the application, or even turn off telemetry events altogether. Custom telemetry trace events have IDs from 700-712.

To configure the **Diagnostic Trace Level** setting, you can use the Business Central Server Administration tool, modify the Business Central Server instance configuration file (CustomSettings.config) directly, or use the Set-NAVServerConfiguration cmdlet of the Business Central Administration Shell.

> **TIP**
> Custom telemetry events are generated by calls to the SENDTRACETAG method in code. For more information, see Instrumenting an Application for Telemetry.

Use the Business Central Server Administration tool

1. To start the Business Central Server Administration tool, select **Start**, and in the **Search programs and files** box, type **Microsoft Dynamics365 Business Central Administration**, and then choose the related link.
2. In the left pane, under **Console root**, select the Business Central Server instance.
3. In the center pane, select the **Edit** button.
4. Under **General**, set the **Diagnostic Trace Level**:

   You use this setting to filter out lower-level events from being emitted. For example, if you set this setting to **Error**, only **Error** and **Critical** events will be emitted.

   Set to **Off** if you do not want to emit telemetry trace events.
5. Select the **Save** button, and then select the **OK** button.

   You must restart the Business Central Server instance for the changes to take effect.
6. To restart, the Business Central Server instance, in the left pane, select the Business Central computer.

   Unless you are administering a remote computer, this is Business Central (local).
7. In the center pane, right-click an instance, and then select **Restart**.

Modify the CustomSettings.config file

1. Open the CustomSettings.config file for the Business Central Server instance in a text editor, such as Notepad.

By default, the file is located in the C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service folder or C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Instances\<instancename> folder (for multitenant installations).

2. Set the **TraceLevel** setting to **Critical**, **Error**, **Warning**, **Normal** (this corresponds to the **Information** level), **Verbose**, or **Off**.
3. Save the file, and then restart the Business Central Server instance.

Use the Business Central Administration Shell

1. Start the Business Central Administration Shell.
2. At the command prompt, run the following command:

```
Set-NAVServerConfiguration -ServerInstance MyServerInstance -KeyName TraceLevel -KeyValue level -
ApplyTo All
```

Substitute `MyServerInstance` with the name of the Business Central Server instance and `level` with either `Critical`, `Error`, `Warning`, `Normal`, `Verbose`, or `Off`.

For more information about how to use the Business Central Administration Shell, see Business Central PowerShell Cmdlets and Set-NAVServerConfiguration Cmdlet.

See Also

Monitoring Business Central Server Events Using Event Viewer
Monitoring Business Central Server Events
Configuring Business Central Server

Monitoring and Analyzing Long Running SQL Queries

2/17/2021 • 2 minutes to read • Edit Online

Microsoft Dynamics NAV 2017 was the first version that allows long running SQL queries to be logged to the Windows Event Log. The queries are logged when the application communicates with the database and the call to the database takes too long. Starting in Business Central 2019 release wave 2, long running queries can also be emitted as telemetry to Microsoft Azure Application Insights. Using Application Insights requires that you first enable it on your tenant.

Defining the long running SQL queries threshold

The time logged when a SQL query runs is the time spent on the called database as seen from the server. There are multiple reasons that can cause a delay. For example, a delay happens when the database waits for a lock to release. Or it runs an operation that's missing indexes.

The threshold of when a SQL query is considered to be long running is controlled by the Business Central Server configuration setting **SqlLongRunningThreshold**. The default value is 1000 milliseconds (ms). By default, the threshold is set to 1000 milliseconds. In this case, if a SQL query runs longer 1000 ms, a message is recorded in the event log and emitted as telemetry. The message indicates that the action took longer than expected or longer than the given threshold. For more information about setting the **SqlLongRunningThreshold** by using Business Central Server Administration tool, see Configuring Business Central Server.

You can also change the setting by Set-NAVServerConfiguration cmdlet in Business Central Administration Shell. The cmdlet includes the `-ApplyTo Memory` parameter that enables you to change the setting without doing a server restart. For example, to change the threshold dynamically to 2000 ms, run the Business Central Administration Shell as Administrator and then type the following PowerShell cmdlet:

```
Set-NAVServerConfiguration -ServerInstance <ServerInstanceName> -KeyName SqlLongRunningThreshold -KeyValue
2000 -ApplyTo Memory
```

Monitor and analyze data

To use the Windows Event Log, see Troubleshooting: Using the Event Viewer to Monitor Long Running SQL Queries.

To set up and use Application Insights, see Enabling Application Insights for Tenant Telemetry.

See Also

Troubleshooting: Using the Event Log to Monitor Long Running SQL Queries
Troubleshooting: Analyzing Long Running SQL Queries Involving FlowFields by Disabling SmartSQL
Monitoring and Analyzing Telemetry
Set-NAVServerConfiguration
Tools for Monitoring Performance Counters and Events
Monitoring Business Central Server Using Performance Counters
Monitoring Business Central Server Events

Optimizing SQL Server Performance with Business Central

2/17/2021 • 2 minutes to read • Edit Online

The following articles describe how to optimize performance in Dynamics 365 Business Central when accessing data from the SQL Server database.

Setting SQL Compatibility Level to Optimize Database Performance

Data Access

Table Keys and Performance

Bulk Inserts

AL Database Methods and Performance on SQL Server

Query Objects and Performance

Configuring Query Hints for Optimizing SQL Server Performance with Business Central

Using Read Scale-Out for Better Performance

Troubleshooting: Analyzing Long Running SQL Queries Involving FlowFields by Disabling SmartSQL

Troubleshooting: Using Query Store to Monitor Query Performance in Business Central

Troubleshooting: Using the Event Log to Monitor Long Running SQL Queries in Business Central

See Also

Installation Considerations for Microsoft SQL Server
Microsoft SQL Server documentation
SumIndexField Technology (SIFT)

Setting SQL Compatibility Level to Optimize Database Performance

2/17/2021 • 2 minutes to read • Edit Online

If your Business Central database is running on Azure SQL Database or SQL Server 2016 or later, set the database's compatibility level to match the database server. This will equip the database with the latest optimization features of Azure SQL Database or SQL Server. This is particularly relevant for demonstration databases that are installed by using the Business Central Setup because the default compatibility level matches SQL Server 2014.

To change the compatibility level

You change the compatibility level of the database by using SQL Server Management Studio. There are two ways to do this:

- Open the database properties, select the **Options** page, and then set the **Compatibility Level**:.

  For more information, see View or Change the Compatibility Level of a Database.

- Run the following query:

  ```
  ALTER DATABASE <database name> SET COMPATIBILITY_LEVEL = { 140 | 130 }
  ```

  where:

  - `<database name>` is the name of the database to be modified.
  - `140` sets the database to be compatible with SQL Server 2017
  - `130` sets the database to be compatible with SQL Server 2016 and Azure SQL Database

  For more information, see ALTER DATABASE (Transact-SQL) Compatibility Level.

> **NOTE**
> The compatibility level for Azure SQL Database is subject to change. Refer to Azure SQL Database documentation for latest compatibility level.

See Also

Optimizing SQL Server Performance
Microsoft SQL Server documentation'

Data Access

2/17/2021 • 9 minutes to read • Edit Online

Data that is needed in the client goes through the following path from the Business Central Server to the SQL Server database:

1. If the data is cached in the Business Central Server data cache, it is returned.
2. If the data is not cached in the Business Central Server data cache, it is fetched from SQL Server over the network as follows:
   a. If the data resides in SQL Servers data cache, it is returned.
   b. If the data does not reside in SQL Servers data cache, it is fetched from storage and returned.

Business Central Server data caching

In Business Central, the data cache is shared by all users who are connected to the same Business Central Server instance. This means that after one user has read a record, a second user who reads the same record gets it from the cache.

The following AL functions utilize the cache system:

- GET
- GETBYSYSTEMID
- FIND
- FINDFIRST
- FINDLAST
- FINDSET
- COUNT
- ISEMPTY
- CALCFIELDS

There are two types of caches, global and private:

- Global cache is for all users connected to a Business Central Server instance.
- Private cache is per user, per company, in a transactional scope. Data in a private cache for a given table and company is flushed when a transaction ends.

The cache that is used is determined by the lock state of a table. If a table is not locked, then the global cache is queried for data; otherwise, the private cache is queried.

Results from query objects are not cached.

For a call to any of the **FIND** functions, 1024 rows are cached. You can set the size of the cache by using the **Data Cache Size** setting in the Business Central Server configuration file. The default size is 9, which approximates a cache size of 500 MB. If you increase this number by one, then the cache size doubles.

You can bypass the cache by using the SELECTLATESTVERSION method (Database).

Business Central synchronizes caching between Business Central Server instances that are connected to the same database. By default, the synchronization occurs every 30 seconds.

You can set the cache synchronization interval by using the *CacheSynchronizationPeriod* parameter in the CustomSettings.config file. This parameter is not included in the CustomSetting.config file by default, so you

must add it manually using the following format:

```
<add key="CacheSynchronizationPeriod" value="hh:mm:ss" />
```

For example, to set the interval to 50 seconds, set the `value` to `"00:00:50"`. For more information about the CustomSettings.config file, see Configuring Business Central Server.

Business Central Server connections to SQL Server

Starting from Microsoft Dynamics NAV 2013, the Business Central Server uses ADO.NET to connect to the SQL Server database. Installations of Microsoft Dynamics NAV 2009 and earlier uses ODBC to connect to the SQL Server database.

The ADO.NET interface is a managed data access layer that supports SQL Server connection pooling, which can dramatically decrease memory consumption by Business Central Server. SQL Server connection pooling also simplifies deployment of the Business Central three-tier architecture for deployments where the three tiers are installed on separate computers. Specifically, administrators are no longer required to manually create SPNs or to set up delegation when the client, Business Central Server, and SQL Server are on separate computers.

There is no longer a one-to-one correlation between the number of client connections and the number of SQL Server connections. In earlier versions of Business Central, each SQL Server connection could consume up to 40 MB of memory. Additionally, memory allocation is now in managed memory, which is generally more efficient than unmanaged memory.

Records are retrieved using Multiple Active Result Sets (MARS). methods such as NEXT, FIND('-'), FIND('+'), FIND('>'), and FIND('<') are generally faster with MARS than the server cursors that earlier versions of Business Central used.

Data read/write performance

AL functions COUNT and AVERAGE formulas can use SIFT indexes. For more information, see CALCSUMS method (Record) and CALCFIELDS method (Record). MIN and MAX formulas use SQL Server MIN and MAX functions exclusively.

RecordIds and SQL Variant columns in a table do not prevent the use of BULK inserts. For more information, see Bulk Inserts.

In most cases, filtering on FlowFields issues a single SQL statement. In earlier versions of Business Central, filtering on FlowFields issued an SQL statement for each filtered FlowField and for each record in the table in order to calculate the filtered FlowFields. The exceptions in Business Central in which filtering on FlowFields does not issue a single SQL statement are as follows:

- You use the ValueIsFilter option on a field and the field has a value.
- A second predicate is specified on a source field and the field that is used for the second predicate has a value. For example, when you specify the CalcFormula Property for a FlowField, you can specify table filters in the **Calculation Formula** window. If you specify two or more filters on the same source field, then filtering does not issue a single SQL statement.

In most cases, calling the FIND or NEXT functions after you have set the view to include only marked records issues a single SQL statement. In earlier versions of Business Central, calling FIND or NEXT functions that have marked records issued an SQL statement for each mark. There are some exceptions if many records are marked. For more information, see MARKEDONLY method (Record).

Using SQL Server table partitioning

As of Microsoft Dynamics NAV 2018, the use of SQL Server table and index partitioning is a supported configuration. The data of partitioned tables and indexes is divided into units that can be spread across more than one filegroup in a SQL Server database. All partitions of a single index or table must reside in the same database. The table or index is treated as a single logical entity when queries or updates are performed on the data. Prior to SQL Server 2016 SP1, partitioned tables and indexes were not available in every edition of SQL Server. Partitioning large tables or indexes can have the following manageability and performance benefits:

- You can perform maintenance operations on one or more partitions more quickly. The operations are more efficient because they target only these data subsets, instead of the whole table. For example, you can choose to rebuild one or more partitions of an index.
- You might be able to improve query performance, based on the types of queries you frequently run and on your hardware configuration. When SQL Server performs data sorting for I/O operations, it sorts the data first by partition. SQL Server accesses one drive at a time, and this might reduce performance. To improve data sorting performance, stripe the data files of your partitions across more than one disk by setting up a RAID (redundant array of independent disks). In this way, although SQL Server still sorts data by partition, it can access all the drives of each partition at the same time.
- You can use partitioning to distribute parts of tables to different IO sub systems. For example, you could archive data for old transactions on slow and inexpensive disks and keep current data on solid-state drives (SSD). You can improve performance by enabling lock escalation at the partition level instead of a whole table. This can reduce lock contention on the table.

For more general information about partitioned tables and indexes in SQL Server, see Partitioned Tables and Indexes.

How Business Central supports partitioning

If you have altered tables in a Business Central database to make them partitioned tables, the synchronization engine, which is responsible for mapping the logical metamodel to physical tables, will respect this configuration during upgrades. After a schema upgrade, even if tables have been dropped and recreated, the partitioning strategy applied to the original tables will be added to the upgraded tables. You can create a partitioned table or index in SQL Server by using SQL Server Management Studio or Transact-SQL.

> **NOTE**
> For partitioning to work, the partition column must be part of the clustering key on the table.

Table Partioning Example

This example uses Transact-SQL to change table **G_L Entry** to be partitioned on the **Posting Date** field, with data partitioned on the year, and where all partitions are aligned to the PRIMARY file group.

1. In SQL query editor, create a partition function that creates partitions that divide on year (this can be used for partitioning multiple tables):

```
CREATE PARTITION FUNCTION [DataHistoryPartitionmethod] (datetime)
AS RANGE LEFT FOR VALUES (
'20151231 23:59:59.997',
'20161231 23:59:59.997',
'20171231 23:59:59.997',
'20181231 23:59:59.997'  )
GO
```

2. Create a partition scheme that maps partitions to file groups. In this example, all partitions are mapped to the PRIMARY file group (this can be used for partitioning multiple tables):

```
CREATE PARTITION SCHEME DataHistoryPartitionScheme
AS PARTITION DataHistoryPartitionmethod ALL TO ([PRIMARY])
GO
```

3. In the Dynamics NAV Development Environment, add the **Posting Date** field to the primary key.

   For more information, see Table Keys.

4. In the Transact-SQL Editor, partition table **G_L Entry** by using the previously defined partition scheme:

```
ALTER TABLE [dbo].[G_L Entry]
DROP CONSTRAINT [G_L Entry$0]
GO

ALTER TABLE [dbo].[G_L Entry]
ADD CONSTRAINT [G_L Entry$0] PRIMARY KEY CLUSTERED
(
[$companyId], [Entry No_], [Posting Date]
)
ON DataHistoryPartitionScheme( [Posting Date] )
GO
```

TIP

SQL Server Management Studio includes the **Create Partition Wizard** to help you create partitioning functions, partitioning schemes, as well as changing a table to be partitioned. For more information, see Create Partitioned Tables and Indexes.

Using SQL Server data compression

As of Business Central April 2019, it is possible to configure data compression directly in table metadata by using the CompressionType property in AL or CSIDE. Previously, compression could only be configured in SQL Server. You use data compression to help reduce the size of selected tables in the database. In addition to saving space, data compression can help improve performance of I/O-intensive workloads because the data is stored in fewer pages and queries will read fewer pages from disk. This is especially useful if your storage system is based on disks and not SSD.

However, extra CPU resources are required on the database server to compress and decompress the data while data is exchanged with the Business Central Server.

With the **CompressionType** property, you can configure row or page type compression or configure the table not to use compression. With these compression settings, Business Central table synchronization process will make changes to the SQL Server table, overwriting the current compression type, if any. You can choose to control data compression directly on SQL Server by setting the **CompressionType** property to **Unspecified**, in which case table synchronization process will not control the data compression.

To evaluate whether a table is a good candidate to compress, you can use the stored procedure `sp_estimate_data_compression_savings` in SQL Server. For more information, see sp_estimate_data_compression_savings (Transact-SQL).

Because SQL Server supports data compression on the partition level, you can combine SQL Server data compression with table partitioning (see the previous section) to achieve flexible data archiving on historical parts of a large table, without having the CPU overhead on the active part of the table.

> **NOTE**
>
> Prior to SQL Server 2016 SP1, compression was not available in every edition of SQL Server.

For more general information about table compression in SQL Server, see Data Compression. For guidance on strategy, capacity planning, and best practices for data compression, see Data Compression: Strategy, Capacity Planning and Best Practices.

See Also

Query Objects and Performance
GetBySystemId(Guid)

Table Keys and Performance in Business Central

2/17/2021 • 2 minutes to read • Edit Online

When you write AL code that searches through a subset of the records in a table, you must consider what keys are defined for the table and write code that optimizes for the keys. For example, the entries for a specific customer are usually a small subset of a table containing entries for all the customers.

Defining Keys to Improve Performance

The time that it takes to complete a loop through a subset of records depends on the size of the subset. If a subset cannot be located and read efficiently, then performance deteriorates.

To maximize performance, you must define the keys in the table that support the code that you run. You must then specify these keys correctly in your code.

For example, to retrieve the entries for a specific customer, you apply a filter to the Customer No. field in the Cust. Ledger Entry table. To run the code efficiently on Microsoft SQL Server, you must define a key in the table that has Customer No. as the first field.

The table could have the following keys:

- Entry No.
- Customer No.,Posting Date

The following is an example of code that finds a subset of records.

```
SETRANGE("Customer No.",'1000');
IF FIND('-') THEN
REPEAT
UNTIL NEXT = 0;
```

SQL Server automatically chooses which index to use in order to retrieve data in the most efficient way. SQL Server calculates the cost of retrieving data using different indexes and then chooses the path that has the smallest cost. For Business Central, that calculation is based only on the statistical distribution of values in a column.

For example, if a table contains 1000 rows and a column in the table contains either the value 0 or the value 1, then that column is said to have a low selectivity. If instead a column contained the values ranging from 1 to 500 then the column is said to have a high selectivity. In the following code example, SQL Server chooses an index that contains the HighSelectivityColumn and then sorts the rows by the LowSelectivityColumn.

```
SETCURRENTKEY("LowSelectivityColumn");
SETFILTER("LowSelectivityColumn",'1');
SETFILTER("HighSelectivityColumn",'777');
FIND('-')
```

See Also

Data Access
Bulk Inserts

Bulk Inserts

2/17/2021 • 2 minutes to read • Edit Online

By default, Business Central automatically buffers inserts in order to send them to Microsoft SQL Server at one time.

By using bulk inserts, the number of server calls is reduced, thereby improving performance.

Bulk inserts also improve scalability by delaying the actual insert until the last possible moment in the transaction. This reduces the amount of time that tables are locked; especially tables that contain SIFT indexes.

Application developers who want to write high performance code that utilizes this feature should understand the following bulk insert constraints.

Bulk Insert Constraints

If you want to write code that uses the bulk insert functionality, you must be aware of the following constraints.

Records are sent to SQL Server when the following occurs:

- You call `COMMIT` to commit the transaction.
- You call `MODIFY` or `DELETE` on the table.
- You call any `FIND` or `CALC` methods on the table.

Records are not buffered if any of the following conditions are met:

- The application is using the return value from an `INSERT` call; for example, "`IF (GLEntry.INSERT) THEN`".
- The table that you are going to insert the records into contains any of the following:
  - BLOB fields
  - Fields with the **AutoIncrement** property set to **True**

The following code example cannot use buffered inserts because it contains a `FIND` call on the **GL/Entry** table within the loop.

```
IF (JnlLine.FIND('-')) THEN BEGIN
  GLEntry.LOCKTABLE;
  REPEAT
    IF (GLEntry.FINDLAST) THEN
      GLEntry.NEXT := GLEntry."Entry No." + 1
    ELSE
      GLEntry.NEXT := 1;
    // The FIND call will flush the buffered records.
    GLEntry."Entry No." := GLEntry.NEXT ;
    GLEntry.INSERT;
  UNTIL (JnlLine.FIND('>') = 0)
END;
COMMIT;
```

If you rewrite the code, as shown in the following example, you can use buffered inserts.

```
IF (JnlLine.FIND('-')) THEN BEGIN
  GLEntry.LOCKTABLE;
  IF (GLEntry.FINDLAST) THEN
    GLEntry.Next := GLEntry."Entry No." + 1
  ELSE
    GLEntry.Next := 1;
  REPEAT
    GLEntry."Entry No.":= GLEntry.Next;
    GLEntry.Next := GLEntry."Entry No." + 1;
    GLEntry.INSERT;
  UNTIL (JnlLine.FIND('>') = 0)
END;
COMMIT;
// The inserts are performed here.
```

Disabling Bulk Inserts

Disabling bulk inserts can be helpful when you are troubleshooting failures that occur when inserting records. To disable bulk inserts, you set the *BufferedInsertEnabled* parameter in the CustomSettings.config file of the Business Central Server to **FALSE**. For more information, see Configuring Business Central Server.

See Also

Data Access
Table Keys and Performance
AL Database Methods and Performance on SQL Server
Query Objects and Performance

AL Database Methods and Performance on SQL Server

2/17/2021 • 5 minutes to read • Edit Online

This topic describes the relationship between basic database methods in AL and SQL statements.

Get, Find, and Next

The AL language offers several methods to retrieve record data. In Dynamics 365 Business Central, records are retrieved using multiple active result sets (MARS). Generally, retrieving records with MARS is faster than with server-side cursors. Additionally, each function is optimized for a specific purpose. To achieve optimal performance you must use the method that is best suited for a given purpose.

- **Record.Get** is optimized for getting a single record based on primary key values.
- **Record.Find** is optimized for getting a single record based on the primary keys in the record and any filter or range that has been set.
- **Record.Find('-')** and **Record.Find('+')** are optimized for reading primarily from a single table when the application might not read all records. Find('-') is implemented by issuing a self-tuning TOP X call, where X can change over time, based on statistics of the number of rows read.

  The following are examples of scenarios in which you should use the Find('-') function to achieve optimal performance:

  - Before you post a general journal batch, you must check all journal lines for validity and verify that all lines balance. After the first line when an error is found, you do not have to retrieve the rest of the rows.
  - if you want to fulfill multiple outstanding orders from a recent purchase but you do not know how many orders are covered by the purchase.
- **Record.FindSet(ForUpdate, UpdateKey)** is optimized for reading the complete set of records in the specified filter and range. The *UpdateKey* parameter does not influence the efficiency of this method in Dynamics 365 Business Central, such as it did in Microsoft Dynamics NAV 2009.

  FindSet is not implemented by issuing a TOP X call.
- **Record.FindFirst** and **Record.FindLast** are optimized for finding the single first or last record in the specified filter and range.
- **Record.Next** can be called at any time. However, if **Record.Next** is not called as part of retrieving a continuous result set, then Business Central calls a separate SQL statement in order to Find the Next record.

Dynamic result sets

Any result set that is returned from a call to the Find methods discussed in the previous section is dynamic. That means that the result set is guaranteed to contain any changes that you make further ahead in the result set. However, this feature comes at a cost. if any modifications are made to a table which is being traversed, then Business Central might have to issue an extra SQL statement to guarantee that the result set is dynamic.

The following code shows how records are most efficiently retrieved. **FindSet** is the most efficient method to use because this example reads all records.

```
if FindSet then
  repeat
    // Insert statements to repeat.
  until Next = 0;
```

CalcFields, CalcSums, and Count

Each call to **CalcFields**, **CalcField**, **CalcSums**, or **CalcSums** methods that calculates a sum requires a separate SQL statement unless the client has calculated the same sum or another sum that uses the same SumIndexFields or filters in a recent operation, and therefore, the result is cached.

Each **CalcFields** or **CalcSums** request should be confined to use only one SifT index. The SifT index can only be used if:

- All requested sum-fields are contained in the same SifT index.
- The filtered fields are part of the key fields specified in the SifT index containing all the sum fields.

if neither of these requirements is fulfilled, then the sum will be calculated directly from the base table.

In Dynamics 365 Business Central, SifT indexes can be used to count records in a filter provided that a SifT index exists that contains all filtered fields in the key fields that are defined for the SifT index.

SetAutoCalcFields

It is a common task to retrieve data and request calculation of associated FlowFields. The following example traverses customer records, calculates the balance, and marks the customer as blocked if the customer exceeds the maximum credit limit. **Note:** the Customer record and associated fields are *imaginary* in the following examples.

```
if Customer.FindSet() then repeat
  Customer.CalcFields(Customer.Balance)
  if (Customer.Balance > MaxCreditLimit) then begin
    Customer.Blocked := true;
    Customer.Modify();
  end
  else if (Customer.Balance > LargeCredit) then begin
    Customer.Caution := true;
    Customer.Modify();
  end;
until Customer.Next = 0;
```

In Dynamics 365 Business Central, you can do this much faster. First, we set a filter on the customer. This could also be done in Business Central 2009, but behind the scenes the same code as mentioned earlier would be executed. In Dynamics 365 Business Central, setting a filter on a record is translated into a single SQL statement.

```
Customer.SetFilter(Customer.Balance,'>%1', LargeCredit);
if Customer.FindSet() then repeat
  Customer.CalcFields(Customer.Balance)
  if (Customer.Balance > MaxCreditLimit) then begin
    Customer.Blocked := true;
    Customer.Modify();
  end
  else if (Customer.Balance > LargeCredit) then begin
    Customer.Caution := true;
    Customer.Modify();
  end;
until Customer.Next = 0;
```

In the previous example, an extra call to CalcFields still must be issued for the code to be able to check the value of Customer.Balance. In Dynamics 365 Business Central, you can optimize this further by using the new **SetAutoCalcFields** method.

```
Customer.SetFilter(Customer.Balance,'>%1', LargeCredit);
Customer.SetAutoCalcFields(Customer.Balance)
if Customer.FindSet() then repeat
  if (Customer.Balance > MaxCreditLimit) then begin
    Customer.Blocked := true;
    Customer.Modify();
  end
  else if (Customer.Balance > LargeCredit) then begin
    Customer.Caution := true;
    Customer.Modify();
  end;
until Customer.Next = 0;
```

Insert, Modify, Delete, and LockTable

Each call to **Insert**, **Modify**, or **Delete** methods requires a separate SQL statement. if the table that you Modify contains SumIndexes, then the operations will be much slower. As a test, select a table that contains SumIndexes and execute one hundred **Insert**, **Modify**, or **Delete** operations to measure how long it takes to maintain the table and all its SumIndexes.

The **LockTable** method does not require any separate SQL statements. It only causes any subsequent reading from the table to lock the table or parts of it.

ModifyAll and DeleteAll

Using **ModifyAll** and **DeleteAll** can improve performance by limiting the amount of SQL calls needed. However, be aware that **ModifyAll** and **DeleteAll** will revert to individual calls if any of the following conditions exist:

- There is trigger code on the table.
- There are event subscribers to the following events: OnBeforeModify, OnAfterModify, OnGlobalModify, OnBeforeDelete, OnAfterDelete, OnGlobalDelete, and OnDatabaseModify.
- Security filtering is active.
- The table contains `Media` or `MediaSet` data type fields.
- There are fields that are added through companion tables.

See Also

Table Keys and Performance

Query Objects and Performance

2/17/2021 • 3 minutes to read • Edit Online

A *query* is an object in Dynamics 365 Business Central that you use to specify a set of data that you want to read from the Business Central database. You can query the database to retrieve one or more fields from a single table or multiple tables. You can specify how to join the tables in the query. You can specify totaling methods on fields, such as sums and averages. This topic describes how to design queries and table keys in the most efficient way.

FlowFields in Queries

A sub-query is automatically added to the SQL statement to retrieve each FlowField in a query. This allows Business Central to retrieve all the data in one request.

> **IMPORTANT**
>
> You cannot use a FlowField on a virtual table in a query because this cannot be converted automatically into a SQL statement.

Covering Indexes

When you use a query to select a subset of fields in a table, you should consider taking advantage of the covering index strategy. A *covering index* is the index that contains all output fields required by the operation performed on that index. A covering index data access strategy can greatly improve performance because the database must retrieve only data from the index instead of finding data by using the index and then retrieving the data in the clustered index. A covering index data access strategy can be used when the following conditions are true:

- All columns in a given data item are part of a single Business Central key.
- All columns that are used in the DataItem table filters are part of the same Business Central key.
- If two DataItems are linked, then the field on the parent DataItem that links the two DataItems (the **Reference Field** on the **DataItemLink** property), must be part of the same Business Central key as the columns in the child DataItem.

The SQL Server optimizer automatically chooses a covering index strategy whenever possible.

For more information about SQL Server covering indexes, see SQL Server Optimization.

For more information about SQL Server clustered and non-clustered indexes, see Types of Indexes.

Covering SIFT Indexes

Similar to how indexes can be used to retrieve data for a query, SIFT indexes can be used to retrieve data for a query that contains totals. SIFT totals are maintained after each insert, modify, or delete call, and so some or all of the totals are already calculated. A SIFT index can be used when the following conditions are true:

- The query contains at least one aggregated column with **Method Type** set to **Totals** and with **Method** set to either **Sum**, **Count**, or **Average**.
- If a DataItem contains an aggregated column, then all columns under that DataItem must be aggregated

columns, must use either the **Sum**, **Count**, or **Average** method, and must be part of a SumIndexField defined on a single Business Central key.

- In a query in which you have aggregations but not on all DataItems, then for the DataItems without aggregations, the columns are part of a SumIndexField.
- All non-aggregated columns under the DataItem that have aggregation are part of the key fields defined for the same SIFT index.
- All columns that are used in the DataItem table filters are part of the same Business Central key.
- If two DataItems are linked, then the field on the parent DataItem that links the two DataItems (the **Reference Field** in the **DataItemLink** property) must be part of the same Business Central key as the columns in the child DataItem.

Business Central Server automatically use a SIFT index for query objects whenever possible.

Differences Between Query and Record Result Sets

Business Central does not do any caching for query result sets. When you run a query, Business Central always gets the data directly from SQL Server.

Query result sets are not guaranteed to be dynamic, whereas record result sets are always dynamic. This means that if you insert or modify data in result set row that you have not yet looped through, then it is not guaranteed that the query result set includes those changes.

Enabling and Disabling Selected Query Hints

SQL Server query optimizer will try to select the best execution plan for SELECT, INSERT, UPDATE, and DELETE statements. Most of the time, query optimizer makes the right choice. Query hints are strategies that can be enforced by the SQL Server query processor to override any execution plan that the query optimizer might select for a query. The Business Central Server instance includes configuration settings that let you enable or disable the use of the selected query hints on the database.

For more information, see Configuring Query Hints for Optimizing SQL Server Performance with Business Central.

See Also

Query Object
Optimizing SQL Server Performance with Business Central

Using Read Scale-Out for Better Performance

2/17/2021 • 3 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

The way Business Central Server interacts with the database can broadly be categorized in two patterns:

- Business processes that read and write data, such as codeunits that run from UI pages or web services.
- Analytical workloads that only read data, such as queries, reports, or API pages.

Where business process transactions typically are frequent and small, transactions from analytical workloads typically read many data and run for a long time. Mixing these two types of transactions in the same database often lead to performance problems. The analytical transactions can cause locking issues, which in turn impose waiting time for business processes. This condition also may disrupt the data cache in the database. Data will essentially be moved from the cache that speeds up business process transactions

If you run the Business Central database in a High Availability architecture, you can use the built-in **Read Scale-Out** feature in Azure SQL Database or SQL Server to load-balance read-only workloads. **Read-Scale-Out** uses the capacity of a read-only replica instead of sharing the read-write replica (also known as the primary database). This way, read-only workloads like reports, queries, and API pages, are isolated from the main read-write workload codeunits. So they won't affect the performance of business processes. As an added bonus, read-only workloads will run on a dedicated database and their performance will likely be better.

Getting started using read scale-out

To start using read scale-out, do these three steps:

1. Enable read scale-out on the Business Central database.
2. As a developer, define the default database access intent (read-only or read-write) on selected reports or query objects.
3. If needed, overwrite the default database access intent on reports, pages of the type API, and queries at runtime.

Enable read scale-out on the Business Central database

In the Business Central Online service, read scale-out is readily available and automatically enabled on the databases.

For on-premises installations, see Configuring your Business Central database for read scale-out to learn how to enable read scale-out on the databases.

Define the default database access intent in AL code

When you develop solutions for Business Central in AL, you can set the default intended database access intent (read-only or read-write) on reports and query objects that you create. For more information, see DataAccessIntent property.

Overwrite the database access intent on reports, API pages, and queries

Read scale-out may introduce a slight delay when reading data from a database's secondary replica. The delay is caused by the way High Availability databases replicate data changes from the primary database to secondary replicas. If a delay isn't acceptable for an object, you can overwrite the default database access intent.

For more information, see Managing Database Access Intent.

FAQ about read scale-out

Which objects are supported?

Reports, API pages, and query objects.

What happens if there's a failover to the replica being used for reading?

The Business Central Server specifies database access intent as a parameter in the connection string to the database. The Business Central Server doesn't know about primary/secondary replicas. So if there's a fail-over, the Business Central Server will be redirected by the database.

Why is there a delay in when a data change is available on the secondary replicas?

When data is committed to the primary database, the transaction log entries are log-shipped to the transaction log for the secondary replicas. Then, an asynchronous transaction log "redo" operation makes the data available in the secondary databases.

What about Sandboxes?

Sandbox environments can't be enabled with read scale-out. Objects that use the **DataAccessIntent** property will compile and run. But they'll just access the primary database.

See also

Optimizing SQL Server Performance
Performance Overview

Configuring a Business Central Database for Read Scale-Out

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

In the Business Central Online service, read scale-out is readily available and automatically enabled on the databases.

For Business Central on-premises, you must do the following steps:

1. Check whether your database supports read scale-out.
2. Enable read scale-out on the database.

   If the Business Central database runs on Azure SQL Database, determine whether the performance tier of the database supports read scale-out. You can then enable it if it's supported. For more information, see Use read-only replicas to load-balance read-only query workloads in the Azure SQL Database documentation.

   If the Business Central database runs on SQL Server, determine whether your installation supports read scale-out and how to enable the feature. For more information, see Configure read-only routing for an Always On availability group in the SQL Server documentation.
3. Enable SQL read-only replica support on the Business Central Server instance.

   Business Central Server includes the **Enable SQL Read-Only Replica Support** (EnableSqlReadOnlyReplicaSupport) setting. This setting isn't enabled by default. For more information, see Configuring Business Central Server.

See also

Using Read Scale-Out for Better Performance
Optimizing SQL Server Performance
DataAccessIntent Property

Troubleshooting: Using Query Store to Monitor Query Performance in Business Central

2/17/2021 • 2 minutes to read • Edit Online

What is SQL Server Query Store?

The SQL Server Query Store feature provides you with insight on database query plan choice and performance. It simplifies database performance troubleshooting by helping you quickly find performance differences caused by query plan changes. Query Store automatically captures a history of queries, plans, and runtime statistics, and retains these for your review.

Where is SQL Server Query Store available?

SQL Server Query Store is available in SQL Server (starting with SQL Server 2016) and in Azure SQL Database.

What are the common scenarios for using the Query Store feature?

Query Store keeps a history of compilation and runtime metrics throughout query execution. With this information, you can get some answers on questions about your workload, such as:

- What was the last *n* queries executed on the database?
- What were the number of executions for each query?
- Which queries had the longest average execution time within last hour?
- Which queries had the biggest average physical IO reads in last 24 hours, with corresponding average row count and execution count?

Do you want to read more?

Please visit the SQL Server documentation for more information on setup, configuration and usage of Query Store:

Monitoring Performance By Using the Query Store

Operating the Query Store in Azure SQL Database

See Also

Installation Considerations for Microsoft SQL Server

Troubleshooting: Using the Event Viewer to Monitor Long Running SQL Queries in Business Central

2/17/2021 • 2 minutes to read • Edit Online

This article shows how you can monitor long running SQL queries in Event Viewer. You use the information determine SQL queries that are good candidates for optimization.

Resolution

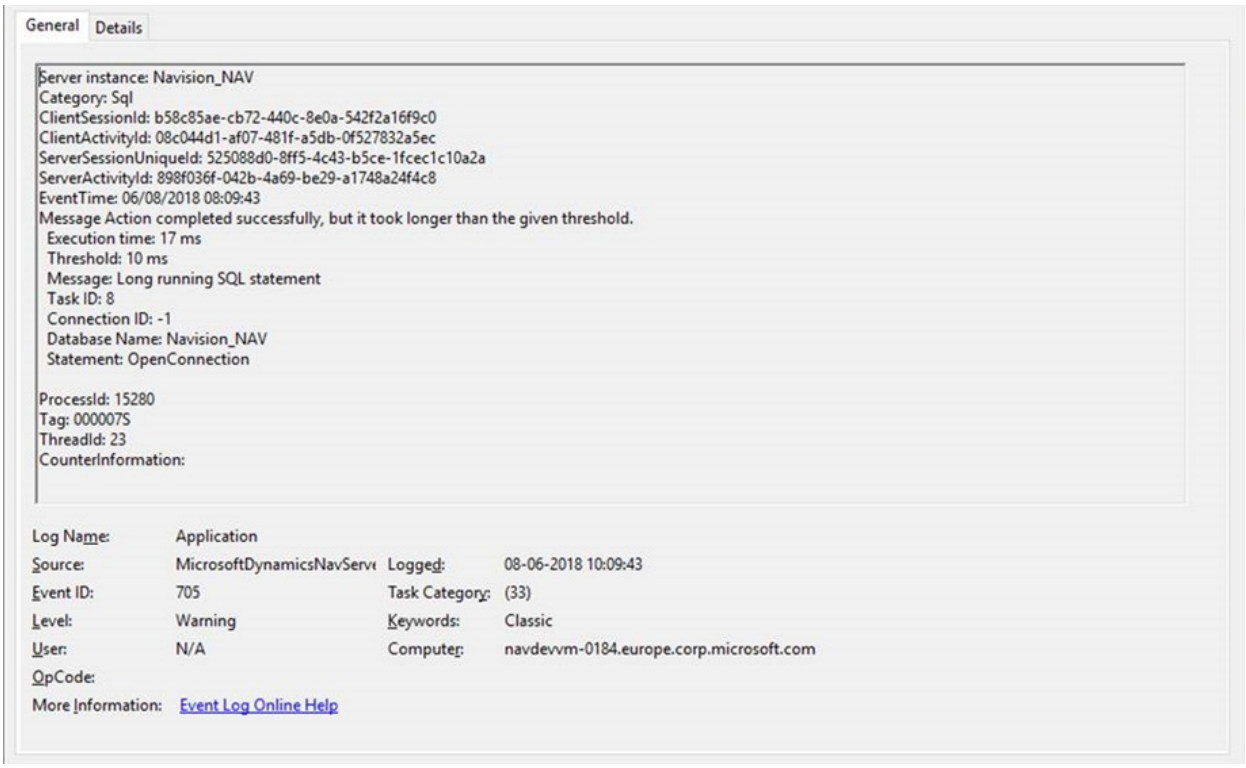
Identifying long running SQL queries can be a good starting point when doing a performance analysis. Open the Event Viewer and go to the Windows Logs Application.

> **NOTE**
>
> The SQL queries which exceed the set threshold will be displayed in the Application window of the Event Viewer as *Warning*.

If the value of the SqlLongRunningThreshold key was set to the default value of 1000 milliseconds, you'll see the message: "*Action completed successfully, but it took longer than the given threshold.*" for actions that took longer.

To meet performance expectations in production, you can set the threshold to a different value. You can set the threshold without restarting the server instance. For more information on how to set the threshold, see Monitoring Long Running SQL Queries using the Event Log.

If the value of the SqlLongRunningThreshold key was set to the default value of 1000 milliseconds, you'll see the message: "*Action completed successfully, but it took longer than the given threshold.*" for actions that took longer. To meet your performance expectations in production, you can set the threshold to a different value without doing a server restart. For more information, see Monitoring Long Running SQL Queries using the Event Log.



Using the AL call stack

Long running queries that involve the explicit execution of AL code will include an AL call stack in the event log. Long running queries that aren't the result of explicit AL code won't include an AL call stack. For example:

- GetPage calls that load data into the UI.
- Calls to triggers that don't include any code. These calls won't include an AL call stack, because there no AL code is run.

Looking at the AL call stack can help you identify what caused the delay. For queries that include an AL call stack, the generated SQL statement is listed in the `AL CallStack` column. The following code example shows which AL method generated a slow running query.

```
Server instance: BC
Category: Sql
ClientSessionId: 00000000-0000-0000-0000-000000000000
ClientActivityId: 828c9342-891a-4631-8eb3-a1da7304fdc9
ServerSessionUniqueId: 24b32889-9be9-439f-b86c-9615d5e51319
ServerActivityId: 19bf285d-a8f2-42b6-a4c0-4afe9fb5b4b4
EventTime: 06/08/2018 08:10:15
Message Action completed successfully, but it took longer than the given threshold.
  Execution time: 33 ms
  Threshold: 10 ms
  Message: Long running SQL statement
  Task ID: 3
  Connection ID: 2
  Database Name: BCDB
  Statement: SELECT "2161"."timestamp","2161"."User","2161"."Default Execute Time","2161"."Current Job Queue
Entry" FROM "SQLDATABASE".dbo."CRONUS International Ltd_$Calendar Event User Config_" "2161"  WITH(UPDLOCK)
WHERE ("2161"."User"=@0) OPTION(OPTIMIZE FOR UNKNOWN)
    AppObjectType: CodeUnit
  AppObjectId: 2160
  AL CallStack: "Calendar Event Mangement"(CodeUnit 2160).GetCalendarEventUserConfiguration line 2
"Calendar Event Mangement"(CodeUnit 2160).FindJobQueue line 1
"Calendar Event Mangement"(CodeUnit 2160).FindOrCreateJobQueue line 1
"Calendar Event Mangement"(CodeUnit 2160).CreateOrUpdateJobQueueEntry line 1
"Calendar Event"(Table 2160).Schedule line 12
"Calendar Event"(Table 2160).OnInsert(Trigger) line 1
"Calendar Event Mangement"(CodeUnit 2160).CreateCalendarEventForCodeunit line 6
"Create Telemetry Cal. Events"(CodeUnit 1352).OnRun(Trigger) line 5

ProcessId: 15280
Tag: 000007L
ThreadId: 10
CounterInformation:
```

See Also

Troubleshooting: Analyzing Long Running SQL Queries Involving FlowFields by Disabling SmartSQL
Monitoring Long Running SQL Queries using the Event Log
Tools for Monitoring Performance Counters and Events
Business Central Server Administration Tool
Troubleshooting: Using Query Store to Monitor Query Performance in Business Central
SQL Trace

Monitoring and Analyzing Telemetry

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

Business Central emits telemetry data for various activities and operations on tenants and extensions. Whether running Business Central Online or On-premises, you can set up your tenants to send telemetry to Application Insights. Application Insights is a service hosted within Azure that gathers telemetry data for analysis and presentation. For more information, see What is Application Insights?. Monitoring telemetry gives you a look at the activities and general health of your tenants. It helps you diagnose problems and analyze operations that affect performance.

Tenant-level and extension-level telemetry

Application Insights can be enabled on two different levels: tenant and extension. When enabled on the tenant, either for a Business Central online tenant or on-premises Business Central Server instance, telemetry is emitted to a single Application Insights resource for gathering data on tenant-wide operations.

With Business Central 2020 release wave 2 and later, Application Insights can also be enabled on a per-extension basis. An Application Insights key is set in the extension's manifest (app.json file). At runtime, certain events related to the extension are emitted to the Application Insights resource. This feature targets publishers of per-tenant extensions to give them insight into issues in their extension before partners and customers report them.

Available telemetry

In Application Insights, telemetry from Business Central is logged as traces. Currently, Business Central offers telemetry on the following operations:

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
App key vault secrets	Provides information about the retrieval of secrets from Azure Key Vaults by extensions.	✓ [1]	✓ [1]	✓	See...
Authorization	Provides information about user sign-in attempts. Information includes success or failure indication, reason for failure, user type, and more.	✓			See...

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
Company lifecycle	Provides information about creating, copying, and deleting of companies.	✓	✓		See...
Configuration package lifecycle	Provides information about operations done on configuration packages, including exporting, importing, applying, and deleting.	✓	✓		See...
Database lock timeouts	Provides information about database locks that have timed out.	✓	✓		See...
Email	Provides information about the success or failure of sending emails.	✓	✓		See...
Extension lifecycle [2]	Provides information about the success or failure of extension-related operations, like publishing, synchronizing, installing, and more.	✓	✓	✓	See...
Extension update	Provides information about errors that occur when upgrading an extension.	✓	✓	✓	See...
Field monitoring trace	Provides information about the usage of the field monitoring feature.	✓	✓		See...

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
Job queue	Provides information about creating and running job queue entries.	✓			See...
Long running AL method trace	Provides information about long running AL methods.	✓	✓		See...
Long running operation (SQL query)	Provides information about SQL queries that take longer than expected to execute.	✓	✓	✓	See...
Page views	Provides information about the pages that users open in the modern client.	✓			See...
Permissions	Provides information about adding, removing, and assigning permission sets.	✓			See...
Report generation	Provides information about the execution of reports.	✓	✓	✓	See...
Incoming web service requests	Provides information about the execution time of incoming web service requests.	✓	✓	✓	See...
Outgoing web service requests	Provides information about the execution time of outgoing web service requests.	✓	✓	✓	See...

AREA	DESCRIPTION	ONLINE	ON-PREMISES	EXTENSION SUPPORT	MORE INFORMATION
Web service access key authentication	Provides information about the authentication of web server access keys on web service requests.	✓	✓		See...

[1]This signal is only emitted to the Application Insights resource that's specified in the extension.

[2]Introduced in Business Central 2020 release wave 1, version 16.3. For extension telemetry, this signal was introduced in 2020 release wave 2, version 17.1.

Enabling Application Insights

Sending telemetry data to Application Insights requires you have an Application Insights resource in Azure. Once you have the Application Insights resource, you can start to configure your tenants and extensions to send telemetry data to your Application Insights resource. The configuration is different for Online and On-premises:

- For Business Central Online, Application Insights is enabled by using the Administration Center. For more information, see Enable Sending Telemetry to Application Insights.
- For Business Central On-premises, see Enable Sending Telemetry to Application Insights.
- For extensions, see Sending Extension Telemetry to Azure Application Insights.

Viewing telemetry data in Application Insights

Telemetry from Business Central is stored in Azure Monitor Logs in the *traces* table. You can view collected data by writing log queries. Log queries are written in the Kusto query language (KQL). For more information, see Logs in Azure Monitor and Overview of log queries in Azure Monitor.

As a simple example, do the following steps:

1. In the Azure portal, open your Application Insights resource.
2. In the **Monitoring** menu, select **Logs**.
3. On the **New Query** tab, type the following to get the last 100 traces:

```
traces
| take 100
| sort by timestamp desc
```

About Custom Dimensions

Each trace includes a `customDimensions` column. The `customDimensions` column, in turn, includes a set dimensions that contain metrics specific to the trace. Each of these custom dimensions has a limit of 8000 characters. If a dimension's value exceeds 8000 characters, additional dimensions will be added to the trace for containing the excess characters. There can be up to two additional parameters, each with a maximum 8000 characters. The additional dimensions will have the names that `<custom_dimension_name>_1` and `<custom_dimension_name>_2`, where `<custom_dimension_name>` is the name of the original dimension. For example, if the custom dimension is `extensionCompilationDependencyList`, then the additional dimensions would be `extensionCompilationDependencyList_1` and `extensionCompilationDependencyList_2`.

> **NOTE**
>
> The 8000 character limit is governed by the Application Insights API.

Application Insights sample code

On the Business Central BCTech repository on GitHub, samples of KQL code are available to make it easy to get started using Application Insights.

For more information, see Business Central BCTech repository on GitHub.

See also

Telemetry Event IDs
Working with Administration Tools
The Business Central Administration Center
Managing Environments
Managing Tenant Notifications
Introduction to automation APIs
LogMessage Method

Enabling Application Insights for Tenant Telemetry On-Premises

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2019 release wave 2 and later

This article describes how to set up tenants to send telemetry data to Azure Application Insights for on-premises environments.

> **IMPORTANT**
> Currently, emitting data to Azure Application Insights resources in Germany regions, like **(Europe) Germany West Central** or **(Europe) Germany North** , doesn't work. Until this issue is fixed, the mitigation is to create an Azure Application Insights resource in a region outside of Germany. Then, when the issue has been fixed, move the resource to the preferred region.

Get started

1. Create an Application Insights resource.

   See Create an Application Insights resource.

2. Get the instrumentation key of the Application Insights resource, which you can get from the Azure Portal.

Enable on tenants

Once you have the resource and its key, you can enable your tenants to send telemetry to your Application Insights resource.

The way you enable Application Insights depends on whether the Business Central Server instance is configured as a single-tenant or multitenant instance:

- For a single-tenant server instance, you set the **Application Insights Instrumentation Key** setting of the server instance. For more information, see Configuring Business Central Server.
- For a multitenant server instance, you enable this feature on a per-tenant basis when you mount tenants on the Business Central Server instance. The Mount-NAVTenant cmdlet includes the `-ApplicationInsightsKey` parameter that you set to the instrumentation key, for example:

  ```
  Mount-NAVTenant -ServerInstance BC150 -Tenant tenant1 -DatabaseName "Demo Database BC (15-0)" -
  DatabaseServer localhost -DatabaseInstance BCDEMO -ApplicationInsightsKey 11111111-2222-3333-4444-
  555555555555
  ```

Enable in Docker

If you are running Business Central in Docker using the latest BcContainerHelper, you can specify the Application Insights Key when creating the container and this will be used in server instance settings if the container is single-tenant or for the default tenant if the container is multi-tenant.

```
New-BcContainer `
    -accept_eula `
    -updateHosts `
    -artifactUrl (Get-BCArtifactUrl -country us) `
    -applicationInsightsKey "11111111-2222-3333-4444-555555555555"
```

You can specify the same or another key when creating additional tenants:

```
New-BcContainerTenant -tenantId "additional" -applicationInsightsKey "55555555-4444-3333-2222-111111111111"
```

See Also

Monitoring Long Running SQL Queries
Enable Application Insights for Online
Monitoring and Analyzing With Telemetry

Analyzing App Key Vault Secret Trace Telemetry

2/17/2021 • 6 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

App key vault telemetry gathers information about the acquisition of secrets in Azure Key Vaults by extensions at runtime. For an overview of app key vaults and secrets, see Using App Key Vaults with Business Central Extensions.

The app key vault secret process has two operations: *initialization* and *retrieval*. The telemetry data provides information about the success or failure for each of these operations. There are various conditions that cause a failure. The failure messages provide insight into the cause of the failure, helping you identify, troubleshoot, and resolve issues.

Initialization

Initialization is the first stage. It verifies the configuration of the app key vault provider in the extension and on the service. This stage is initiated by the `TryInitializeFromCurrentApp` method call in the extension code. Some conditions that cause failures in this stage include:

- The extension doesn't specify a key vault in it's app.json file.
- The Azure Key Vault Client Identity settings are incorrect. For example, it could be that the application (client) ID that you specified for the key vault reader application in Azure is wrong.
- The Business Central Server lacks permission to the private key of the Azure Key Vault client certificate.

Retrieval

Retrieval is the second stage, and occurs after a successful initialization. In this stage, the service tries to get a secret from a specified key vault. This stage is initiated by the `GetSecret` method call in the extension code. Some conditions that cause failures include:

- The secret name requested by the extension is doesn't exist or isn't valid.
- The key vault doesn't exist.
- The application ID doesn't have permission to read from the key vault.

For more information about using key vault secrets with extensions, see App Key Vaults with Business Central Extensions.

App Key Vault secret initialization succeeded

Occurs when an extension secret was successfully initialized.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault initialization succeeded: '{keyVaultUri}'.**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the request, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0014**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.
keyVaultUrls	Specifies the DNS name of the Azure key vault that was used in the request. The keyVaultUris are specified in the app.json file of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

App Key Vault initialization failed

Occurs when a key vault failed to be initialized.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault initialization failed.**

DIMENSION	DESCRIPTION OR VALUE
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed request, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0015**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.
failureReason	Specifies the error that occurred.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

App Key Vault secret retrieval succeeded

Occurs when a secret used by an extension is successfully retrieved from an Azure Key Vault.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault secret retrieval succeeded from key vault '{keyVaultUri}'.**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0016**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.
keyVaultUrl	Specifies the DNS name of the Azure key vault that was used in the request. The keyVaultUris are specified in the app.json file of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

App Key Vault secret retrieval failed

Occurs when an extension failed to retrieve a secret from a specified Azure key vault.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**App Key Vault secret retrieval failed from key vault '{keyVaultUri}'.**
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
alObjectId	Specifies the ID of the AL object that was run by request.
alObjectName	Specifies the name of the AL object that was run by request.
alObjectType	Specifies the type of AL object that was run by request.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0017**
extensionId	Specifies the AppID of the extension that requested the secret.
extensionName	Specifies the name of the extension that requested the secret.
extensionPublisher	Specifies the publisher of the extension that requested the secret.
extensionVersion	Specifies the version of the extension that requested the secret.

DIMENSION	DESCRIPTION OR VALUE
keyVaultUrl	Specifies the DNS name of the Azure key vault that was used in the request. The keyVaultUris are specified in the app.json file of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

App Key Vaults with Business Central Extensions
Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Company Lifecycle Trace Telemetry

2/17/2021 • 11 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.1

Company lifecycle telemetry gathers data about the success or failure of the following company-related operations:

- creating a company
- copying a company
- deleting a company

Failed operations result in a trace log entry that includes a reason for the failure.

Company created

Occurs when the company has been successfully created.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company created: {companyName}** `{companyName}` indicates the name of the new company.
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.

DIMENSION	DESCRIPTION OR VALUE
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0001**
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to create the company. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to create the company. The time has the format hh:mm:ss.ssssss.

Company creation canceled

Occurs when creating a company was canceled.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company creation canceled: {companyName}** `{companyName}` indicates the name of the company being created.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.

DIMENSION	DESCRIPTION OR VALUE
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0002**
failureReason	**Operation was canceled**
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server create the company before being canceled. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to create the company before being canceled. The time has the format hh:mm:ss.sssssss.

Company creation failed

Occurs when a company failed to be created.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company creation failed: {companyName}** `{companyName}` indicates the name of the company being created.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0003**
failureReason	Specifies the exception that indicates the cause of the failure.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server create the company before it failed. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to create the company before it failed. The time has the format hh:mm:ss.sssssss.

Company copied

Occurs when a company has been copied from another company successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company copied: {companyNameSource} to {companyNameDestination}** • `{companyNameSource}` is name of the company that was copied. • `{companyNameDestination}` is the name of new company that was created.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyNameDestination	Specifies the name of the new company.
companyNameSource	Specifies the name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LT0004**
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to copy the company. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.

DIMENSION	DESCRIPTION OR VALUE
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to copy the company. The time has the format hh:mm:ss.sssssss.

Company copy canceled

Occurs when a copying a company was canceled.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company copied canceled: {source company name} to {destination company name}** • `{companyNameSource}` is name of the company that was being copied. • `{companyNameDestination}` is the name of new company that was being created.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed operation, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0005**
failureReason	**Operation was canceled**.
result	**Failure**

DIMENSION	DESCRIPTION OR VALUE
serverExecutionTime	Specifies the amount of time it took the server to copy the company. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that were executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to do the operation. The time has the format hh:mm:ss.sssssss.

Company copy failed

Occurs when a company failed to be copied from another company.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company copy failed: {companyNameSource} to {companyNameDestination}** • `{companyNameSource}` is name of the company that was being copied. • `{companyNameDestination}` is the name of new company that was being created.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types

DIMENSION	DESCRIPTION OR VALUE
eventId	**LC0006**
failureReason	Specifies the exception that indicates the cause of the failure.
result	**Failure**
serverExecutionTime	Specifies the amount of time on the server. The time has the format hh:mm:ss.ssssss.
sqlExecutes	Specifies the number of SQL statements that were executed.
sqlRowsRead	Specifies the number of table rows that by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to create the company before it failed. The time has the format hh:mm:ss.ssssss.

Company deleted

Occurs when a company has been deleted successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company deleted: {companyName}** `{companyName}` indicates the name of the company that was deleted.
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0007**
result	**Success**
serverExecutionTime	Specifies the amount of time it took on server. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the operation executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to delete the company. The time has the format hh:mm:ss.sssssss.

Company deletion canceled

Occurs when deleting a company failed was canceled.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company deletion canceled: {companyName}** `{companyName}` indicates the name of the company that was being deleted.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.

DIMENSION	DESCRIPTION OR VALUE
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0008**
failureReason	**Operation was canceled**
result	**Failure**
serverExecutionTime	Specifies the amount of time it took on server. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to delete the company before being canceled. The time has the format hh:mm:ss.sssssss.

Company deletion failed

Occurs when a company failed to be deleted.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Company deletion failed: {companyName}** `{companyName}` indicates the name of the company that was being deleted.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.

DIMENSION	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0009**
failureReason	Specifies the exception that caused the failure.
result	**Failed**
serverExecutionTime	Specifies the amount of time it took on server. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that executed by the operation.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to delete the company before it failed. The time has the format hh:mm:ss.sssssss.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Configuration Package Lifecycle Trace Telemetry

2/17/2021 • 4 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, version 17.2, and later

Configuration package telemetry gathers data about the following operations on configuration packages:

- Export
- Import
- Apply
- Delete

For information about working with configuration packages, see Prepare a Configuration Package in the Business Central Application Help.

Configuration package export started

Occurs when an export operation on a configuration package is started.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package export started: {alPackageCode}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3F**
alExecutionId	Specifies the ID of the export operation.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the export of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the export of configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package being exported.

Common custom dimensions

The following table explains additional custom dimensions that are common to all configuration package traces.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**RapidStart**
alDataClassification	**SystemMetadata**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Configuration package exported successfully

Occurs when an export operation on a configuration package completes successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package exported successfully: {alPackageCode}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3G**
alExecutionId	Specifies the ID of the export operation.
alExecutionTimeInMs	Specifies the number of milliseconds it took to complete the export operation.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the export of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the export of configuration packages.
alObjectType	**CodeUnit**

DIMENSION	DESCRIPTION OR VALUE
alPackageCode	Specifies the ID of the configuration package that was exported.
See common custom dimensions	

Configuration package import started

Occurs when an import operation on a configuration package is started.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package import started: {alPackageCode}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3H**
alExecutionId	Specifies the ID of the import operation.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the import of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the import of configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package being imported.
See common custom dimensions	

Configuration package imported successfully

Occurs when an import operation on a configuration package completes successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package imported successfully: {alPackageCode}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3I**
alExecutionTimeInMs	Specifies the number of milliseconds it took to complete the import operation.
alExecutionId	Specifies the ID of the import operation.
alPackageCode	Specifies the ID of the configuration package that was imported.
alObjectId	**8614**, which is the ID of the base application codeunit that handles the import of configuration packages.
alObjectName	**Config. XML Exchange**, which is the name of the base application codeunit that handles the import of configuration packages.
alObjectType	**CodeUnit**
See common custom dimensions	

Configuration package apply started

Occurs when an apply operation on a configuration package is started.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package apply started: {alPackageCode}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3N**
alExecutionId	Specifies the ID of the apply operation.
alObjectId	**8611**, which is the ID of the base application codeunit that handles applying configuration packages.
alObjectName	**Config. Package Management**, which is the name of the base application codeunit that handles applying configuration packages.
alObjectType	**CodeUnit**

DIMENSION	DESCRIPTION OR VALUE
alPackageCode	Specifies the ID of the configuration package being applied.
See common custom dimensions	

Configuration package applied successfully

Occurs when an apply operation on a configuration package completes successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package applied successfully: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3O**
alExecutionId	Specifies the ID of the apply operation.
alExecutionTimeInMs	Specifies the number of milliseconds it took to complete the apply operation.
alErrorCount	Specifies the number of errors that occurred when applying the configuration package.
alFieldCount	Specifies the number of fields that were included in the migration table of the applied configuration package.
alObjectId	**8611**, which is the ID of the base application codeunit that applies configuration packages.
alObjectName	**Config. Package Management**, which is the name of the base application codeunit that applies configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package that was applied.
alRecordCount	Specifies the number of records that were included in the applied configuration package.
See common custom dimensions	

Configuration package deleted successfully

Occurs when a configuration package is deleted successfully.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Configuration package deleted successfully: {alPackageCode}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
eventId	**AL0000E3P**
alObjectId	**8611**, which is the ID of the base application codeunit that deletes configuration packages.
alObjectName	**Config. Package Management**, which is the name of the base application codeunit that handles deleting configuration packages.
alObjectType	**CodeUnit**
alPackageCode	Specifies the ID of the configuration package that was deleted.
See common custom dimensions	

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Prepare a Configuration Package in the Business Central

Analyzing Database Lock Timeout Trace Telemetry

2/17/2021 • 5 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.0

Database lock timeout telemetry gathers information about database locks that have timed out. The telemetry data allows you to troubleshoot what caused these locks.

In the client, when a lock has timed out, the user is presented with a message, similar to the following message:

The operation could not complete because a record in the [table name] table was locked by another user. Please retry the activity.

Two types of trace events are emitted to Application Insights:

- The first is a **Database lock timed out** event. This event includes general information about the lock request. This event includes information like the AL object and code that is impacted by the lock, the extension involved, and more.
- The **Database lock timed out** event then triggers one or more **Database lock snapshot** events. **Database lock snapshot** events provide details about SQL sessions that hold database locks at the time of lock timeout, including the session that caused the lock timeout. These events include specific details about the SQL lock request on the database, like the type, status, mode, and the table.

> **TIP**
> When analyzing database lock timeout telemetry, it's useful to look at combined data from the **Database lock timed out** event and **Database lock snapshot** events. You can combine data from different events by using *joins* in your Kusto queries. For an example, see LockTimeouts.kql in the **Microsoft/BCTech** repository on GitHub. For more general information about using joins, see Joins in Azure Monitor log queries in the Microsoft Azure documentation.

Database lock timed out

Occurs when a database lock has timed out.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Database lock timed out**
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.

DIMENSION	DESCRIPTION OR VALUE
alExecutingMethodScope	Specifies the AL action that is running the transaction that caused the lock.
alObjectId	Specifies the ID of the running AL object that requested the lock.
alObjectName	Specifies the name of the running AL object that requested the lock. not shown
alObjectType	Specifies the type of the running AL object that requested the lock, such as a page or report.
alStackTrace	The stack trace in AL.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0012**
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
extensionId	Specifies the AppID of the extension that was involved in the lock.
extensionName	Specifies the name of the extension that was involved in the lock.
extensionVersion	Specifies the version of that was involved in the lock.
sessionId	Specifies the ID of the session that requested the lock.
snapshotId	Specifies the ID of the database snapshot. This ID is used to identify associated **Database lock snapshot** trace events.

DIMENSION	DESCRIPTION OR VALUE
sqlServerSessionId	Specifies the ID of the SQL server session that requested the lock.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Database lock snapshot

Occurs when a database lock has timed out. Each **Database lock snapshot** trace event is associated with a specific **Database lock timed out** trace event.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Database lock snapshot: {snapshotId}** The value of the `{snapshotId}` maps to the `snapshotId` dimension of the **Database lock timed out** trace event that triggered this event.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alExecutingMethodScope	Specifies the AL action that is running the transaction that caused the lock.
alObjectId	Specifies the ID of the running AL object that requested the lock.
alObjectName	Specifies the name of the running AL object that requested the lock.
alObjectType	Specifies the type of the running AL object that requested the lock, such as a page or report.
alStackTrace	The stack trace in AL.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
eventId	**RT0013**
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
extensionId	Specifies the AppID of the extension that was involved in the lock.
extensionName	Specifies the name of the extension that was involved in the lock.
extensionVersion	Specifies the version of that was involved in the lock.
sessionId	Specifies the ID of the session that requested the lock.
snapshotId	Specifies the ID of the database snapshot. All messages in the snapshot share this ID.
sqlLockRequestMode	Specifies the lock mode that determines how concurrent transactions can access the resource. For more information, see Lock Modes.
sqlLockRequestStatus	Specifies the current status of the lock, which can be one of the following values: • `CNVRT` - means that the lock is transitioning from another mode, but the conversion is blocked by another process that holds a lock with a conflicting mode. • `GRANT` - means that the lock is active. • `WAIT` - means that the lock is blocked by another process that holds a lock with a conflicting mode.
sqlLockResourceType	Specifies the database resource affected by the lock. For example, `DATABASE`, `FILE`, `OBJECT`, `PAGE`, `KEY`, and more.
sqlServerSessionId	Specifies the ID of the SQL server session that requested the lock.
sqlTableName	Specifies the name of table on which the lock was held.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Email Trace Telemetry

2/17/2021 • 2 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, update 17.2, and later

Email telemetry gathers data about the following operations:

- An email was sent successfully
- An attempt to send an email failed

Before you can collect this data, you'll have to set up email. For more information, see Set Up Email in the Business Central application help.

Email sent successfully

Occurs when an email was successfully sent from the client.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Email sent successfully**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**Email**
alConnector	Specifies the email-provider connector used to send the email. Possible values include: • Current User • Microsoft 365 • SMTP • Other custom connectors installed by extensions. The connector is specified on the email accounts that are set up in Business Central. For more information, see Adding Email Accounts.
alDataClassification	**SystemMetadata**
alEmailMessageID	Specifies the GUID assigned to email, like C7A56676-9F3F-4044-90F0-D7F3196AC366.
alObjectId	**8888**, which is the ID of the system application codeunit that sends emails.

DIMENSION	DESCRIPTION OR VALUE
alObjectName	**Email Dispatcher**, which is the name of the system application codeunit that sends the emails.
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**AL0000CTV**
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Failed to send email

Occurs when an email failed to be sent from the client.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Failed to send email.**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alCategory	**Email**
alConnector	Specifies the email-provider connector used to send the email. Possible values include: • Current User • Microsoft 365 • SMTP • Other custom connectors installed by extensions. The connector is specified on the email accounts that are set up in Business Central. For more information, see Adding Email Accounts.

DIMENSION	DESCRIPTION OR VALUE
alDataClassification	**SystemMetadata**
alEmailMessageID	Specifies the GUID assigned to email, like C7A56676-9F3F-4044-90F0-D7F3196AC366.
alObjectId	**8888**, which is the ID of the system application codeunit that sends emails.
alObjectName	**Email Dispatcher**, which is the name of the system application codeunit that sends the emails.
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**AL0000CTP**
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

TIP

You can also view failed emails in the **Email Outbox** page in the Business Central client.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Extension Lifecycle Trace Telemetry

2/17/2021 • 30 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.3. For extension telemetry, this signal was introduced in 2020 release wave 2, version 17.1

Extension lifecycle telemetry gathers data about the success or failure of the following extension-related operations:

- Compiling an extension
- Synchronizing an extension
- Publishing an extension
- Installing an extension
- Updating an extension
- Uninstalling an extension
- Unpublishing an extension

Failed operations result in a trace log entry that includes a reason for the failure.

Data is gathered when the operations are done using:

- Extension Management page in the client.
- Manage Apps page in the Business Central administration center.
- Extension management PowerShell cmdlets from the Business Central Administration Shell.

Behavior overview

- Compiling, publishing, and unpublishing extensions

  Data for these operations is only recorded for extensions that are published in the tenant scope.

  - For on-premises, it includes extensions that are published by running the Publish-NAVApp cmdlet with the `-Scope Tenant` parameter.
  - For online, it includes per-tenant extensions uploaded from the **Extension Management** page in the client. It doesn't include Microsoft extensions or AppSource extensions.

- Synchronizing extensions

  - For on-premises, data for this operation is recorded when an extension is synchronized by using the Sync-NAVApp cmdlet.
  - For online, data is recorded when an extension is installed from the **Extension Management** page in the client. Or, when upgraded from the Manage Apps page in the Business Central administration center.

- Installing and uninstalling extensions

  - For on-premises, data for these operations is recorded when an extension is installed or uninstalled by using the Install-NAVApp cmdlet or Uninstall-NAVApp cmdlet. Or, when an extension is installed or uninstalled from the **Extension Management** page in the client.
  - For on-premises, data for these operations is recorded when an extension is installed or uninstalled from the **Extension Management** page in the client.

- Updating an extension
  - For on-premises, data for this operation is recorded when an extension is upgraded by using the Start-NAVAppDataUpgrade cmdlet.
  - For online, data is recorded when an extension is updated from the Manage Apps page in the Business Central administration center.
- For some operations, you might experience that certain custom dimensions aren't available.

  The reason is that custom dimensions are added to the signal gradually, as the information is retrieved. If the operation fails before the custom dimension is retrieved, it isn't included in the result.

  For example, if you try to uninstall an extension using the Ininstall-NAVApp cmdlet, and the specified extension name is wrong, the operation fails. In this case, the `extensionid` and `extensionVersion` will be excluded from the results.

ENVIRONMENT/SERVER-SIDE OPERATIONS

Extension compiled successfully

Occurs when an extension compiles successfully on the service. An extension compiles when it's published or when it's repaired by using the Repair-NAVApp cmdlet.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension compiled successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0020**

DIMENSION	DESCRIPTION OR VALUE
extensionCompilationDependencyList	Specifies details about the extensions on which the compiled extension has dependencies. **Note:** If the value exceeds 8000 characters, one or two additional dimensions will be included in the trace to cover the complete dependency list. For more information, see About Custom Dimensions.
extensionCompilationResult	**Compilation succeeded without errors or warnings.**
extensionName	Specifies the name of the extension that was compiled.
extensionId	Specifies the AppID of the extension that was compiled.
extensionPublishedAs	Specifies whether the compiled extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the compiled extension.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Extension failed to compile

Occurs when an extension failed to compile on the service. An extension compiles when it's published or when it's repaired by using the Repair-NAVApp cmdlet.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to compile: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0021**
extensionCompilationDependencyList	Specifies details about the extensions on which the compiled extension has dependencies. **Note:** If the value exceeds 8000 characters, one or two additional dimensions will be included in the trace to cover the complete dependency list. For more information, see About Custom Dimensions.
extensionCompilationResult	Specifies details about the error that occurred during compilation.
extensionName	Specifies the name of the extension that failed to compile.
extensionId	Specifies the AppID of the extension that failed to compile.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the compiled extension.
failureReason	Specifies the error that occurred when compiling the extension.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Extension published successfully

Occurs when an extension published successfully on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension published successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0014**
extensionId	Specifies the AppID of the extension that was published.
extensionIsRad	Specifies whether the extension that was RAD published. **True** indicates the extension was RAD published. **False** indicates normal publishing. RAD (Rapid Application Development) publishing is done from the AL development environment. RAD publishing is a partial publishing operation that only publishes objects application objects that have changed during development. For more information, see Working with Rapid Application Development.
extensionName	Specifies the name of the extension that published.
extensionId	Specifies the AppID of the extension that published.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the published extension.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.

DIMENSION	DESCRIPTION OR VALUE
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to publish

Occurs when an extension failed publish on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0015**
extensionId	Specifies the AppID of the extension that was published.

DIMENSION	DESCRIPTION OR VALUE
extensionIsRad	Specifies whether the extension that was RAD published. **True** indicates the extension was RAD published. **False** indicates normal publishing. RAD (Rapid Application Development) publishing is done from the AL development environment. RAD publishing only is a partial publishing operation that only publishes objects application objects that have changed during development. For more information, see Working with Rapid Application Development.
extensionName	Specifies the name of the extension that published.
extensionId	Specifies the AppID of the extension that published.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the published extension.
failureReason	Specifies the error that occurred when publishing.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension unpublished successfully

Occurs when an extension was unpublished successfully on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension unpublished successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0018**
extensionId	Specifies the AppID of the extension that was unpublished.
extensionName	Specifies the name of the extension that was unpublished.
extensionId	Specifies the AppID of the extension that was unpublished.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the unpublished extension.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to un-publish

Occurs when an extension fails to unpublish on the service.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to un-publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0019**
extensionId	Specifies the AppID of the extension that failed to unpublish.
extensionName	Specifies the name of the extension that failed to unpublish.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the unpublished extension.
failureReason	Specifies the error that occurred when unpublishing.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

TENANT OPERATIONS

Extension synchronized successfully

Occurs when an extension synchronizes successfully on the tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension synchronized successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0012**
extensionId	Specifies the AppID of the extension that was synchronized.
extensionName	Specifies the name of the extension that was synchronized.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionSynchronizationMode	Specifies whether the extension was synchronized in one of the following modes: • **Add** - The database schema defined by the objects in the extension are added to the database schema of the tenant database. This mode is typically used mode after you publish an extension for the first time. • **Clean** - The database schema defined by all versions of the extension will be removed from the database and all data is lost. This mode is typically used when an extension will no longer be used and all versions unpublished. • **Development** - This mode is acts similar to Add, except it is intended for use during development. It lets you to sync the same version of an extension that is already published. However, to run this mode, only one version the App can be currently published. • **ForceSync** - This mode like **Add** except it supports destructive schema changes (like removing fields, renaming them, changing their datatypes, and more). It is typically used during development, and is the mode used when an extension is published and installed from the AL development environment. For more information about the modes, see Sync-NAVApp cmdlet -Mode.
extensionVersion	Specifies the version of the extension was synchronized.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.ssssss.

Extension synchronized failed

Occurs when an extension fails to synchronize on the tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to synchronize: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0013**
extensionId	Specifies the AppID of the extension that failed to synchronize.
extensionName	Specifies the name of the extension that failed to synchronize.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionSynchronizationMode	Specifies whether the extension was synchronized in one of the following modes: • **Add** - The database schema defined by the objects in the extension are added to the database schema of the tenant database. This mode is typically used mode after you publish an extension for the first time. • **Clean** - The database schema defined by all versions of the extension will be removed from the database and all data is lost. This mode is typically used when an extension will no longer be used and all versions unpublished. • **Development** - This mode is acts similar to Add, except it is intended for use during development. It lets you to sync the same version of an extension that is already published. However, to run this mode, only one version the App can be currently published. • **ForceSync** - This mode like **Add** except it supports destructive schema changes (like removing fields, renaming them, changing their datatypes, and more). It is typically used during development, and is the mode used when an extension is published and installed from the AL development environment. For more information about the modes, see Sync-NAVApp cmdlet -Mode.
extensionVersion	Specifies the version of the extension was synchronized.
failureReason	Specifies the error that occurred when synchronizing the extension.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension installed successfully

Occurs when an extension installs successfully on a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0010**
extensionId	Specifies the AppID of the extension that was installed.
extensionName	Specifies the name of the extension that failed to synchronize.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.

DIMENSION	DESCRIPTION OR VALUE
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension was installed.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to install

Occurs when an extension failed to install on a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0011**
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that failed to uninstall.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension was installed.
failureReason	Specifies the error that occurred when the extension was installed.
result	**Failed**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

DIMENSION	DESCRIPTION OR VALUE
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension un-installed successfully

Occurs when an extension is successfully uninstalled from a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension un-installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
doNotSaveData	Specifies whether the uninstall operation was run with option not to save the data in database table fields that are added by the extension. When using the Uninstall-NAVApp cmdlet, this condition is set with the -DoNotSaveData switch parameter.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0016**
extensionId	Specifies the AppID of the extension that was uninstalled.
extensionName	Specifies the name of the extension that was uninstalled.

DIMENSION	DESCRIPTION OR VALUE
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension was uninstalled.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to un-install

Occurs when an extension failed to uninstall on a tenant.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to un-install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
doNotSaveData	Specifies whether the uninstall operation was run with option not to save the data in database table fields that are added by the extension. When using the Uninstall-NAVApp cmdlet, this condition is set with the -DoNotSaveData switch parameter.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0017**
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that failed to uninstall.
extensionPublishedAs	Specifies whether the extension was published as one of the following options: • **Dev** - published from the AL development environment. • **Global** - published to the global scope. • **Tenant** - published to the tenant scope.
extensionPublisher	Specifies the extension's publisher.
extensionScope	Specifies whether the extension was published to one of the following scopes: • **Global** - the extension can be installed on all tenants connected the service instance. • **Tenant** - the extension can only be installed on the tenant to which it was published. .
extensionVersion	Specifies the version of the extension that failed to uninstall.

DIMENSION	DESCRIPTION OR VALUE
failureReason	Specifies the error that occurred when the extension was uninstalled.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.ssssss.

Extension updated successfully

Occurs when an extension updates successfully on the service.

> **NOTE**
> Data is also recorded for any upgrade code that was run. For more information, see Analyzing Extension Update Trace Telemetry.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension updated successfully: {extensionName} version {extensionVersion} by {extensionPublisher}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0022**
extensionCulture	Specifies the language version for which the extension that was upgraded. The value is a language culture name, such as **en-US** or **da-DK**. If a language wasn't specified when the extension was installed, then en-US is used by default.
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that was being upgraded.
extensionPublisher	Specifies the extension's publisher.
extensionVersion	Specifies the new version of the extension being upgraded.
extensionVersionFrom	Specifies the old version of the extension being upgraded.
result	**Success**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

Extension failed to update

Occurs when an extension failed to update on the service.

> **NOTE**
> Data is also recorded for any upgrade code that was run. For more information, see Analyzing Extension Update Trace Telemetry.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension failed to update: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})**
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**LC0023**
extensionCulture	Specifies the language version for which the extension that was upgraded. The value is a language culture name, such as **en-US** or **da-DK**. If a language wasn't specified when the extension was installed, then en-US is used by default.
extensionId	Specifies the AppID of the extension that failed to uninstall.
extensionName	Specifies the name of the extension that was being upgraded.
extensionPublisher	Specifies the extension's publisher.
extensionVersion	Specifies the new version of the extension being upgraded.

DIMENSION	DESCRIPTION OR VALUE
extensionVersionFrom	Specifies the old version of the extension being upgraded.
failureReason	Specifies the error that occurred during upgrade.
result	**Failure**
serverExecutionTime	Specifies the amount of time it took the server to complete the request. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took to process the request. The time has the format hh:mm:ss.sssssss.

See also

Upgrading Extensions
Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Extension Update Trace Telemetry

2/17/2021 • 2 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1, version 16.2

Extension upgrade telemetry gathers information about failures that occur during extension upgrades. Specifically, it provides information about exceptions that are thrown by code run by upgrade codeunits. The trace events include the following information:

- The codeunit that threw the exception.
- AL stack trace.
- Exception message.

This data can help you identify, troubleshoot, and resolve issues with per-tenant and AppSource extensions that are blocking data upgrade.

Extension Update Failed: exception raised in extension

Occurs when an extension upgrade fails because of an exception in an upgrade codeunit.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Extension Update Failed: exception raised in extension {extensionName} by {extensionPublisher} (updating to version {extensionTargetedVersion})**
severityLevel	**3**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.
alStackTrace	The stack trace in AL.
companyName	The display name of the Business Central company that was used at time of execution.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0010**
extensionName	Specifies the name of the extension that was being upgraded.
extensionId	Specifies the AppID of the extension that was being upgraded.
extensionTargetedVersion	Specifies the new version of the extension being upgraded.
extensionVersion	Specifies the old version of the extension being upgraded.
failureReason	Specifies the exception that was thrown by the upgrade code. Some exception messages can contain customer data. As a precaution, Business Central only emits information that's classified as SystemMetadata. Exception messages that contain other data classifications, like customer data, are not shown. Instead, the following message is shown: "Message not shown because the NavBaseException(string, Exception, bool) constructor was used."
failureType	**DataUpdate**
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Example


```
{"Telemetry schema version":"0.3","telemetrySchemaVersion":"0.3","Component
version":"16.0.12582.0","componentVersion":"16.0.12582.0","Environment
type":"Production","environmentType":"Production","deprecatedKeys":"Company name, AL Object Id, AL Object
type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension version,
Telemetry schema version, AadTenantId, Environment name, Environment type, Component, Component
version","Company name":"CRONUS International Ltd.","AL Object Id":"0","AadTenantId":"tenant1-
1","companyName":"CRONUS International Ltd.","Client type":"Background","aadTenantId":"tenant1-
1","Component":"Dynamics 365 Business Central
Server","clientType":"Background","alObjectId":"0","extensionVersion":"5.0.0.0","component":"Dynamics 365
Business Central Server","eventId":"RT0010","extensionTargetedVersion":"5.0.0.1","Extension
version":"5.0.0.0","Extension App Id":"f0f770f4-8c9c-48de-adad-1c0281bef849","AL Stack
trace":"CustomerListExt(CodeUnit 50100).OnUpgradePerCompany line 7 - MyExtension by MyPublisher
publisher\n\"\"Upgrade Triggers\"\"(CodeUnit 2000000008).OnUpgradePerCompany line 2","Extension
name":"MyExtension","failureReason":"Attempted to divide by
zero.","extensionName":"MyExtension","alStackTrace":"CustomerListExt(CodeUnit 50100).OnUpgradePerCompany line
7 - MyExtension by MyPublisher publisher\n\"\"Upgrade Triggers\"\"(CodeUnit 2000000008).OnUpgradePerCompany
line 2","failureType":"DataUpdate","extensionId":"f0f770f4-8c9c-48de-adad-1c0281bef849"}
```


See also

Analyzing Field Monitoring Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 (update 17.1) and later

Keeping sensitive data secure and private is a core concern for most businesses. To add a layer of security, you can monitor important fields when someone changes a value. For example, you might want to know if someone changes your company's IBAN number.

To gather this data, you'll have to start field monitoring and specify the fields that you want to monitor. For more information, see Auditing Changes in Business Central - Monitoring Sensitive Fields in the Application help.

Telemetry is then logged for the following operations:

- When field monitoring is stopped or started
- When a field is added or removed for monitoring
- When a field value is changed

Sensitive field monitor status changed

Occurs when the field monitor feature is started or stopped in the company.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Sensitive field monitor status has changed to {almonitorStatus}**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alDataClassification	**SystemMetadata**
almonitorStatus	**Yes** - the field monitor feature was started. **No** the field monitor feature was stopped.
alObjectId	**1392**
alObjectName	**Monitor Sensitive Field**
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**

DIMENSION	DESCRIPTION OR VALUE
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types. This dimension isn't included for Business Central on-premises environments.
eventId	**AL0000DD3**
extensionName	Specifies the name of the base application.
extensionId	Specifies the ID of the base extension.
extensionPublisher	Specifies the publisher of the extension.
extensionVersion	Specifies the version of the base application.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Sensitive field value has changed

Occurs the value of a monitored field has changed in the company.

General dimensions

DIMENSION	DESCRIPTION OR VALUE
message	**Sensitive field value has changed: {alfieldCaption} in table {altableCaption}**
severityLevel	**1**

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.

DIMENSION	DESCRIPTION OR VALUE
alDataClassification	**SystemMetadata**
alfieldCaption	Specifies the name of the field that was changed.
altableCaption	Specifies the name of the table that the changed field is included.
alObjectId	**1392**
alObjectName	**Monitor Sensitive Field**
alObjectType	**CodeUnit**
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments. This dimension isn't included for Business Central on-premises environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types. This dimension isn't included for Business Central on-premises.
eventId	**AL0000CTE**
extensionName	Specifies the name of the base application.
extensionId	Specifies the ID of the base extension.
extensionPublisher	Specifies the publisher of the extension.
extensionVersion	Specifies the version of the base application.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

NOTE

Changes to fields in the following tables are always logged:

- Fields in the **Field Monitoring Setup** table. Many of the fields are included on the **Field Monitoring Setup** page.
- Fields in the **Change Log Setup (Field)** table. Many of the fields are included on the **Field Monitoring Worksheet** page.
- The **Contact Email** field in the **User** table.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Long Running AL Methods Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later, version 17.1

The Business Central Server server will emit telemetry about the execution time of long running AL methods, including the time spent in the database. The signal also includes a breakdown of how much time each event subscriber added to the total time. As a partner, this data gives you insight into bad performing code and enables you to troubleshoot performance issues caused by extensions.

> **NOTE**
> To collect this telemetry for Business Central on-premises, the **AL Function Timing** and **AL Function Logging Threshold - Application Insights** settings must be configured on the Business Central Server instance. For more information, see Configuring Business Central Server. With Business Central online, this telemetry is enabled with a specific threshold on a case-by-case basis by the service.

General dimensions

The following table explains the general dimensions included in the trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Operation exceeded time threshold (AL method)**
severityLevel	2

CustomDimensions

This table describes the different dimensions of a **Operation exceeded time threshold (AL method)** operation.

COLUMN (KEY)	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID when using Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alMethod	The name of the long running AL method.
alObjectId	The type of the AL object that executed the AL method.
alObjectName	The name of the AL object that executed the AL method.
alObjectType	The type of the AL object that executed the AL method.
alStackTrace	The stack trace in AL.

COLUMN (KEY)	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the AL method, such as Background or Web. For a list of the client types, see ClientType Option Type.
companyName	The display name of the Business Central company that was used at time of execution.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentType	Specifies the environment type of the Business Central solution, such as Production or Sandbox.
environmentName	Specifies the environment name of the Business Central solution, such as Production or Sandbox.
eventId	**RT0018**
executionTime	Specifies the total time that it took to execute the AL method. The value has the format hh:mm:ss.sssssss.
extensionInfo	Specifies information about individual extensions that contributed to the execution time spent in the call stack up until the point at which the long-running threshold was exceeded and the trace was emitted. The following information is included for each extension: • `id` - the ID of the extension. • `extensionName` - the name of the extension. • `extensionVersion` - the version of the extension. • `extensionPublisher` - the publisher of the extension. • `subscriberExecutionCount` - the number of event subscribers executed in this extension. • `executionTime` - the total execution time for this extension in the call stack. The value has the format hh:mm:ss.sssssss.
extensionName	Specifies the name of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.
extensionPublisher	Specifies the publisher of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.

COLUMN (KEY)	DESCRIPTION OR VALUE
extensionVersion	Specifies the version of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.
extensionId	Specifies the ID of the extension that was currently executing when the long-running threshold was exceeded and the trace was emitted.
longRunningThreshold	Specifies the time that defines a long-running AL method, after which the trace is emitted. The value has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Monitoring and Analyzing Long Running SQL Queries On-Premises
The Business Central Administration Center

Analyzing Long Running Operation (SQL Query) Telemetry

2/17/2021 • 4 minutes to read • Edit Online

A SQL query that takes longer than 1000 milliseconds to execute will be sent to your Application Insights resource.

To get a quick overview, you can go the Application Insights Overview dashboard.

> **NOTE**
> With Business Central On-premises, you can change the threshold that defines long running queries. For more information, see Defining Long Running SQL Queries Threshold.

There are multiple reasons that affect the time it takes SQL queries to run. For example, the database could be waiting for a lock to be released. Or, the database is executing an operation that does badly because of missing indexes. Sometimes you can look at the SQL statement that was generated by the code to see what caused the delay. This information is found in the **CustomDimension** data, specifically the **AL Stack Trace** column.

Dimensions in Application Insights

This table explains the columns included in long running query events emitted to Application Insights. Bold text indicates that the value of the columns is a constant. Some columns are standard for Application Insights. These columns are indicated by *Application Insights*.

COLUMN	DESCRIPTION OR VALUE
timestamp	Specifies the date and time that the long running query event occurred, such as 2019-08-20T07:23:07.9996696Z
message	Version 16.1 and later: **Operation exceeded time threshold (SQL query)** Before version 16.1: **Action took longer than expected**
severityLevel	**2** (This level indicates a warning. Long running queries are always recorded as warnings)
itemType	**trace**
customDimensions	(see table that follows)
operation_Name	**Long Running Operation (SQL Query)** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
operation_Id	Specifies the GUID assigned to the client operation. An operation is created whenever the user does something in the client, such as selecting an action.
operation_ParentId	Currently this column is the same as the operation_Id. This behavior might change in a future release.
session_Id	Specifies the GUID of the client session. When a client makes a connection to the Business Central Server instance, a session is created and assigned an ID.
client_Type	*Application Insights*
client_IP	*Application Insights*
client_City	*Application Insights*
client_StateOrProvince	*Application Insights*
client_CountryOrRegion	*Application Insights*
cloud_RoleName	Specifies the display name of Business Central tenant. For on-premises, this value is the same as the cloud_RoleInstance.
cloud_RoleInstance	Specifies the name of Business Central tenant.

COLUMN	DESCRIPTION OR VALUE
appId	*Application Insights*
appName	*Application Insights*
iKey	*Application Insights*
sdkVersion	*Application Insights*

CustomDimensions

This table describes the different dimensions of a **Long Running Operation (SQL Query)** operation.

COLUMN (KEY)	DESCRIPTION OR VALUE
extensionVersion	Specifies the version of the extension.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
environmentType	Specifies the environment type of the Business Central solution, such as Production or Sandbox.
environmentName	Specifies the environment name of the Business Central solution, such as Production or Sandbox.
extensionName	Specifies the name of the extension.
alObjectType	The type of the AL object that executed the SQL statement
alObjectName	The name of the AL object that executed the SQL statement
alStackTrace	The stack trace in AL.
companyName	The display name of the Business Central company that was used at time of execution.
extensionId	Specifies the AppID of the extension.
eventId	**RT0005** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID when using Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
clientType	Specifies the type of client that executed the SQL Statement, such as Background or Web. For a list of the client types, see ClientType Option Type.
alObjectId	The type of the AL object that executed the SQL statement.
component	Specifies the Business Central Server instance name and the platform version.
executionTime	Specifies the time that it took to execute the SQL statement**. The value has the format hh:mm:ss.sssssss.
longRunningThreshold	Specifies the amount of time that an SQL query can run before a warning event is recorded. The value has the format hh:mm:ss.sssssss. This threshold is controlled by the Business Central Server configuration setting called SqlLongRunningThreshold.
sqlStatement	Specifies the SQL statement that was executed for the long running query. The value is limited to 8192 characters. If the value exceeds 8192 characters, it will be truncated in manner that still provides the most pertinent information.
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.

** From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Example

The following code snippet shows an example of the CustomDimensions.


```
{"extensionVersion":"16.0.10962.0","telemetrySchemaVersion":"0.3","componentVersion":"15.0.40494.0","environmentType":"Production","environmentName":"Pr
Application","alObjectType":"Report","alObjectName":"Suggest Worksheet Lines","alStackTrace":"AppObjectType: Report\r\n AppObjectId: 840\r\n AL CallStac
(Report 840).DeleteEntries line 10 - Base Application by Microsoft\r\n\"Suggest Worksheet Lines\"(Report 840).\"Cash Flow Forecast - OnPostDataItem\"(Tr
Application by Microsoft\r\n\"Cash Flow Management\"(CodeUnit 841).UpdateCashFlowForecast line 32 - Base Application by Microsoft\r\n\"Cash Flow Forecas
842).OnRun(Trigger) line 18 - Base Application by Microsoft\r\n\"Job Queue Start Codeunit\"(CodeUnit 449).OnRun(Trigger) line 11 - Base Application by M
Dispatcher\"(CodeUnit 448).HandleRequest line 30 - Base Application by Microsoft\r\n\"Job Queue Dispatcher\"(CodeUnit 448).OnRun(Trigger) line 19 - Base
Microsoft","companyName":"CRONUS USA, Inc.","extensionId":"437dbf0e-84ff-417a-965d-ed2bb9650972","aadTenantId":"8ca62103-8877-486d-88e2-
9a91303abfc6","clientType":"Background","alObjectId":"840","component":"Dynamics 365 Business Central Server","executionTime":"00:00:05.7470000","sqlSta
\"SQLDATABASE\".dbo.\"CURRENTCOMPANY$Cash Flow Forecast Entry$437dbf0e-84ff-417a-965d-ed2bb9650972\" WHERE (\"Cash Flow Forecast No_\"=@0)"}
```


See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights
Monitoring and Analyzing Long Running SQL Queries On-Premises
The Business Central Administration Center

Analyzing Report Generation Telemetry

2/17/2021 • 10 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Report generation telemetry gathers data about reports that are run on the service. It provides information about whether the report dataset generation succeeded, failed, or was canceled. For each report, it tells you how long it ran, how many SQL statements it executed, and how many rows it consumed.

You use this data to gather statistics to help identify slow-running reports.

Success report generation

Occurs when a report dataset generates without any errors.

General dimensions

The following table explains the general dimensions of this trace.

DIMENSION	DESCRIPTION OR VALUE
operation_Name	**Success report generation** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
message	Version 16.1 and later: **Report rendered: {report ID} - {report name}** Before version 16.1: **The report {report ID} '{report name}' rendered successfully**
severityLevel	1

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.

DIMENSION	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0006** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
numberOfRows	Specifies the number of rows/records generated for the report dataset.
reportingEngine	Specifies the reporting engine used to generate the report, such as **ProcessingOnly**, **Rdlc**, or **Word**. This dimension was added in version 17.3
result	**Success**
serverExecutionTime	Specifies the amount of time it took the service to complete the request[1]. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the report executed[1].

DIMENSION	DESCRIPTION OR VALUE
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements[1].
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time it took for the system to generate the dataset and render the report[1]. The time has the format hh:mm:ss.sssssss.

[1]From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Failed report generation

This operation occurs when the report dataset couldn't be generated because of an error.

General dimensions

The following table explains the general dimensions of the **Failed report generation** operation.

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later: **Report rendering failed: {report ID} - {report name}** Before version 16.1: **The report {report ID} '{report name}' couldn't be rendered**
operation_Name	**Failed report generation** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
severityLevel	3

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.

DIMENSION	DESCRIPTION OR VALUE
cancelReason[2]	Specifies why the report was canceled.
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0006** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
numberOfRows	Specifies the number of rows/records generated for the report dataset.
result	Specifies the title of the exception that was thrown, such as **NavNCLDialogException**.
serverExecutionTime	Specifies the amount of time used by service on the request. The time has the format hh:mm:ss.ssssss.
sqlExecutes	Specifies the number of SQL statements that the report executed.
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.

DIMENSION	DESCRIPTION OR VALUE
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.
totalTime	Specifies the amount of time used to generate the dataset and render the report before it failed. The time has the format hh:mm:ss.sssssss.

[2] Available in Business Central 2020 release wave 2 and later only.

Analyzing report generation failures

When a report fails to generate, the `result` column in the CustomDimensions will include the title of the exception that was thrown by the service or the AL code.

Cancellation report generation

This operation occurs when the report dataset generation was canceled. There are various conditions that can cancel a report. The **Cancellation report generation** operation emits different trace messages for each condition.

General dimensions

The following table explains the general dimensions of the **Cancellation report generation** operation.

DIMENSION	DESCRIPTION OR VALUE
operation_Name	**Cancellation report generation** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.
message	Specifies the reason why the report was canceled. See Analyzing cancellation messages section for details.
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.

DIMENSION	DESCRIPTION OR VALUE
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0007** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Analyzing report cancellations

The cancellation messages indicate events that caused the report to be canceled. The telemetry can help identify slow-running reports - reports that take longer than expected to run and generate a large number of rows.

> **NOTE**
>
> The service evaluates cancellation events in a specific order, and the evaluation is done every five seconds. For more information, see Report Generation and Cancellation Flow.

Cancellation event received. Requesting cancellation of the action.

This message occurs when the session canceled a report as it was being generated.

Received a cancellation request from the user. Requesting cancellation of the action.

This message occurs when a user canceled a report in the client as it was being generated.

The action took longer to complete ({0}) than the specified threshold ({1}). Requesting cancellation of the action.

The service is configured to cancel reports if they take longer to generate than a set amount of time. With Business Central online, you can't change the threshold. With Business Central on-premises, you change the threshold by setting the **Max Execution Timeout** parameter on the Business Central Server instance. There's no timeout for on-premises by default. For more information, see Configuring Business Central Server.

The rendering of the word report has been cancelled because it took longer than the specified threshold ({0})"

This message occurs when a report that based on a Word layout takes longer to generate than the specified threshold. The event is only relevant for Business Central online. There's no timeout for on-premises.

The number of processed rows exceeded ({0} rows) the maximum number of rows ({1} rows). Requesting cancellation of the action.

The service is configured to cancel reports if they generate more than a set number of rows. With Business Central online, you can't change this threshold. With Business Central on-premises, you change the threshold by setting the **Max Rows** parameter on the Business Central Server instance. There's no limit on rows for on-premises by default. For more information, see Configuring Business Central Server.

Report cancelled but a commit occurred

This operation occurs when the report dataset generation was canceled but a COMMIT operation occurred before the cancellation. This pattern isn't recommended. Reconsider the report design.

General dimensions

The following table explains the general dimensions of the **Report cancelled but a commit occurred** operation.

DIMENSION	DESCRIPTION OR VALUE
message	The report <ID> '<Name>' is being canceled, but a COMMIT() has been performed. This can lead to data inconsistency if the report is not idempotent
severityLevel	2

Custom dimensions

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the report object that was run.
alObjectName	Specifies the name of the report object that was run.
alObjectType	**Report**.
alStackTrace	The stack trace in AL.
cancelReason	The reason why the report was cancelled
clientType	Specifies the type of client that executed the SQL Statement, such as **Background** or **Web**. For a list of the client types, see ClientType Option Type.

DIMENSION	DESCRIPTION OR VALUE
companyName	Specifies the display name of the Business Central company for which the report was run.
component	**Dynamics 365 Business Central Server**.
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	Specifies a comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0011** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
extensionId	Specifies the appID of the extension that the report object belongs to.
extensionName	Specifies the name of the extension that the report object belongs to.
extensionPublisher	Specifies the name of the extension publisher that the report object belongs to.
extensionVersion	Specifies the version of the extension that the report object belongs to.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Incoming Web Services Request Telemetry

2/17/2021 • 5 minutes to read • Edit Online

INTRODUCED IN: Business Central 2020 release wave 1

Web services telemetry gathers data about SOAP, OData, and API requests through the service. It provides information like the request's endpoint, time to complete, the SQL statements run, and more.

As a developer, you use the data to learn about conditions that you can change to improve performance. The following table provides some examples:

CONDITION	ANALYSIS
A web service request results in a long running SQL query	Adjust or fine-tune code.
Web service requests to a specific endpoint read more rows than requests to the other endpoints	Consider adding filtering to limit the rows that are read.
Fewer API type requests compared with other types	With SOAP and OData requests, computation resources are used on UI elements that aren't relevant. Instead of exposing normal pages as web service endpoints, use the built-in API pages. API pages are optimized for this scenario.
High number of requests to endpoints that include Power BI	This condition may indicate excessive Power BI integration.

For more performance guidelines, see Writing efficient Web Services.

> **NOTE**
> Business Central online and on-premises are configured with various limits on web service requests. For example, there is a request timeout and a maximum connections limit. For online, you can't change these limits, but it is helpful to know what the limits are. See Current API Limits. For on-premises, you change the limits on the Business Central Server instance. See Configuring Business Central Server. Web service calls that exceed the timeout limit result in a **408 - Request Timeout**. These calls are recorded in Application Insights with a totalTime that is equal to the timeout threshold.

General dimensions

The following table explains the general dimensions included in an incoming **Web Services Call** trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
operation_Name	**Web Services Call** **Note:** The use of the `operation_Name` column was deprecated in version 16.1. In future versions, data won't be stored in this column. So in version 16.1 and later, use the custom dimension column `eventID` column custom in Kusto queries instead of `operation_Name`.

DIMENSION	DESCRIPTION OR VALUE
message	Version 16.1 and later (depending on the type): • **Web service called (API): {endpoint}** • **Web service called (ODataV4): {endpoint}** • **Web service called (ODataV3): {endpoint}** • **Web service called (SOAP): {endpoint}** Before version 16.1: • **Received a web service request of type API** • **Received a web service request of type ODataV4** • **Received a web service request of type ODataV3** • **Received a web service request of type SOAP**
severityLevel	**1**

Custom dimensions

The following table explains the custom dimensions included in a **Web Services Call** trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.[1]
alObjectName	Specifies the name of the AL object that was run by the request.[1]
alObjectType	Specifies the type of the AL object that was run by the request.[1]
category	Specifies the service type. Values include: **API**, **ODataV4**, **ODataV3**, and **SOAP**.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0008** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
httpHeaders	Introduced in version 16.3. Specifies the http headers set in the request.

DIMENSION	DESCRIPTION OR VALUE
httpMethod	Introduced in version 16.3. Specifies the HTTP method used in the request. Values include: POST, GET, PUT, PATCH, orDELETE.
httpStatusCode	Introduced in version 16.3. Specifies the http status code returned when a request has completed. This dimension further indicates whether request succeeded or not, and why. Use it to verify whether there was an issue with a request even though the request was logged as successful. The dimension displays one of the following values: • **200** OK. The request succeeded. • **401** Access denied. The user who made the request doesn't have proper permissions. For more information, see Web Services Authentication and Assign Permissions to Users and Groups. • **404** Not found. The given endpoint was not valid. For more information, see Publishing a Web Service • **408** Request timed out. The request took longer to complete than the threshold configured for the service. For information about this threshold in Business Central online, see OData request limits. For on-premises, the timeout is determined by the ODataServicesOperationTimeout setting of the Business Central Server. For more information, see Configuring Business Central Server • **429** Too Many Requests. The request exceeded the maximum simultaneous requests allowed on the service. For information about this threshold in Business Central online, see OData request limits. For on-premises, the timeout is determined by the ODataMaxConnections setting of the Business Central Server. For more information, see Configuring Business Central Server This dimension was introduced in Business Central 2020 release wave 1, version 16.3. This dimension is not populated for SOAP endpoints.
serverExecutionTime	Specifies the amount of time it took the server to complete the request**. The time has the format hh:mm:ss.sssssss.
sqlExecutes	Specifies the number of SQL statements that the request executed.[1] [2]
sqlRowsRead	Specifies the number of table rows that were read by the SQL statements.[1] [2]
totalTime	Specifies the amount of time it took to process the request.[2] The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

[1]This dimension isn't relevant for $metadata calls, like `https://localhost:7048/BC/ODataV4/$metadata`, so it won't be in the trace.

[2]From telemetrySchemaVersion **0.6** and onwards, this value also includes the CompanyOpen operation.

Example trace

The following code snippet is a CustomDimensions example:

{"telemetrySchemaVersion":"0.6","componentVersion":"16.0.11329.0","environmentType":"Production","deprecatedKeys":"Company name, AL
AL Object type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension version, Telemetry schema v
AadTenantId, Environment name, Environment type, Component, Component version, Telemetry schema
version","serverExecutionTime":"00:00:00.3886441","component":"Dynamics 365 Business Central
Server","aadTenantId":"common","sqlExecutes":"21","sqlRowsRead":"117","totalTime":"00:00:00.3886441","alObjectType":"Page","alObject
Document Line
Entity","alObjectId":"6403","category":"ODataV4","endpoint":"BC/ODataV4/Company()/workflowSalesDocumentLines","httpStatusCode":"200"

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Outgoing Web Service Request Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2 and later

Outgoing web service request telemetry gathers data about outgoing web service requests sent using the AL HTTPClient module. As a partner, the data gives you insight into the execution time and failures that happen in external services that your environment and extensions depend on. Use the data to monitor environments for performance issues caused by external services, and be more proactive in preventing issues from occurring.

General dimensions

The following table explains the general dimensions included in an outgoing **Web Services Call (Outgoing)** trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Web Service Called (Outgoing): {endpoint}**
severityLevel	**1**

Custom dimensions

The following table explains the custom dimensions included in a **Web Services Call** trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alAuthenticationMethod	Specifies the user authentication used by the Business Central service . Values include: Windows, UserName, NavUserPassword, AccessControlService. For more information about the authentication types, see Authentication and Credential Types.
alHttpTimeout	Specifies the timeout defined for the request. The timeout is the time to wait before a request gets canceled. The value has the format hh:mm:ss. The timeout is defined either by the NavHttpClientMaxTimeout setting on the Business Central Server instance or by a TimeOut method call in extension code. The TimeOut method call takes precedence.
alObjectId	Specifies the ID of the AL object that made the request.
alObjectName	Specifies the name of the AL object that made the request.
alObjectType	Specifies the type of the AL object that made the request.
companyName	Specifies the display name of the Business Central company from which the request was made.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request. The endpoint is cleaned to include only the base URI.
environmentName	Specifies the name of the tenant environment. See Managing Environments.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0019**
extensionId	Specifies the appID of the extension that made the request.
extensionName	Specifies the name of the extension that made the request.
extensionVersion	Specifies the version of the extension that made the request.

DIMENSION	DESCRIPTION OR VALUE
httpHeaders	Introduced in version 17.2. Specifies the http headers set in the request.
httpMethod	Specifies the HTTP method used in the outgoing request. Values include: POST, GET, PUT, PATCH, orDELETE.
httpReturnCode	**Deprecated in version 17.2. Use the dimension httpStatusCode instead.** Specifies the http status code returned when a request has completed. This dimension further indicates whether request succeeded or not, and why. Use it to verify whether there was an issue with a request even though the request was logged as successful. The dimension displays one of the following values: • **200** OK. The request succeeded. • **404** Not found. The given endpoint wasn't valid.
httpStatusCode	Specifies the http status code returned when a request has completed. This dimension further indicates whether request succeeded or not, and why. Use it to verify whether there was an issue with a request even though the request was logged as successful. The dimension displays one of the following values: • **200** OK. The request succeeded. • **404** Not found. The given endpoint wasn't valid.
serverExecutionTime	Specifies the amount of time it took the server to complete the request, including the time to open the company. The time has the format hh:mm:ss.sssssss.
totalTime	Specifies the amount of time it took to process the request, including the time to open the company. The time has the format hh:mm:ss.sssssss.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Example trace

The following code snippet is a CustomDimensions example:


```
{"Telemetry schema version":"0.2","telemetrySchemaVersion":"0.2","Component version":"17.0.15765.0","Environment type":"Production","componentVersion":"17.0.1
AL Object Id, AL Object type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension version, Telemetry schema version, Comp
version":"17.0.15821.0","extensionVersion":"17.0.15821.0","Extension App Id":"11111111-aaaa-2222-bbbb-333333333333","aadTenantId":"afbb64dc-bc88-43a3-9153-dd9
type":"CodeUnit","Extension name":"Base Application","AL Object name":"My Codeunit","component":"Dynamics 365 Business Central Server","companyName":"CRONUS U
Codeunit","alObjectType":"CodeUnit","extensionId":"11111111-aaaa-2222-bbbb-
333333333333","eventId":"RT0019","httpMethod":"GET","serverExecutionTime":"00:00:00.1971767","totalTime":"00:00:00.1971767","endpoint":"https://mycorp.dynamic
```


See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Analyzing Web Service Access Key Telemetry

2/17/2021 • 3 minutes to read • Edit Online

APPLIES TO: Business Central 2020 release wave 2, version 17.3, and later

The Business Central emits telemetry data about the success or failure of authenticating web service access keys on web service requests.

In a future release, web service access key feature will be deprecated. As a partner or customer, this data lets you monitor the use of web service access keys on your environments in preparation for this change.

For information about web service access keys, see How to use an Access Key for SOAP and OData Web Service Authentication.

Authentication with web service key succeeded

Occurs when a web service access key was authenticated successfully.

General dimensions

The following table explains the general dimensions included in the trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Authentication with Web Service Key succeeded: {endpoint}**
severityLevel	**1**

Custom dimensions

The following table explains the custom dimensions included in the trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.
authenticationStatus	**Succeeded**
authenticationType	Specifies whether the service uses NavUserPassword or AccessControl (Azure AD) authentication.
category	Specifies the service type. Values include: **Api**, **ODataV4**, **ODataV3**, and **SOAP**.
component	**Dynamics 365 Business Central Server**
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)

DIMENSION	DESCRIPTION OR VALUE
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request. Private and sensitive information is omitted from the endpoint.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	**RT0020** This dimension was introduced in Business Central 2020 release wave 1, version 16.1.
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Authentication with web service key failed

Occurs when a web service access key failed to authenticate.

General dimensions

The following table explains the general dimensions included in the trace. The table lists the dimensions that are specific to Business Central.

DIMENSION	DESCRIPTION OR VALUE
message	**Authentication with Web Service Key failed: {endpoint}**
severityLevel	**3**

Custom dimensions

The following table explains the custom dimensions included in the trace.

DIMENSION	DESCRIPTION OR VALUE
aadTenantId	Specifies that Azure Active Directory (Azure AD) tenant ID used for Azure AD authentication. For on-premises, if you aren't using Azure AD authentication, this value is **common**.
alObjectId	Specifies the ID of the AL object that was run by request.
authenticationStatus	**Failed**
authenticationType	Specifies whether the service uses NavUserPassword or AccessControl (Azure AD) authentication.
category	Specifies the service type. Values include: **Api**, **ODataV4**, **ODataV3**, and **SOAP**.
component	**Dynamics 365 Business Central Server**

DIMENSION	DESCRIPTION OR VALUE
componentVersion	Specifies the version number of the component that emits telemetry (see the component dimension.)
deprecatedKeys	A comma-separated list of all the keys that have been deprecated. The keys in this list are still supported but will eventually be removed in the next major release. We recommend that update any queries that use these keys to use the new key name.
endpoint	Specifies the endpoint for the request. Private and sensitive information is omitted from the endpoint.
environmentType	Specifies the environment type for the tenant, such as **Production**, **Sandbox**, **Trial**. See Environment Types
eventId	RT0021
failureReason	Specifies the exception that was thrown by the server. Some exception messages can contain customer data. As a precaution, Business Central only emits information that's classified as SystemMetadata. Exception messages that contain other data classifications, like customer data, are not shown. Instead, the following message is shown: "Message not shown because the NavBaseException(string, Exception, bool) constructor was used."
telemetrySchemaVersion	Specifies the version of the Business Central telemetry schema.

Example trace

The following code snippet is a CustomDimensions example:

```
{"component":"Dynamics 365 Business Central
Server","category":"Api","AadTenantId":"common","environmentType":"Production","Component":"Dynamics 365 Business Central
Server","AL Object Id":"0","alObjectId":"0","Telemetry schema
version":"0.1","telemetrySchemaVersion":"0.1","authenticationType":"NavUserPassword","componentVersion":"17.0.15855.0","Compo
version":"17.0.15855.0","eventId":"RT0020","endpoint":"BC170/ODataV4/Company()/purchaseDocumentLines","deprecatedKeys":"Compa
name, AL Object Id, AL Object type, AL Object name, AL Stack trace, Client type, Extension name, Extension App Id, Extension
version, Telemetry schema version, Component, Component version, Telemetry schema version, AadTenantId, Environment name,
Environment type","aadTenantId":"common","Environment type":"Production","authenticationStatus":"Succeeded"}
```

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Telemetry Event IDs in Application Insights

2/17/2021 • 3 minutes to read • Edit Online

The following tables list the IDs of Business Central telemetry trace events that can be emitted in Azure Application Insights.

Application events

EVENT ID	AREA	MESSAGE
AL0000CTV	Email	Email sent successfully
AL0000CTE	Field monitoring	Sensitive field value has changed: {alfieldCaption} in table {altableCaption}
AL0000CTP	Email	Failed to send email
AL0000DD3	Field monitoring	Sensitive field monitor status has changed to {almonitorStatus}
AL0000E2A	Permissions	User-defined permission set added: {alPermissionSetId}
AL0000E2B	Permissions	User-defined permission set removed: {alPermissionSetId}
AL0000E28	Permissions	Permission set link added: {alSourcePermissionSetId} -> {alLinkedPermissionSetId}
AL0000E29	Permissions	Permission set link removed: {alSourcePermissionSetId} -> {alLinkedPermissionSetId}
AL0000E2C	Permissions	Permission set assigned to user: {alPermissionSetId}
AL0000E2D	Permissions	Permission set removed from user: {alPermissionSetId}
AL0000E2E	Permissions	Permission set assigned to user group: {alPermissionSetId}
AL0000E2F	Permissions	Permission set removed from user group: {alPermissionSetId}

Client events

EVENT ID	AREA	MESSAGE
CL0001	Page views	Page opened: {alObjectName}

Lifecycle events

EVENT ID	AREA	MESSAGE
AL0000E24	Job Queue Lifecycle	Job queue entry enqueued: {alJobQueueId}
AL0000E25	Job Queue Lifecycle	Job queue entry started: {alJobQueueId}
AL0000E26	Job Queue Lifecycle	Job queue entry finished: {alJobQueueId}
AL0000E3F	Configuration Package	Configuration package export started: {alPackageCode}
AL0000E3G	Configuration Package	Configuration package exported successfully: {alPackageCode}
AL0000E3H	Configuration Package	Configuration package import started: {alPackageCode}
AL0000E3I	Configuration Package	Configuration package imported successfully: {alPackageCode}
AL0000E3N	Configuration Package	Configuration package apply started: {alPackageCode}
AL0000E3O	Configuration Package	Configuration package applied successfully: {alPackageCode}
AL0000E3P	Configuration Package	Configuration package deleted successfully: {alPackageCode}
LC0001	Company Lifecycle	Company created: {companyName}
LC0002	Company Lifecycle	Company creation canceled: {companyName}
LC0003	Company Lifecycle	Company creation failed: {companyName}
LC0004	Company Lifecycle	Company copied: {companyNameSource} to {companyNameDestination}
LC0005	Company Lifecycle	Company copied canceled: {companyNameSource} to {companyNameDestination}

EVENT ID	AREA	MESSAGE
LC0006	Company Lifecycle	Company copy failed: {companyNameSource} to {companyNameDestination}
LC0007	Company Lifecycle	Company deleted: {companyName}
LC0008	Company Lifecycle	Company deletion canceled: {companyName}
LC0009	Company Lifecycle	Company deletion failed: {companyName}
LC0010	Extension Lifecycle	Extension installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0011	Extension Lifecycle	Extension failed to install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0012	Extension Lifecycle	Extension synchronized successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId}
LC0013	Extension Lifecycle	Extension failed to synchronize: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0014	Extension Lifecycle	Extension published successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0015	Extension Lifecycle	Extension failed to publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0016	Extension Lifecycle	Extension un-installed successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0017	Extension Lifecycle	Extension failed to un-install: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0018	Extension Lifecycle	Extension unpublished successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})

EVENT ID	AREA	MESSAGE
LC0019	Extension Lifecycle	Extension failed to un-publish: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0020	Extension Lifecycle	Extension compiled successfully: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0021	Extension Lifecycle	Extension failed to compile: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})
LC0022	Extension Lifecycle	Extension updated successfully: {extensionName} version {extensionVersion} by {extensionPublisher}
LC0023	Extension Lifecycle	Extension failed to update: {extensionName} version {extensionVersion} by {extensionPublisher} ({extensionId})

Runtime events

EVENT ID	AREA	MESSAGE
RT0001	Authorization	AuthorizationFailed(PreOpenCompany) : {failure reason}
RT0002	Authorization	Authorization Failed (Open Company): {failure reason}
RT0003	Authorization	AuthorizationSucceeded(PreOpenCompany)
RT0004	Authorization	Authorization Succeeded (Open Company)
RT0005	Performance	Operation exceeded time threshold (SQL query)
RT0006	Report generation	Success report generation
RT0006	Report generation	Report rendering failed: {report ID} - {report name}
RT0007	Report generation	Cancellation report generation
RT0008	Incoming Web service requests	Web service called ({category of request}): {endpoint}

EVENT ID	AREA	MESSAGE
RT0010	Extension lifecycle	Extension Update Failed: exception raised in extension {extensionName} by {extensionPublisher} (updating to version {extensionTargetedVersion})
RT0011	Report generation	Report cancelled but a commit occurred
RT0012	Performance	Database lock timed out
RT0013	Performance	Database lock snapshot: {snapshotId}
RT0014	Security	App Key Vault initialization succeeded: '{keyVaultUri}'
RT0015	Security	App Key Vault initialization failed
RT0016	Security	App Key Vault secret retrieval succeeded from key vault '{keyVaultUri}'
RT0017	Security	App Key Vault secret retrieval failed from key vault: '{keyVaultUri}'
RT0018	Performance	Operation exceeded time threshold (AL method)
RT0019	Outgoing Web service requests	Web Service Called (Outgoing): {endpoint}
RT0020	Web service key request	Authentication with Web Service Key succeeded: {endpoint}
RT0021	Web service key request	Authentication with Web Service Key failed: {endpoint}

See also

Monitoring and Analyzing Telemetry
Enabling Application Insights for Tenant Telemetry On-Premises
Enable Sending Telemetry to Application Insights

Session Timeout Settings and Configuration

2/17/2021 • 4 minutes to read • Edit Online

When you start a client, such as connecting to Business Central in a browser, a connection is established with the Business Central Server instance and a corresponding session is added on Business Central Server.

Business Central Server includes several timeout settings that determine when a session closes as a result of inactivity over the client connection, lost client connection, or closing of the client. To help you configure the timeout settings, this document provides an overview of how the session timeouts work and answers some basic questions about session behavior.

Session timeout settings overview

This section provides an overview of the settings that are available in Business Central to control when a Business Central Server session for a Business Central client connection times out and closes. Some of the settings are set on Business Central Server and others are set for the Business Central Web client. For more details about using these settings, see the other sections in this topic.

Business Central Server timeout settings

The following table describes the session timeout settings that are used by Business Central Server.

SETTING	DESCRIPTION	REMARKS
ClientServicesReconnectPeriod	The amount of time during which a client can reconnect to an existing session on Business Central Server before a session closes.	For more information, see Configuring How Long a Session Remains Open after the Client Connection is Lost.
ClientServicesIdleClientTimeout	The interval of time that a Business Central client connection can remain inactive before the session is closed.	For more information, see Configuring How Long a Session Remains Open When the Client Connection is Inactive.

These settings are available in the CustomSettings.config file of Business Central Server. For more information about this file, see Configuring Business Central Server.

Business Central Web client timeout settings

The following table describes the session timeout settings that are used by the Business Central Web client.

SETTING	DESCRIPTION	REMARKS
SessionTimeout	Specifies the amount of time that session remains open when there is no activity over the connection from the Business Central Web client to Business Central Server.	For more information, see Configuring Business Central Server.

This setting is available in the navsettings.json configuration file of the Business Central Web Server. For more information about this file, see Configuring Web Server.

Configuring How Long a Session Remains Open When the Client

Connection is Inactive

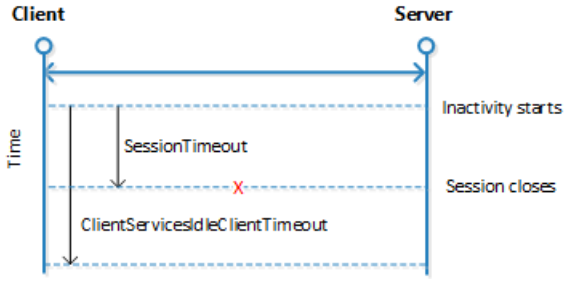
Inactivity on a connection is when the Business Central client is not sending messages to Business Central Server.

Configuring the inactive session timeout for the Business Central Web client

There are two settings that control when a Web client session closes because of inactivity on a connection:

- *ClientServicesIdleClientTimeout* setting on Business Central Server.
- *SessionTimeout* setting on the Business Central Web Server.

The session closes according to the setting that has the shortest time period. By default, the *ClientServicesIdleClientTimeout* setting is set to **MaxValue**, which means no time limit, and the *SessionTimout* setting is 00:20:00 (20 minutes). This means that when client connection is inactive, a session will close after 20 minutes. The following figure illustrates the timeout behavior:

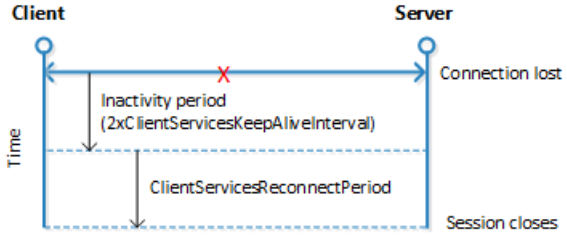


Configuring How Long a Session Remains Open after the Client Connection is Lost

Occasionally, a Business Central client can lose the network connection to Business Central Server. You can use *ClientServicesReconnectPeriod* setting on Business Central Server to control how long a session remains open after the connection is lost to allow time for the client to reconnect to the session.

The time a session remains open actually depends two settings: *ClientServicesKeepAliveInterval* and *ClientServicesReconnectPeriod*. The *ClientServicesKeepAliveInterval* setting is used to specify an initial inactivity period. The initial inactivity period is equal to two times the *ClientServicesKeepAliveInterval* setting value. After this initial inactivity period, the session remains open for the time period that is specified *ClientServicesReconnectPeriod* setting. By default, the *ClientServicesKeepAliveInterval* setting is 120 seconds (2 minutes) and the *ClientServicesReconnectPeriod* setting is 10 minutes. This means that Business Central Server waits approximately 14 minutes for the client to reconnect before closing the session.

The following figure illustrates the reconnect session timeout behavior.



The process that occurs when a client does not reconnect to the session is explained as follows:

1. The connection is lost and the initial inactivity period starts (default is 4 minutes).
2. After the initial inactivity period, the service channel enters a faulted state.

   When the service channel is in the faulted state, Business Central Server considers the session with the

client as orphaned and waits for it to reconnect.

3. If the client does not reconnect within the time period that is specified by the *ClientServicesReconnectPeriod* setting (default is 10 minutes), then Business Central Server closes the session.

4. The session is then removed from the **Active Session** table in the Business Central.

FAQ

This section answers some typical questions about session timeout.

What happens to the session if the client loses the connection to Business Central Server?

By default, it will take approximately 14 minutes for the Business Central Server to close the current session. The time it takes to close the session is in part determined by the *ClientServicesReconnectPeriod* setting on Business Central Server plus an initial 10 minute inactivity period. For more information, see Configuring How Long a Session Remains Open after the Client Connection is Lost.

What happens if the session is still active when Business Central Server tries to close it?

1. The server stops any executing threads when the next statement is to be executed and the current call stack is aborted so any uncommitted transactions will be rolled back.

2. The server cancels any callbacks to the client (similar to waiting for the response to a Confirm dialog).

3. The session is closed, and then removed from the Active Session table.

Preparing Dynamics 365 Sales for Integration

2/17/2021 • 2 minutes to read • Edit Online

This article describes how to set up and configure Dynamics 365 Sales for integrating with Business Central. Complete the following tasks:

1. Create a user for connecting to and synchronizing data from Business Central.
2. Install the Business Central integration solution for Dynamics 365 Sales.

   This task is optional. You only need to complete this task if you want the functionality that is provided by the Business Central integration solution.

> **IMPORTANT**
> To perform the tasks in this topic, you must have the System Administrator security role or equivalent privileges in Dynamics 365 Sales.

Create a Dynamics 365 for Sales User for Connecting to Business Central

As a minimum, this user must be a non-interactive user account that has the required privileges to write, read, modify, and delete data in the entities that will be integrated with Business Central.

You will use this user account to set up the connection to Dynamics 365 Sales from Business Central.

> **IMPORTANT**
> You should not use this account to sign in to Dynamics 365 Sales to modify entities records that are integrated with Business Central because the changes will be ignored by integration synchronization jobs in Business Central.

Create the connection user

- For more information about how to create users in Dynamics 365 Sales, see https://go.microsoft.com/fwlink/?LinkID=616518.

Install the Business Central Integration Solution

Business Central includes a solution that enables users to access coupled records in Business Central, such as customers and items, from records in Dynamics 365 Sales, such as accounts and products. The solution adds a link on the Dynamics 365 Sales record pages that opens the coupled Business Central record. The solution is also used to display information from Business Central in a part on certain entity records in Dynamics 365 Sales, such as accounts. Installing this solution is optional.

1. From Business Central installation media (DVD), copy either the DynamicsNAVIntegrationSolution_v8.zip or DynamicsNAVIntegrationSolution_v9.zip file to your computer.

   These files are located in the **CrmCustomization** folder. This file is the solution package.

   Use the zip version that matches the Dynamics 365 Sales SDK version. Use DynamicsNAVIntegrationSolution_v8.zip for legacy services running CRM or Dynamics 365 Sales version 8.x and earlier. Use DynamicsNAVIntegrationSolution_v9.zip for Dynamics 365 Sales versions 9.0 and

later.

2. In Dynamics 365 Sales, import the DynamicsNAVIntegrationSolution.zip as a solution.

   This step adds the **Business Central Connection** entity and **Business Central Account Statistics** entity in the system and additional items such as Business Central integration security roles.

   For more information about how to manage solutions in Dynamics 365 Sales, https://go.microsoft.com/fwlink/?LinkID=616519.

3. (Optional) Set up the **Business Central Connection** entity to display in the **Settings** area of Dynamics 365 Sales.

   This setup enables Dynamics 365 Sales users who are assigned the **Business Central Admin** role to modify the entity in Dynamics 365 Sales. For more information about how to modify entities in Dynamics 365 Sales, see https://go.microsoft.com/fwlink/?LinkID=616521.

4. Assign the **Business Central Integration Administrator** role to the Business Central connection user.

5. Assign the **Business Central Integration User** role to all users who require the use of the features provided by the Business Central integration solution.

If you install the Business Central integration solution after you have set up the connection to Dynamics 365 Sales from in Business Central, you must modify the connection setup to point to the URL of the Business Central Web client.

See Also

Connecting On-Premises Versions

Registering Business Central On-Premises in Azure AD for Integrating with Other Services

2/17/2021 • 5 minutes to read • Edit Online

APPLIES TO Business Central on-premises. Business Central online is automatically configured for integration with other online services.

This article describes how to set up Business Central on-premises to use services that are based on Microsoft Azure. There are several services that you can integrate with Business Central on-premises, like Cortana Intelligence and Power BI. Before using the services, you have to register Business Central on-premises in Azure Active directory and give it access to the services. For example, the Sales and Inventory Forecast extension requires that you specify an API key and API URI. Other services require similar information.

NOTE

In Business Central version earlier than 16.4, the **Set up Azure Active Directory** wizard has an **Auto register** action. Previously, you could use this action to automatically register Business Central in Azure AD. The auto-register functionality has since been removed. Now, you must register the application manually, regardless of your version. The wizard in earlier versions still includes the **Auto register** link. But the link now opens this article, which guides you through the manual registration.

Prerequisites

- An Azure Active Directory (AD) tenant.

  You'll need a tenant on Azure AD that has at least one user. For more information, see Quickstart: Set up a tenant.

  If the Business Central deployment is using Azure AD authentication, then you already have a tenant with users. See Authenticating Business Central Users with Azure Active Directory.

  If your deployment uses NavUserPassword authentication, you'll need the credentials (sign in email and password) of a user account later in this article.

- An Azure portal account

  You'll need an account for accessing the Azure portal. In most cases, this account is the same as your Business Central account. You'll use this account to access Azure Active AD tenant via the Azure portal.

Register an application in Azure Active Directory

The first task is to use Azure portal to register an application for Business Central on your Azure AD tenant. As part of the registration, you'll also give the relevant services access to the application. The purpose of registration is to ensure Business Central on-premises and the services know each other's Azure Active Directory (Azure AD) details.

TIP

The following steps describe how to register a new application. However, if you're using AZure AD authentication, you already have a registered application for Business Central. So instead of registering a new application, you can use the existing application. But if you do, make sure you modify it based on the information in the steps that follow.

1. Sign in to the Azure portal and register an application for Business Central on-premises in Azure Active Directory tenant.

   a. Follow the general guidelines at Register your application with your Azure Active Directory tenant.

      When you add an application to an Azure AD tenant, you must specify the following information:

SETTING	DESCRIPTION
Name	Specify a name for your Business Central on-premises solution, such as *Business Central on-premises* or *Azure Services for Business Central on-premises*.
Supported account types	Select **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**
Redirect URI	Set the first box to **Web** to specify a web application. Enter the URL for your Business Central on-premises browser client, followed by *OAuthLanding.htm*. For example, *https://MyServer:443/BC160/OAuthLanding.htm*. This file is used to manage the exchange of data between Business Central on-premises and other services through Azure AD.

      When completed, an **Overview** displays in the portal for the new application.

   b. Copy the **Application (Client) ID** that was assigned the application and also redirect URL that you specified. You'll use this information later.

2. Create a client secret for the registered application.

   a. Follow the general guidelines at Add credentials to your web application.

   b. Before you leave the **Certificates & secrets** page, copy the secret's value to a temporary location. The value isn't accessible once you leave the page. You'll use this key later in your client application code.

3. Grant the registered application delegated permission to access the required service APIs, like Power BI.

   Follow the general guidelines at Add permissions to access web APIs for each service.

   Use the following table to help you set the minimum permissions:

API / PERMISSION NAME	TYPE	DESCRIPTION
Microsoft Graph / User.Read	Delegated	Sign in and read user profile
Power BI Service / Report.Read.All	Delegated	View all reports

Set up the registered application in Business Central

After you register the application, the next task is to configure the Business Central tenant to use the application. You'll need the information about the application that you created in the previous task: redirect URL, application (client) ID, and client secret.

1. In the top-right corner, choose the  icon, enter **Assisted Setup**, and then choose the related link.
2. Select **Set up Azure Active Directory**, then **Next**.

   The **Connect With Azure** page opens.
3. In the **Redirect URL** field, make sure the URL matches the redirect URL that's assigned the registered Business Central application in Azure AD.
4. In the **Application ID** field, specify the application (client) ID of the Business Central application in Azure AD that you copied in the previous task.
5. In the **Key** field, specify the value of the client secret used by the Business Central application in Azure AD.
6. Choose **Next**.

   If you're using NavUserPassword authentication, you're prompted to sign in to the Azure AD tenant. In this case, enter the sign in email and password of a valid account.

Unless you see an error message, you're now done. The Business Central on-premises solution is registered and ready to connect to services such as Cortana Intelligence, or embedding Power BI in Business Central.

Fixing problems

This section provides solutions to problems that might occur.

Sorry, but we're having trouble signing you in

When you try to connect, you get a message similar to the following:

AADSTS50011: The reply URL specified in the request does not match the reply URLs configured for the application: '1111111-aaaa-2222-bbbb-333333333333'

To fix this issue, verify that the **Reply URL** in the Setup Azure AD page is correct. It must match the Reply URL set on the registered app in Azure AD.

See Also

Business Central and Power BI
FAQ about Connecting to the Intelligent Cloud from On-Premises Solutions
Deployment of Dynamics 365 Business Central
Migrating On-Premises Data to Business Central Online

Upgrading to Dynamics 365 Business Central

2/17/2021 • 2 minutes to read • Edit Online

This section provides an overview of how to upgrade to Business Central. The upgrade process depends on different factors, including on your decision to deploy Business Central on-premises or move your solution online.

> **TIP**
>
> Before you decide how to upgrade your solution, make sure that you read the upgrade considerations at Important Information and Considerations for Before Upgrading to Dynamics 365 Business Central.

Depending on your decision to upgrade to either Business Central online or on-premises, different scenarios are supported. Read the upgrade considerations, and then revise the supported scenarios in the respective sections. If you cannot find guidance for your migration scenario, you can check at the Ideas site if someone else has already suggested support for the same migration path. For more information, see https://aka.ms/businesscentralideas.

Depending on which version you are upgrading from, you may have to convert or migrate the existing Help for your solution. For more information, see Migrate Legacy Help to the Business Central Format and User Assistance Model.

See Also

Upgrading to Dynamics 365 Business Central Online
Upgrading to Dynamics 365 Business Central On-Premises
Before You Upgrade
Migrate Legacy Help to the Business Central Format
Product and Architecture Overview
Migrating to Multitenancy
Deployment

Supported Upgrade Paths to Dynamics 365 Business Central Releases

2/17/2021 • 2 minutes to read • Edit Online

Business Central is available in several release versions. You can upgrade an existing Dynamics NAV or Business Central solution to any of these releases. Depending on your solution's current version, it might not be possible to upgrade directly to a particular release. You might have to upgrade indirectly through an intermediate release, before upgrading to the target release.

The following sections provide the supported upgrade paths to the different Business Central releases.

> **NOTE**
> Minor updates are regularly made available for the major releases. Make sure you upgrade to an update of the release version that is compatible with your source source version. Otherwise, you might encounter problems upgrading the application. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

Upgrade to Business Central October 2018 (v13)

SOURCE VERSION	UPGRADE PATH
• Microsoft Dynamics NAV 2015 • Microsoft Dynamics NAV 2016 • Microsoft Dynamics NAV 2017 • Microsoft Dynamics NAV 2018	Direct

Upgrade to Business Central Spring 2019 (v14)

SOURCE VERSION	UPGRADE PATH
• Microsoft Dynamics NAV 2015 • Microsoft Dynamics NAV 2016 • Microsoft Dynamics NAV 2017 • Microsoft Dynamics NAV 2018 • Business Central v13	Direct

Upgrade to Business Central 2019 Release Wave 2 (v15)

SOURCE VERSION	UPGRADE PATH
• Microsoft Dynamics NAV 2015 • Microsoft Dynamics NAV 2016 • Microsoft Dynamics NAV 2017 • Microsoft Dynamics NAV 2018 • Business Central v13	Indirect. Upgrade to Business Central v14 first.

SOURCE VERSION	UPGRADE PATH
• Business Central v14	Direct

Upgrade to Business Central 2020 Release Wave 1 (v16)

SOURCE VERSION	UPGRADE PATH
• Microsoft Dynamics NAV 2015 • Microsoft Dynamics NAV 2016 • Microsoft Dynamics NAV 2017 • Microsoft Dynamics NAV 2018 • Business Central v13	Indirect. Upgrade to Business Central v14 first.
• Business Central v14) • Business Central (v15)	Direct

Upgrade to Business Central 2020 Release Wave 2 (v17)

SOURCE VERSION	UPGRADE PATH
• Microsoft Dynamics NAV 2015 • Microsoft Dynamics NAV 2016 • Microsoft Dynamics NAV 2017 • Microsoft Dynamics NAV 2018 • Business Central v13	Indirect. Upgrade to Business Central v14 first.
• Business Central v14) • Business Central (v15) • Business Central (v16)	Direct

See Also

Upgrading to Business Central Spring 2019
Upgrading to Business Central 2019 release Wave 2
Upgrading to Business Central 2020 release Wave 1
Migrating On-Premises Data to Business Central Online

Upgrading to Dynamics 365 Business Central On-Premises Spring 2019 (v.14)

2/17/2021 • 2 minutes to read • Edit Online

The upgrade process depends on different factors, such as the version of Dynamics NAV that you are upgrading from, and the degree to which your solution differs from the standard version of Dynamics NAV. The mains tasks range from converting the database to upgrading application code and data.

Use the following table to determine the procedures that you must complete for your upgrade scenario:

SCENARIO	TASKS
Full upgrade from one of the following versions: • Microsoft Dynamics NAV 2015 • Microsoft Dynamics NAV 2016 • Microsoft Dynamics NAV 2017 • Microsoft Dynamics NAV 2018 • Business Central October 2018	From these versions, you can upgrade directly to the latest version of Business Central by following these tasks: 1. Upgrade the Application Code 2. Upgrade the Data
Full upgrade from one of the following versions: • Microsoft Dynamics NAV 2013 • Microsoft Dynamics NAV 2013 R2	1. Upgrade to Microsoft Dynamics NAV 2018. For more information, see Upgrading to Microsoft Dynamics NAV 2018. 2. Upgrade to Business Central. a. Upgrade the Application Code b. Upgrade the Data: Single-Tenant Deployment or Upgrade the Data: Multitenant Deployment

SCENARIO	TASKS
Full upgrade from one of the following versions: • Microsoft Dynamics NAV 2009 SP1 • Microsoft Dynamics NAV 2009 R2 • Microsoft Dynamics NAV 5.0 • Microsoft Dynamics NAV 4.0	There are two different upgrade paths to Business Central, depending on the version you are upgrading from. For Microsoft Dynamics NAV 5.0 and Microsoft Dynamics NAV 4.0, you must go through Microsoft Dynamics NAV 2013. For Microsoft Dynamics NAV 2009 SP1 and Microsoft Dynamics NAV 2009 R2, you can choose to go through Microsoft Dynamics NAV 2013 or Microsoft Dynamics NAV 2015 **Through Microsoft Dynamics NAV 2013** 1. Upgrade to Microsoft Dynamics NAV 2013. For more information, see Upgrading to Microsoft Dynamics NAV 2013 in Previous Versions Documentation. 2. Upgrade to Microsoft Dynamics NAV 2018. For more information, see Upgrading to Microsoft Dynamics NAV 2018. 3. Upgrade to Business Central. a. Upgrade the Application Code b. Upgrade the Data: Single-Tenant Deployment or Upgrade the Data: Multitenant Deployment **Through Microsoft Dynamics NAV 2015** 1. Upgrade to Microsoft Dynamics NAV 2015. For more information, see Dynamics NAV Team Blog. 2. Upgrade to Business Central. a. Upgrade the Application Code b. Upgrade the Data: Single-Tenant Deployment or Upgrade the Data: Multitenant Deployment After the upgrade, links between interaction records and logged email messages is lost. To resolve this issue, the administrator has to log all mails again to restore the links. For more information, see Logging Interaction Links are Lost When You Upgrade from Microsoft Dynamics NAV 2009 R2.
Platform-only upgrade of Dynamics NAV or Business Central to a new platform version, such as with a cummulative update	• Technical Upgrade You can also use this procedure to convert a database to Business Central technical requirements, and then upgrade the application and data later.

Before you begin the upgrade process, see Important Information and Considerations for Before Upgrading for tips about things to consider when you prepare to upgrade to Business Central.

See Also

Upgrading to Dynamics 365 Business Central
Upgrading to Dynamics 365 Business Central Online
Product and Architecture Overview
Migrating to Multitenancy
Migrating On-Premises Data to Business Central Online
Deployment

Important Information and Considerations for Before Upgrading to Dynamics 365 Business Central Spring 2019 and Later Versions

2/17/2021 • 5 minutes to read • Edit Online

Depending on which version you are upgrading from, and the degree to which your solution differs from the standard version of Business Central, you may want to prepare your solution for the upgrade. This topic provides important information and tips for things to consider when you prepare to upgrade to Business Central.

Migrate from Dynamics NAV to Business Central online

You can upgrade to Business Central online from supported versions of Dynamics NAV on-premises, provided that your application customization is handled by extensions. Any data from tables with code customizations cannot be carried forward from Dynamics NAV.

> **NOTE**
> Upgrade your solution to Business Central Spring 2019 (version 14) or later, and then migrate to Business Central online.

The process consists of two parts:

- Upgrade from Dynamics NAV to Business Central using the tools described in Upgrading to Business Central on-premises (version 14). For more information, see Supported Upgrade Paths to Dynamics 365 Business Central Releases.
- Convert non-standard functionality and customizations to apps and per-tenant extensions. For more information, see Deploying a Tenant Customization.
- Run the cloud migration tool as described in Running the Cloud Migration Tool, and then switch to use Business Central online going forward.

Upgrading from Dynamics NAV

This section lists specific changes between Dynamics NAV and Business Central.

Codeunit 1 has been deprecated and replaced

Dynamics NAV included codeunit **1 ApplicationManagement**. In Business Central, this codeunit has been retired, and new 'system' codeunits have been introduced in the 2 billion range.

For information, see Transitioning from Codeunit 1 to System Codeunits.

V1 Extensions have been discontinued

With Business Central, extensions V1 are no longer supported for on-premise installations. As a result, any custom extensions V1 must be converted to extensions V2 in the old environment before upgrading to Business Central.

For information about how to convert to extensions V2, see Converting Extensions V1 to Extensions V2.

MenuSuite not used for page and report search

With Business Central, the MenuSuite is no longer used to control whether a page or report can be found in the

search feature of the Web client. This is now determined by specific properties on the page and report objects. As part of the application code upgrade process, you change these properties on existing pages and reports used by the MenuSuite to ensure that they are still searchable from the Web client. For more information, see Making Pages and Reports Searchable After an Upgrade.

Dynamics 365 Sales integration

Because of changes in Dynamics 365 Sales and the integration since pervious releases, if your application is integrating with Dynamics 365 Sales, then you must perform a full upgrade instead of just a technical upgrade.

New and changed application features

There a several new and changed application features available in Business Central April 2019 for users, administrators, and developers. For an overview of these features, see Overview of Dynamics 365 Business Central April '19 release.

To take advantage of these all these features, you will have to perform an application code upgrade, not just a technical (platform) upgrade.

Changes to profiles in the CRONUS International Ltd. demonstration database and promoted actions

With the Business Central April 2019 release, profiles that are part of the CRONUS International Ltd. demonstration database, such as the **Sales Order Processor** profile, customize fewer pages compared to earlier releases. For customers that rely on these profiles, their users might experience slight differences in the layout of actions in the action bar on pages. Additionally, the layout of promoted actions on over 380 core application pages has been fine-tuned.

To ensure that users are not disrupted by these changes, we recommend that administrators and partners who are upgrading a customer to Business Central April 2019, review the layout of promoted actions when combined with their own code and profile customization.

Names of variables

Business Central introduces new methods and statements. If your solution includes variables where the name is now used by a standard AL method or statement such as REGISTERTABLECONNECTION or FOREACH, you must change the variables before you upgrade to Business Central. Alternatively, you can enclose the variable names in quotation marks. If you do not, and you import an object that has this code in text format, you cannot compile the object.

Deprecated or redesigned functionality

If you are upgrading a solution that depends on functionality that is deprecated or changed in the default version of Business Central, you must verify that the upgrade codeunits migrate data correctly. See the See Also section for links to descriptions of deprecated functionality.

Deprecated fields and fields marked as obsolete

Sometimes Microsoft will refactor code so that fields are no longer used, or the functionality is moved from the base application to an extension, for example. Typically, the upgrade toolkit will manage the upgrade impact, but for transparency, you can find a list of fields that are deprecated in the current release or marked to be obsolete in a later release. For more information, see Deprecated Fields, and Fields Marked as Obsolete.

Upgrade codeunits

When you introduce changes to the database schema, Business Central will check if these changes are

destructive or not. If the database check indicates that the change may lead to data deletion, such as if you are dropping a table column so that the contents of that column will be deleted, this is considered a destructive change. You will be prompted to handle the situation using upgrade codeunits.

Company names

If a company name includes a special character, an error may display during the upgrade. In this context, special characters include the following:
[~ @ # $ % & * ( ) . ! % - + / = ?]

If you are going to upgrade a database where one or more company name includes a special character, we recommend that you rename the company before you start the upgrade process. After the upgrade is successfully finished, you can rename the company again.

System tables with non-English names

In older versions of Dynamics NAV, you could translate the columns in system tables to a language other than English. Starting with version 3.0, we advised heavily against this, and versions later than Microsoft Dynamics NAV 2013 R2 require that all columns in all system tables are in English. As a result, if you try to open a database with non-English system tables in Microsoft Dynamics NAV 2013 R2 or later, an error displays, saying that one or more columns do not exist.

Make sure that all objects where compiled in a development environment with the right .ETX and .STX files. You can verify that you are running in the correct environment with English (US) as the base language by opening the **ndo$dbproperty** table in SQL Server Management Studio. In the **Identifiers** column, the word `Object` must be written exactly as shown here.

See Also

Upgrading the Application Code
Upgrading the Data
Deprecated Fields, and Fields Marked as Obsolete

Transitioning from Codeunit 1 to System Codeunits

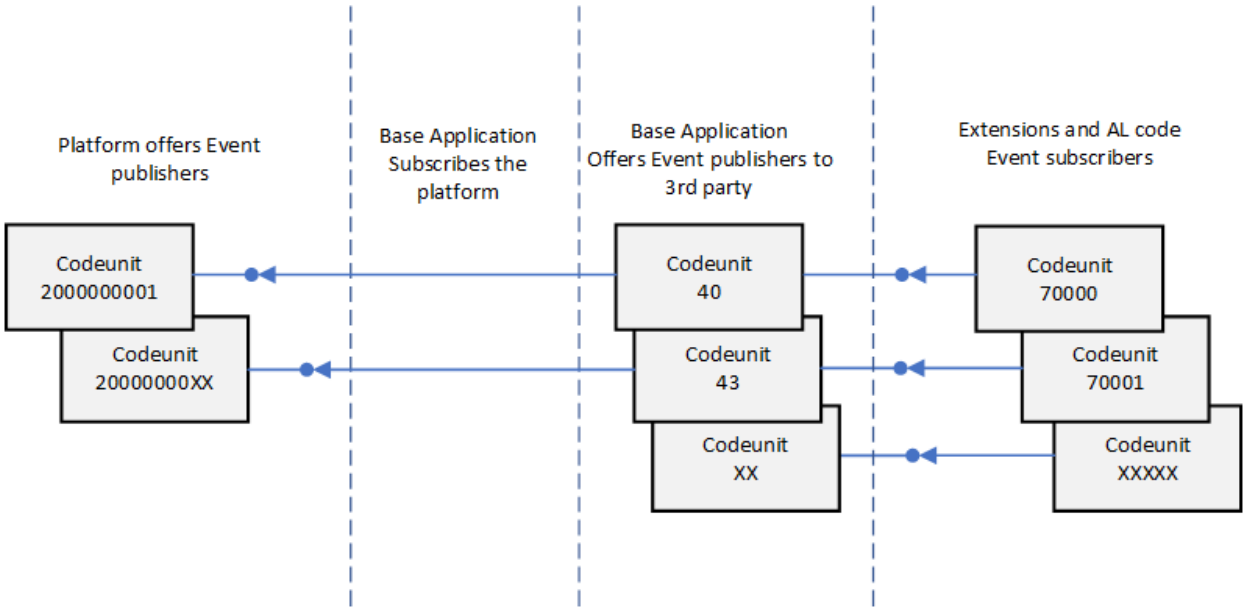
2/17/2021 • 3 minutes to read • Edit Online

With Business Central, codeunit 1 **Application Management** has been removed and replaced with new System codeunits. No functionality has been removed by this change. All system method triggers, event publishers, and their code have been moved to other codeunits.

However, this change will affect the upgrade process from Dynamics NAV and how you develop from now on.

Overview

The foundation of this change is events - publishers and subscribers. System codeunits don't contain code. They only contain event publishers. Instead of running codeunit 1 and calling respective functions, Business Central Server runs system codeunits. The system codeunits will in turn raise published events. There are various management codeunits that subscribe to these events. Like codeunit 1, these subscriber codeunits contain method triggers and integration event publishers. They can call application functionality and raise events. The following figure illustrates the process:



About system codeunits

- They have IDs in the 2 billion range.
- You can't modify them.
- Currently, we don't recommend that code subscribes to the events in the new system codeunits 2000000001..2000000010 directly. Although not blocked, it might be in a future release. Instead, you should subscribe to one of the integration events.

Mapping Codeunit 1 method triggers to events

The following table lists the mappings between the codeunit 1 triggers and the new method triggers/publishers in the management codeunits.

CODEUNIT 1 TRIGGER	NEW CODEUNIT ID	NEW CODEUNIT NAME	NEW METHOD
CompanyClose	40	LogInManagement	CompanyClose

CODEUNIT 1 TRIGGER	NEW CODEUNIT ID	NEW CODEUNIT NAME	NEW METHOD
CompanyOpen	40	LogInManagement	CompanyOpen
GetSystemIndicator	40	LogInManagement	GetSystemIndicator
OnAfterCompanyClose	40	LogInManagement	OnAfterCompanyClose
OnAfterCompanyOpen	40	LogInManagement	OnAfterCompanyOpen
OnBeforeCompanyClose	40	LogInManagement	OnBeforeCompanyClose
OnBeforeCompanyOpen	40	LogInManagement	OnBeforeCompanyOpen
FindPrinter	44	ReportManagement	GetPrinterName
ApplicationVersion	9015	Application System Constants	ApplicationVersion
CustomApplicationVersion	N/A	N/A	
ReleaseVersion	9015	Application System Constants	ReleaseVersion
ApplicationBuild	9015	Application System Constants	ApplicationBuild
CustomApplicationBuild	N/A	N/A	
ApplicationLanguage	43	LanguageManagement	ApplicationLanguage
DefaultRoleCenter	9170	Conf./Personalization Mgt.	DefaultRoleCenterID
MakeDateTimeText	41	TextManagement	MakeDateTimeText
GetSeparateDateTime	41	TextManagement	GetSeparateDateTime
MakeDateText	41	TextManagement	MakeDateText
MakeTimeText	41	TextManagement	MakeTimeText
MakeText	41	TextManagement	MakeText
MakeDateTimeFilter	41	TextManagement	MakeDateTimeFilter
MakeDateFilter	41	TextManagement	MakeDateFilter
MakeTextFilter	41	TextManagement	MakeTextFilter
MakeCodeFilter	41	TextManagement	MakeTextFilter
MakeTimeFilter	41	TextManagement	MakeTimeFilter

CODEUNIT 1 TRIGGER	NEW CODEUNIT ID	NEW CODEUNIT NAME	NEW METHOD
AutoFormatTranslate	45	AutoFormatManagement	AutoFormatTranslate
ReadRounding	45	AutoFormatManagement	ReadRounding
CaptionClassTranslate	42	CaptionManagement	CaptionClassTranslate
GetCueStyle	9701	Cue Setup	GetCueStyle
SetGlobalLanguage	43	LanguageManagement	SetGlobalLanguage
ValidateApplicationlLanguage	43	LanguageManagement	ValidateApplicationLanguage
LookupApplicationlLanguage	43	LanguageManagement	LookupApplicationLanguage
GetGlobalTableTriggerMask	49	GetGlobalTableTriggerMask	
OnGlobalInsert	49	GlobalTriggerManagement	OnGlobalInsert
OnGlobalModify	49	GlobalTriggerManagement	OnGlobalModify
OnGlobalDelete	49	GlobalTriggerManagement	OnGlobalDelete
OnGlobalRename	49	GlobalTriggerManagement	OnGlobalRename
GetDatabaseTableTriggerSetup	49	GlobalTriggerManagement	GetDatabaseTableTriggerSetup
OnDatabaseInsert	49	GlobalTriggerManagement	OnDatabaseInsert
OnDatabaseModify	49	GlobalTriggerManagement	OnDatabaseModify
OnDatabaseDelete	49	GlobalTriggerManagement	OnDatabaseDelete
OnDatabaseRename	49	GlobalTriggerManagement	OnDatabaseRename
OnDebuggerBreak	N/A	N/A	
LaunchDebugger	N/A	N/A	
OpenSettings	9170	Conf./Personalization Mgt.	OpenSettings
OpenContactMSSales	50	SaaS Log In Management	OpenContactMSSales
InvokeExtensionInstallation	2501	ExtensionMarketplaceMgmt	InvokeExtensionInstallation
CustomizeChart	9180	Generic Chart Mgt	CustomizeChart
HasCustomLayout	44	ReportManagement	HasCustomLayout

CODEUNIT 1 TRIGGER	NEW CODEUNIT ID	NEW CODEUNIT NAME	NEW METHOD
MergeDocument	44	ReportManagement	MergeDocument
ReportGetCustomRdlc	44	ReportManagement	ReportGetCustomRdlc
ReportScheduler	44	ReportManagement	ScheduleReport
OnBeforeOpenSettings	9170	Conf./Personalization Mgt.	OnBeforeOpenSettings
OnAfterGetApplicationVersion	9015	Application System Constants	OnAfterGetApplicationVersion
OnRoleCenterOpen	9170	Conf./Personalization Mgt.	OnRoleCenterOpen
OnAfterGetSystemIndicator	79	OnAfterGetSystemIndicator	
OnAfterFindPrinter	44	ReportManagement	OnAfterGetPrinterName
OnAfterGetDefaultRoleCenter	9170	Conf./Personalization Mgt.	OnAfterGetDefaultRoleCenter
OnAfterMakeDateText	N/A	N/A	
OnAfterMakeTimeText	N/A	N/A	
OnAfterMakeText	N/A	N/A	
OnAfterMakeDateTimeFilter	41	LanguageManagement	OnAfterMakeDateTimeFilter
OnAfterMakeDateFilter	41	LanguageManagement	OnAfterMakeDateFilter
OnAfterMakeTextFilter	41	LanguageManagement	OnAfterMakeTextFilter
OnAfterMakeCodeFilter	N/A	N/A	
OnAfterMakeTimeFilter	41	LanguageManagement	OnAfterMakeTimeFilter
OnAfterAutoFormatTranslate	45	AutoFormatManagement	OnAfterAutoFormatTranslate
OnAfterCaptionClassTranslate	42	CaptionManagement	OnAfterCaptionClassTranslate
OnAfterGetGlobalTableTriggerMask	49	GlobalTriggerManagement	OnAfterGetGlobalTableTriggerMask
OnAfterOnGlobalInsert	49	GlobalTriggerManagement	OnAfterOnGlobalInsert
OnAfterOnGlobalModify	49	GlobalTriggerManagement	OnAfterOnGlobalModify
OnAfterOnGlobalDelete	49	GlobalTriggerManagement	OnAfterOnGlobalDelete

CODEUNIT 1 TRIGGER	NEW CODEUNIT ID	NEW CODEUNIT NAME	NEW METHOD
OnAfterOnGlobalRename	49	GlobalTriggerManagement	OnAfterOnGlobalRename
OnAfterGetDatabaseTableTriggerSetup	49	GlobalTriggerManagement	OnAfterGetDatabaseTableTriggerSetup
OnAfterOnDatabaseInsert	49	GlobalTriggerManagement	OnAfterOnDatabaseInsert
OnAfterOnDatabaseModify	49	GlobalTriggerManagement	OnAfterOnDatabaseModify
OnAfterOnDatabaseDelete	49	GlobalTriggerManagement	OnAfterOnDatabaseDelete
OnAfterOnDatabaseRename	49	GlobalTriggerManagement	OnAfterOnDatabaseRename
OnAfterHasCustomLayout	44	ReportManagement	OnAfterHasCustomLayout
OnAfterReportGetCustomRdlc	9650	Edit MS Word Report Layout	OnAfterReportGetCustomRdlc
OnBeforeOnDatabaseInsert	49	GlobalTriggerManagement	OnBeforeOnDatabaseInsert
OnBeforeOnDatabaseModify	49	GlobalTriggerManagement	OnBeforeOnDatabaseModify
OnBeforeOnDatabaseDelete	49	GlobalTriggerManagement	OnBeforeOnDatabaseDelete
OnBeforeOnDatabaseRename	49	GlobalTriggerManagement	OnBeforeOnDatabaseRename
OnEditInExcel	6710	OnEditInExcel	
OnInstallAppPerDatabase	N/A	N/A	
OnInstallAppPerCompany	N/A	N/A	
OnCheckPreconditionsPerDatabase	9900	OnCheckPreconditionsPerDatabase	
OnCheckPreconditionsPerCompany	9900	RaiseOnCheckPreconditionsPerCompany	
OnUpgradePerDatabase	9900	OnUpgradePerDatabase	
OnUpgradePerCompany	9900	OnUpgradePerCompany	
OnValidateUpgradePerDatabase	9900	OnValidateUpgradePerDatabase	
OnValidateUpgradePerCompany	9900	OnValidateUpgradePerCompany	

What does this mean for upgrade?

Full upgrade - application code and data

Move custom logic from the old codeunit 1 to the management codeunits and methods. Change custom code that call into codeunit 1 to call or subscribe to the new methods. You can do this step as part of the application code upgrade.

Technical upgrade

After you convert the old database to the Business Central platform, import and compile the replacement codeunit 1 object. You get the replacement object from Codeunit 1 Replacement.

See Also

Converting a Database

Converting a Database to Business Central Spring 2019 - Technical Upgrade

2/17/2021 • 13 minutes to read • Edit Online

See print-friendly quick reference

This article describes how to convert a database from one of the following versions to the latest Business Central platform:

- Microsoft Dynamics NAV 2015
- Microsoft Dynamics NAV 2016
- Microsoft Dynamics NAV 2017
- Microsoft Dynamics NAV 2018

This article can also be used to update your current Business Central database to the latest cumulative update (CU) platform.

Overview

About technical upgrade and database conversion

Converting a database, which is often referred to as a *technical upgrade*, changes the database so that it works on the latest Business Central platform. The conversion updates the system tables of the old database to the new schema (data structure), and upgrades of all reports to support Report Viewer 2015. It provides you with the latest platform features and performance enhancements.

The process is slightly different when you have multitenant deployment compared to a single-tenant deployment. The steps that follow indicate the differences where applicable.

> **IMPORTANT**
>
> Before you begin, read the article Important Information and Considerations for Before Upgrading. This article contains information about limitations in a technical upgrade, such as using V1 extensions or Dynamics 365 Sales integration.
>
> If you are upgrading a single-tenant database to Business Central Cumulative Update 02, 03, 04, or 05, read Tenant synchronization issue with technical upgrade to Business Central Cumulative Updates 02–05 on the Business Central for Partners blog before starting the upgrade.

Tools

To complete the steps in the article, you will use the following tools:

- Dynamics NAV Development Environment (the version that matches your old database and the new version)
- Dynamics NAV Server Administration tool and/or Business Central Server Administration tool
- SQL Server Management Studio

Task 1: (Dynamics NAV upgrade only) Preparation

1. Transition from the use of codeunit 1

With Business Central, codeunit 1 Application Management is no longer used and has been replaced. For more information, see Transitioning from Codeunit 1. To prepare for this change when doing a technical upgrade, do the following:

a. If you have any custom code in codeunit 1, export the existing codeunit 1 as a .fob or .txt file.
b. Go to Codeunit 1 Replacement, and make a .txt file that includes the replacement code for codeunit 1. You will use this file later.

2. Convert V1 extensions to V2 extensions

   Business Central does not support V1 extensions. If you are updating a Dynamics NAV database that includes custom V1 extensions, and you want to continue to use them, you have to convert them to V2 extensions. For more information, see Converting Extensions V1 to Extensions V2.

   V1 extensions that are produced by Microsoft (first-party extensions, publisher=Microsoft) are now available as V2 extensions on the Business Central installation media (DVD), so you do not have to convert these. If you want to keep the functionality provided and data collected by these V1 extensions, you will have to publish the V2 versions as part of the technical upgrade later in Task 4.

   You will have to uninstall all V1 extension to successfully completes the technical upgrade.

Task 2: Preparing the Old Database

Before you convert the old database to Business Central, complete the following steps.

1. Make a copy of the old database or create full database backup.

   For more information, see Create a Full Database Backup (SQL Server).

2. For single-tenant mode, uninstall all extensions. For multitenant mode, uninstall all V1 extensions.

   You can do this from **Extension Management** page in the Dynamics NAV client or by using the Uninstall-NAVApp cmdlet of the Dynamics NAV Administration Shell.

   To get a list of the extensions that are installed, run this command:

   ```
   Get-NAVAppInfo -ServerInstance <ServerInstanceName> -Tenant <TenantID>
   ```

   For a single-tenant mode, set the `-Tenant` parameter to `default`. V1 extensions are indicated by `ExtensionType: CSIDE`.

   For each extension, run this command to uninstall it:

   ```
   Uninstall-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
   ```

   Alternately, to remove them all at once, you can run this command:

   ```
   Get-NAVAppInfo -ServerInstance <ServerInstanceName> -Tenant default | % { Uninstall-NAVApp -
   ServerInstance <ServerInstanceName> -Name $_.Name -Version $_.Version }
   ```

3. Unpublish extensions versions that you do not want to use in the upgraded database. This is optional and typically done for V1 extensions because they are no longer supported.

   For example:

   ```
   Unpublish-NAVApp -ServerInstance dynamicsnav110 -Name System -Version 11.0.12925.0
   ```

4. Open the Dynamics NAV Development Environment that matches the Dynamics NAV or Business Central version of the old database, and then connect to the old application database.

   For more information, see Open Databases.

5. In Object Designer, verify that all objects are compiled and no objects are locked.

   For more information about compiling objects, see Compiling Objects.

   If one or more objects are locked, the conversion process cannot update the database version number. As a result, the conversion does not complete. For more information, see Locking and Unlocking Objects.

6. On the **Tools** menu, choose **Build Server Application Objects**, and then choose the **Yes** button.

7. If any errors occur, they are shown in the **Error List** window. Make sure that you address all compilation errors before you continue.

8. Run the schema synchronization with validation to synchronize the database schema changes.

   For more information, see Synchronizing the Tenant Database and Application Database.

9. Upload the Business Central Partner license to the database.

   For more information, see Uploading a License File for a Specific Database.

   > **IMPORTANT**
   > The license that you upload must be a developer license. During the conversion, the development environment will convert the report objects that are stored in the old database to the RDL format.

10. (Multitenant only) Dismount tenants.

    Use the Dynamics NAV Server Administration tool or Dismount-NAVTenant cmdlet of the Dynamics NAV Administration Shell to dismount all tenants from the Dynamics NAV Server instance.

    ```
    Dismount-NAVTenant -ServerInstance <serverinstance> -Tenant <tenantID>
    ```

11. Stop the Dynamics NAV Server instance, and close the development environment.

    You can use the Dynamics NAV Server Administration tool or Set-NAVServerInstance cmdlet of the Dynamics NAV Administration Shell.

    To use the Set-NAVServerInstance cmdlet, run the following command:

    ```
    Set-NAVServerInstance –ServerInstance <ServerInstanceName> -Stop
    ```

12. Clear all records from the **dbo.Server Instance** and **dbo.Debugger Breakpoint** tables in the old application database in SQL Server.

    Using SQL Server Management Studio, open and clear the **dbo.Server Instance** and **dbo.Debugger Breakpoint** tables of the old database. For example, you can run the following SQL query:

    ```
    DELETE FROM [<My NAV Database Name>].[dbo].[Server Instance]
    DELETE FROM [<My NAV Database Name>].[dbo].[Debugger Breakpoint]
    ```

13. Close all to connections to the database.

This includes but is not limited to Dynamics NAV client tools and SQL Server Management Studio.

Task 3: Run Technical Upgrade on the Old Database

Next, you will convert the old database so that it can be used with Business Central.

> **TIP**
> If you want to write a script that helps you convert databases, you can use the Invoke-NAVDatabaseConversion function in the Dynamics NAV Development Shell.

> **IMPORTANT**
> Before you run the technical upgrade, delete any corrupt databases that are on the same SQL Server instance as the database that you intend to upgrade. Otherwise, when you run the database conversion, you will get the error "The Symbol Reference field on the Object Metadata table does not exist in the SQL Server table or view.".

1. If the database is on Azure SQL Database, add your user account to the **dbmanager** database role on the master database.

   This membership is only required for converting the database, and can be removed afterwards.

2. Install Business Central.

   Run the Business Central Setup, and install the following components as a minimum:

   - Server
   - SQL Server Database Components
   - Administration Tool
   - Dynamics NAV Development Environment

   > **IMPORTANT**
   > For a multitenant installation, configure the Business Central Server instance to be a multitenant instance.

3. Run the newly installed Dynamics NAV Development Environment as an administrator.

   - If the Dynamics NAV Development Environment is already connected to the old application database, a dialog box about converting the database appears. Go to the next step.
   - Otherwise, connect to the old application database that you prepared in the previous task, and then go to the next step.

   For more information, see Open Databases.

   > **IMPORTANT**
   > If you do not run the development environment as an administrator, you will get an error and the conversion will be stopped.

4. In the dialog box that appears, read the instructions about converting the database carefully because this action cannot be reversed. When you are ready, choose the **OK** button, and then choose the **OK** button to confirm that you want to convert the database.

   Dynamics NAV Development Environment will now convert the database. This includes an upgrade of

system tables and reports.

5. When you are notified that the conversion was successful, choose the **OK** button.
6. If the database references any assemblies (such as client control add-ins) that are not included on the Business Central installation media (DVD), then add the assemblies to the Add-ins folder on Business Central Server.

   For Business Central Server, the default path is the C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins folder.
7. Connect a Business Central Server instance to the converted database.

   Use the Business Central Server Administration tool or the Set-NAVServerConfiguration cmdlet to connect a Business Central Server instance to the converted database.

   > **IMPORTANT**
   > The service account that is used by the Business Central Server instance must be a member of the **db_owner** role in the Dynamics NAV database on SQL Server or Azure SQL Database.

   For more information, see Connect a Server Instance to a Database and Giving the account necessary database privileges in SQL Server.
8. Go to the development environment, and set it to use the Business Central Server instance that connects to the database.

   For more information, see Change the Server Instance.
9. If upgrading from Dynamics NAV, import the codeunit 1 replacement text file you created earlier.
10. Compile all objects without table schema synchronizing (**Synchronize Schema** set to **Later**); you will do this later.

    For more information, see Compiling Objects.
11. Fix compilation errors.

    If any errors occur, they are shown in the **Error List** window. For help on resolving the errors, see the following:

    - Resolving Compilation Errors

    You can find all objects which did not compile in the **Object Designer** window, by setting a field filter on the **Compiled** field.
12. Recompile V2 extensions that you uninstalled previously.

    Use the Repair-NAVApp cmdlet of the Business Central Administration Shell to compile the published extensions to make sure they are work with the new platform.

    For example, you can run the following command to recompile all extensions:

    ```
    Get-NAVAppInfo -ServerInstance <ServerInstanceName> | Repair-NAVApp
    ```
13. (Multitenant only) Mount the tenant.

    Use the Mount-NAVTenant cmdlet.

```
Mount-NAVTenant -ServerInstance <serverinstance> -Tenant <tenantID> -DatabaseName
<tenantdatabasename>
```

`-AllowAppDatabaseWrite` is optional but is required for some post-upgrade tasks, like upgrading the control add-ins. When you are done upgrading, you can dismount and mount the tenant without this parameter as needed.

14. Run the schema synchronization with validation to complete the database conversion.

    For more information, see Synchronizing the Tenant Database and Application Database.

Task 4: Post-upgrade

1. Upgrade Javascript-based control add-ins to new versions.

   The Business Central Server installation includes new versions of Microsoft-provided Javascript-based control add-ins, such as the Business Chart control add-in. If you application is using any of these add-ins, you must upgrade them to the new versions as follow:

   a. Open the Business Central client.

   b. Search for and open the **Control Add-ins** page.

   c. Choose **Actions** > **Control Add-in Resource** > **Import**.

   d. Locate and select the .zip file for the control add-in and choose **Open**.

      The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There is a sub-folder for each add-in. For example, the path to the Business Chart control add-in is

      ```
      C:\Program Files\Microsoft Dynamics 365 Business Central\140\Service\Add-
      ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
      ```

      .

   e. After you have imported all the new control add-in versions, restart Web Server instance.

2. (Single tenant only) Install the V2 extensions that you uninstalled previously.

   Use the Install-NAVApp cmdlet to compile the published extensions to make sure they are work with the new platform.

   For each V2 extension, run the following command to install it:

   ```
   Install-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
   ```

3. (Dynamics NAV 2017 or 2018 upgrade only) If the old database used first-party V1 extensions (publisher Microsoft), then you should publish and install the corresponding V2 extensions that are found on the installation media (DVD). For example, **Sales and Inventory Forecast** and **PayPal Payment Standards** were available as V1 extensions. Now, they are available as V2 extensions.

   a. Publish the system.app and test.app symbol files.

      If you installed the **AL Development Environment**, you can find the symbol files where you installed the environment, which by default is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160. Otherwise, you can find the files in the **ModernDev** folder on the installation media.

      To publish the symbols, open the Business Central Administration Shell as an administrator, and run the following command for each of the symbol files:

```
Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <SymbolFilePath> -PackageType
SymbolsOnly
```

b. Generate the application symbol references by using the finsql.exe file.

   Open a command prompt as an administrator, change to the directory where the `finsql.exe` file has been installed as part of Dynamics NAV Development Environment, and then run the following command:

```
finsql.exe Command=generatesymbolreference,  ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database="<MyDatabaseName>"
```

   Replace values for the `Database` and `ServerName` settings to suit.

   If the application database contains test objects (ID 130000-139999), then make sure to exclude these objects when generating symbols. You can do this by using the `-Filter` parameter and running the command twice:

```
finsql.exe command=generatesymbolreference, ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database="<MyDatabaseName>, filter="Object ID=1..129999"
```

```
finsql.exe command=generatesymbolreference, ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database="<MyDatabaseName>, filter="Object ID=140000..1999999999"
```

   NOTE

   This command does not generate a file. It populates the **Object Metadata** table in the database.

   When you run the command, the console returns to an empty command prompt, and does not display or provide any indication about the status of the run. However, the finsql.exe may still be running in the background. It can take several minutes for the run to complete, and the symbols will not be generated until such time. You can see whether the finsql.exe is still running by using Task Manager and looking on the **Details** tab for **finsql.exe**.

   When the process ends, a file named **navcommandresult.txt** is saved to the Dynamics NAV Client connected to Business Central installation folder. If the command succeeded, the file will contain text like `[0] [06/12/17 14:36:17] The command completed successfully in '177' seconds.` If the command failed, another file named **naverrorlog.txt** will be generated. This file contains details about the error(s) that occurred.

   For more information about generation symbols, see Running C/SIDE and AL Side-by-Side.

c. Configure the **Enable loading application symbol references at server startup** (EnableSymbolLoadingAtServerStartup) setting on the Business Central Server instance, and restart the instance.

   For more information, see Configuring Business Central Server.

d. Publish the new V2 extension by running the Publish-NAVApp cmdlet for each extension:

```
Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <ExtensionFileName>
```

e. Synchronize the schema with the database by running the Sync-NAVApp cmdlet for each extension:

```
Sync-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

f. For each V2 extension, run the following command to install it:

```
Install-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

4. (Dynamics NAV upgrade only) Transition the custom code in the old codeunit 1 to use the new system event implementation.

   For more information, see Transitioning from Codeunit 1.

5. (Microsoft Dynamics NAV 2016 upgrade only) Modify C/AL code to ensure that the **My Settings** page works properly in the Business Central Web client.

   For more information, see Resolving My Settings Page Implementation After a Database Conversion.

6. (Dynamics NAV upgrade only) Configure pages and reports included in the MenuSuite to be searchable in the Web client.

   The MenuSuite is no longer used to control whether a page or report can be found in the search feature of the Web client. This is now determined by specific properties on the page and report objects. For more information, see Making Pages and Reports Searchable in Web client After an Upgrade.

7. Build the object search index to make objects able to be searched from **Tell Me** in the client. If you completed step 6, you can skip this step.

   In the **Tools** menu of the Dynamics NAV Development Environment, select **Build Object Search Index**.

8. Upload the customer license to the converted database.

   For more information, see Uploading a License File for a Specific Database.

   You have now completed the conversion of the database to be accessed from Business Central. To test the converted database, you can connect it to the Business Central Server instance that is used by Dynamics NAV clients, and then open a client.

Database and Windows collations

Starting from SQL Server 2008, SQL Server collations are fully aligned with the collations in Windows Server. If you upgrade to Business Central from Microsoft Dynamics NAV 2009, the step to convert the database includes upgrading the database from using SQL collations to using Windows collation. This collation change provides users with the most up-to-date and linguistically accurate cultural sorting conventions. For more information, see Collation and Unicode Support.

See Also

Upgrading the Application Code Upgrading the Data
Upgrading to Business Central

Business Central Technical Upgrade Quick Reference

2/17/2021 • 2 minutes to read • Edit Online

This article provides an overview of the technical upgrade process for Business Central. For more detailed steps, see Technical Upgrade.

Prerequisites

STEP	MORE INFO	DONE
Convert custom V1 extensions to V2 extensions.	See...	
Transitioning from codeunit 1.	See...	

Prepare the old application database

STEP	MORE INFO	DONE
Backup the database(s).	See...	
(Single-tenant mode only) Uninstall all extensions. (Multitenant mode) Unintsall all V1 extensions.	See...	
(Optional) Unpublish unwanted extension versions .	See...	
Ensure all objects are compiled, unlocked, and tables are synchronized.	See...	
Upload a Business Central partner license.	See...	
(Multitenant mode only) Dismount the tenant	See...	
Stop the Dynamics NAV or old Business Central Server Instance		
Clear the **dbo.Server Instance** and **dbo.Debugger Breakpoint** tables in SQL Server.	See...	
Close all connections to the database.		

Run the technical upgrade

STEP	MORE INFO	DONE
Install Business Central.	See...	
Open the Dynamics NAV Development Environment as an administrator.		
Connect to and convert the application database.	See...	
Add custom control add-ins to the server instance.	See...	
Connect a Business Central Server instance to the converted application database.	See...	
Connect development environment to the server instance.	See...	
Import codeunit 1 replacement.	See...	
Compile all objects. **Important:** Choose to synchronize schema **later**.	See...	
Fix compilation errors.	See...	
Repair published V2 extensions.	See...	
(Multitenant mode only) Mount the tenant database.	See...	
Synchronize the tenant/database.	See...	

Post-upgrade tasks

STEP	MORE INFO	DONE
Upgrade Javascript-based control add-ins to new versions available on Business Central Server.	See...	
(Single-tenant mode only) Install the V2 extensions that were previously uninstalled.	See...	
If the old database used first-party V1 extensions, publish and install the V2 extensions that replace them. **Important** As part of this step, make sure to publish the system and test symbols and generate application symbols.	See...	

STEP	MORE INFO	DONE
Transition custom code from old codeunit 1 to management codeunits. (Dynamics NAV 2018 and earlier)	See...	
Configure pages and reports included in the MenuSuite to be searchable in the Web client (Dynamics NAV 2018 and earlier)	See...	
Build object search index.		
Upload the customer license.	See...	

Upgrading the Application Code in Dynamics 365 Business Central

2/17/2021 • 18 minutes to read • Edit Online

Typically, customers want all the customizations that have been implemented in their existing databases to be migrated to their new Business Central databases. Depending on the version of Business Central that a database is being upgraded from, the amount of code changes between the two versions can vary. To upgrade the application code, you must merge code from different versions of the application. This merge process is known as a *code upgrade* or *application upgrade*. You must upgrade the application before you upgrade the data.

> **IMPORTANT**
>
> Before you begin, read the article Important Information and Considerations for Before Upgrading. This article contains information about limitations and things that might require you to perform extra tasks before you upgrade, such as the use of extensions V1 and the deprecation of codeunit 1.

Application Upgrade Overview

During an upgrade, you have to first identify which changes you have to make, and then you'll have to upgrade the application objects and the application code, and finally, you might have to upgrade data so that it fits the new database schema.

For the application portion of the upgrade, you must analyze and process code changes by comparing and merging three separate versions of the database:

VERSION	DESCRIPTION
Original version	This is the baseline version of the solution that you want to upgrade, such as the original release of Business Central October 2018 or Microsoft Dynamics NAV 2018.
Modified version	This is the version that you want to upgrade, such as a customer's Business Central October 2018 or Microsoft Dynamics NAV 2018 database with customizations and add-on solutions.
Target version	This is the target of the merge process that you want to upgrade your application to, such as the standard version of the Business Central database.

When you merge the application objects from these three versions, you can import the result into a new Business Central database that then contains the upgraded application. At the end of the process, you export the merged Business Central objects from this database to a .fob file that you will use during the data upgrade.

Single-tenant and multitenant deployments

The process for upgrading the application code is basically the same for a single-tenant and multitenant deployment. However, there are some inherent differences because with a single-tenant deployment, the application and business data is included in the same database, while with a multitenant deployment application code is in a separate database than the business data (tenants). Here is the general process for each deployment type. In the tasks that follow this section, tasks are marked as *Single-tenant only* or *Multitenant only* where

applicable.

Single-tenant

1. Upgrade the application code.
2. Create a new database on the new platform.
3. Import the upgraded application to the new database.
4. Export the application to a .fob file.
5. Upgrade the data. Here you will import the .fob file.

Multitenant

1. Upgrade the application code.
2. Create a new database on the new platform.
3. Import the upgraded application to the new database.
4. Upgrade the data by mounting the tenant on the application database.

With a multitenant deployment, you will perform the steps in this article on the application database, that is, the database that contains the application object definitions.

Different ways of upgrading application code

You can use any tool or set of tools to help you compare and merge code. Dynamics NAV and Business Central include Windows PowerShell cmdlets and sample scripts that can help you upgrade your application. The cmdlets are available through the Microsoft Dynamics NAV Development Shell and Dynamics NAV Development Shell, or by importing the Microsoft.Dynamics.NAV.Model.Tools.psd1 module into the Windows PowerShell Integrated Scripting Environment (ISE). You can find the sample scripts on the product installation media, in the *WindowsPowerShellScripts\ApplicationMergeUtilities* folder. We recommend that you use these cmdlets and sample scripts because they can make it faster to merge most changes. For example, you can combine several steps in a command that uses a cmdlet such as the Merge-NAVApplicationObject. The sections in this article describe how you can use the Merge-NAVApplicationObject cmdlet and other Windows PowerShell cmdlets. For more information, see Comparing and Merging Application Object Source Files.

Task 1: Install the Prerequisites and Tools

To complete the tasks in this article, you will use various tools and components of the old Dynamics NAV version and Business Central. Ensure that you have the following installed:

Dynamics NAV to Business Central upgrade

PRODUCT	TOOL/COMPONENT
Old Dynamics NAV version	• Dynamics NAV Development Environment • Dynamics NAV Development Shell
Business Central	• Business Central Server • Dynamics NAV Development Shell(**DISCONTINUED AFTER:** Business Central Spring 2019) • Business Central Administration Shell • Dynamics NAV Development Environment (**DISCONTINUED AFTER:** Business Central Spring 2019)

Business Central to Business Central upgrade

PRODUCT	TOOL/COMPONENT
Old Business Central version	• Dynamics NAV Development Environment • Dynamics NAV Development Shell
New Business Central version	• Business Central Server • Dynamics NAV Development Shell(**DISCONTINUED AFTER:** Business Central Spring 2019) • Business Central Administration Shell • Dynamics NAV Development Environment(**DISCONTINUED AFTER:** Business Central Spring 2019)

Task 2: Prepare the Application Object Text Files

You must prepare text files that contain the application objects for the different application versions previously described (original, modified, and target). The text files provide the input for the application merge process.

There are three ways to export application objects to text files:

- Use the Dynamics NAV Development Environment version that matches the application database version.

  For more information see To export objects by using the development environment UI.

- Use the finsql.exe that matches the application database version to run the ExportObjects command.

  For more information, see To export objects by running finsql.exe with the ExportObjects command .

- Use the Microsoft Dynamics NAV Development Shell or Dynamics NAV Development Shell version that matches the application database version.

  This is the way that is described in the tasks of this article. Note that the Microsoft Dynamics NAV Development Shell is not available for Microsoft Dynamics NAV 2013 and Microsoft Dynamics NAV 2013 R2. For these versions, you must use development environment or finsql.exe.

Create the application text files

> **NOTE**
> Be sure to upload a valid developer license to the database before doing the following steps.

1. Create four folders on the computer, and name them as follows:

   - ORIGINAL

     This folder will be used to store the application object text file(s) from the baseline version, such as the original release of Business Central October 2018, Microsoft Dynamics NAV 2018, or Microsoft Dynamics NAV 2017.

   - MODIFIED

     This folder will be used to store the application object text file(s) from the modified version, such as the customer's database.

   - TARGET

     This folder will be used to store the application object text file(s) from the latest Business Central version.

- **RESULT**

  This folder will be used to store the application object text file(s) that are the result of the application merge. It will also contain zero or more .CONFLICT files that describe conflicting code.

2. Export all application objects except system tables from the original version of the old application database, such as the original Microsoft Dynamics NAV 2018 database.

   Do not export system tables, which have the IDs in the 2000000000 range. Name the file **OldBaseVersion.txt**, and then save the file in the **ORIGINAL** folder that you created earlier.

   For example, start the Microsoft Dynamics NAV Development Shell version that matches the database version, and run the **Export-NAVApplicationObject** function as follows:

   ```
   Export-NAVApplicationObject –DatabaseServer MyServer –DatabaseName "Demo Database NAV (11-0)" –Path
   C:\Upgrade\ORIGINAL\OldBaseVersion.txt -Filter 'Id=1..1999999999'
   ```

3. Export all application objects, except system tables, from the old modified application database, such as the customer's customized Microsoft Dynamics NAV 2018 database.

   Name the file **OldCustomVersion.txt**, and then save the file in the **MODIFIED** folder that you created earlier.

   For example (using the Microsoft Dynamics NAV Development Shell version that matches the database version), if the customer's database is called *MyCustomerNAV2016Database*, you can run the following command:

   ```
   Export-NAVApplicationObject –DatabaseServer MyServer –DatabaseName "MyCustomerNAV2018Database" –Path
   C:\Upgrade\MODIFIED\OldCUSTOMVersion.txt -Filter 'Id=1..1999999999'
   ```

   > **TIP**
   > In some cases, existing customizations might be irrelevant after the upgrade because they correspond to new functionality in Business Central.

4. Export all application objects, except system tables, from the new base version, such as the original Business Central database.

   Name the file **NewBaseVersion.txt**, and then save the file in the **TARGET** folder that you created earlier.

   For example, using the Dynamics NAV Development Shell for Business Central, run the following command:

   ```
   Export-NAVApplicationObject –DatabaseServer MyServer –DatabaseName "Demo Database BC (14-0)" –Path
   C:\Upgrade\Target\NewBaseVersion.txt -Filter 'Id=1..1999999999'
   ```

Optionally, you can use the Split-NAVApplicationObjectFile cmdlet to split each text file into separate text files for each application object. This can make it easier to keep track of the process. The end result at this stage is three folders with one or more text files that contain the three sets of application objects that you want to merge.

Task 3: Merge Versions

You now merge the three sets of application objects to create the application for the new database. This section illustrates how to do this by using the Merge-NAVApplicationObject cmdlet.

The product installation media contains sample scripts that provide examples of how you can use the Merge-NAVApplicationObject cmdlet to merge application objects. For more information, see Merge Application Changes.

> **NOTE**
> In certain scenarios, you can choose to use the Compare-NAVApplicationObject cmdlet to identify the changes between the existing customized application and the new application. You can then choose to use the Update-NAVApplicationObject cmdlet to apply all or some of the changes to the new version. For more information, see Compare and Update Application Object Source Files. However, we recommend that you use the Merge-NAVApplicationObject cmdlet in most cases.

Merge the application object versions into text files

1. Run the new Dynamics NAV Development Shell as an administrator.
2. At the command prompt, change to the directory that contains the four folders that contain the application text files, and then run the following command:

   ```
   Merge-NAVApplicationObject -OriginalPath ORIGINAL -TargetPath TARGET -ModifiedPath MODIFIED -
   ResultPath RESULT
   ```

   For example:

   ```
   Merge-NAVApplicationObject -OriginalPath C:\Upgrade\ORIGINAL -TargetPath C:\Upgrade\TARGET -
   ModifiedPath C:\Upgrade\MODIFIED -ResultPath C:\Upgrade\RESULT
   ```

Depending on the number of objects that you are merging and the number of differences found, this can take a few seconds, a few minutes, or longer. When the cmdlet completes, the result of the merge is shown, including a description of any application objects with conflicting code. The **RESULT** folder will contain a text file (.TXT) for each merged application object and possibly one or more .CONFLICT files that describe the code conflicts that occurred during the merge.

At this point, you can either go to Task 4 to analyze and eventually resolve the conflicts, or you can go directly to Task 5 to import the merged objects as-is from the **RESULT** folder to the new Business Central database.

Task 4: Handling Conflicts

Depending on the application that you are upgrading, you can choose to analyze and fix the conflicting code before you import the merged objects into the Dynamics NAV Development Environment for Business Central. The conflicts are shown in the merged text files but are also identified in .CONFLICT files in the subfolders of the **RESULT** folder. The subfolders **ConflictOriginal**, **ConflictModified**, and **ConflictTarget** folders then contain copies of the source files from the versions that have conflicting code.

You can analyze the conflicts in any tool, make the relevant changes, and then run the merge operation again. For more information, see Handling Merge Conflicts. Alternatively, you can go directly to task 5 to import the merged files into the Dynamics NAV Development Environment, and resolve the conflicts there.

Task 5: Import and Compile Merged Objects in an Empty Database

After you have completed the merge, you import the new merged application objects as text files into a new (empty) Business Central database, and then compile all objects. You must resolve any compilation errors before you can continue. The text files include successfully merged code, and code that is partially merged. You can import the partially merged objects into the Business Central development environment and resolve the

conflicts there.

1. Create a new Business Central database for the new upgraded application. The database should be empty, except for the system tables.

   For example, give the database the name *My Upgraded App*. For more information, see Creating and Altering Databases.

   > **IMPORTANT**
   > You must set the collation of the new database to match the collation of the old application database. To see the collation of the old database, open it in the old Dynamics NAV Development Environment version, then choose **File** > **Database** > **Alter** > **Collation**.

2. Make sure the database includes a valid Business Central license.

   For more information, see Uploading a License File for a Specific Database

3. Import the new merged application object text files (.TXT) from the **Result** folder into the new database.

   There are three ways to import the files:

   - Use the Dynamics NAV Development Environment for Business Central.

     For more information see To import objects by using the development environment UI.

   - Use the finsql.exe to run the ImportObjects command.

     For more information, see To import objects by running finsql.exe with the ImportObjects command .

   - Use the Dynamics NAV Development Shell (or Microsoft.Dynamics.NAV.Model.Tools.psd1 module).

     The shell includes the **Join-NAVApplicationObjectFile** cmdlet and **Import-NAVApplicationObject** function. The **Join-NAVApplicationObjectFile** cmdlet combines multiple application object text files into one text file. The **Import-NAVApplicationObject** function runs the ImportObjects command to import an object file.

     This means that you can run a command similar to following to create a single text file from the merge application text files in the **Result** folder:

     ```
     Join-NAVApplicationObjectFile –Source C:\Upgrade\RESULT\*.txt -Destination C:\Upgrade\all-merged.txt
     ```

     Then, you can run this command to import the text file:

     ```
     Import-NAVApplicationObject –DatabaseServer MyServer –DatabaseName "My Upgraded App" –Path C:\Upgrade\all-merged.txt
     ```

4. Connect the new Business Central Server instance to the database.

   You can do this with the Business Central Server Administration tool or the Set-NAVServerConfiguration cmdlet in the Dynamics NAV Administration Shell. In addition, you must add the service account that is used by the Business Central Server instance as a member of the **db_owner** role in the Business Central database on SQL Server.

   For more information about how to do this using the Business Central Server Administration tool, see How to: Connect a Microsoft Dynamics NAV Server Instance to a Database and Giving the account

necessary database privileges in SQL Server.

5. Compile all the newly imported objects. Choose to synchronize **later**.

   You can use the Dynamics NAV Development Environment or finsql.exe. For more information, see Compiling Objects.

   If you use the Dynamics NAV Development Environment, you will first have to set it to use the Business Central Server instance that connects to the database. For more information, see Change the Server Instance Used in C/SIDE.

   When you compile the objects, an error is thrown for each code conflict, and you can use the tools that are available in the development environment to resolve the conflicts.

Task 6: Check/change the application family and version

The application and tenant databases are tagged with `Family` and `Version`. To perform the data upgrade, the `Family` on the application must match that tenant's `Family`. The `Version` of the application must be greater than or equal to the tenant's `Version`. The easiest way to ensure that `Family` and `Version` of the upgraded application are compatible for data upgrade is to set `Family` to the same value as the old application, and set the `Version` to a higher value than the old application.

To get the `Family` and `Version`, use the Get-NAVApplication cmdlet, for example:

```
Get-NAVApplication -ServerInstance BC
```

To set the `Family` and `Version`, use the Set-NAVApplication cmdlet. For example, to set the family, run the following command:

```
Set-NAVApplication -ServerInstance <ServerInstanceName> -ApplicationFamily <Family>
```

To increase the version by 1, run the following command:

```
Set-NAVApplication -ServerInstance <ServerInstanceName> -IncrementApplicationVersion
```

Or, to specify change to another version, run the following command:

```
Set-NAVApplication -ServerInstance <ServerInstanceName> -ApplicationVersion <N.N.N.N> -Force
```

Task 7: (Dynamics NAV upgrade only) Configure pages and reports to be searchable

The MenuSuite is no longer used to control whether a page or report can be found in the search feature of the Web client. This is now determined by specific properties on the page and report objects. This task is not required at this point, and can be done after the data upgrade as well.

For more information, see Making Pages and Reports Searchable After an Upgrade.

Task 8: Build object search index

Build the object search index to make objects able to be searched from **Tell Me** in the client. If you completed step 7, you can skip this step.

In the **Tools** menu of the Dynamics NAV Development Environment, select **Build Object Search Index**.

For more information, see Making Pages and Reports Searchable After an Upgrade.

Task 9. (Dynamics NAV upgrade only) Transition the custom code from old codeunit 1 to use the new implementation

Because codeunit 1 has been deprecated in Business Central, you must move any custom logic that was included in the old codeunit 1 into the management codeunits and methods described in the article Transitioning from Codeunit 1.

You now have a new database with a fully upgraded application. For a multitenant deployment, you can start the data upgrade. For this, you will use the new server instance that connects to the upgraded application database. See Upgrading the Data.

Task 10: (Single-tenant mode only) Export all objects

With a single-tenant deployment, export all objects of the new database to a .fob type file, such as **objects.fob** file. You will use this .fob file as part of the data upgrade process. The export must include customized objects, upgraded reports, and all other Business Central objects.

As with exporting objects in Task 1, you can use either the development environment, finsql.exe, or Dynamics NAV Development Shell.

With the Dynamics NAV Development Shell, you can run a command that is similar to the following:

```
Export-NAVApplicationObject c:\Upgrade\objects.fob -DatabaseName "My Upgraded App" -DatabaseServer
[server_name]\[database_instance]
```

This completes the upgrade of the application code for single-tenant deployment. Next, you must upgrade the data in the database. See Upgrading the Data.

Task 11: (Multitenant mode only) Import the upgrade toolkit objects

The upgrade toolkit includes upgrade codeunits for handling the data upgrade.

For W1 versions, you can find the default upgrade toolkit objects in the **UpgradeToolKit\Data Conversion Tools** folder on the Business Central installation media (DVD). Choose the FOB that matches the Dynamics NAV version from which you are upgrading:

FROM	TO BUSINESS CENTRAL APRIL 2019	TO BUSINESS CENTRAL OCTOBER 2018
Microsoft Dynamics NAV 2015	Upgrade80014x.FOB	Upgrade800130.FOB
Microsoft Dynamics NAV 2016	Upgrade90014x.FOB	Upgrade900130.FOB
Microsoft Dynamics NAV 2017	Upgrade100014x.FOB	Upgrade1000130.FOB
Microsoft Dynamics NAV 2018	Upgrade110014x.FOB	Upgrade1100130.FOB
Business Central Fall 2018	Upgrade13x14x.FOB	Not applicable

For local versions, you will find the upgrade toolkit objects in the **UpgradeToolKit\Local Objects** folder. The files follow the same naming convention except they include the 2-letter local version, such as **Upgrade110014x.DK.fob** for Denmark or **Upgrade110014x.DE.fob** for Germany.

For information about importing objects, see Importing Objects.

Task 12: (Multitenant mode only) Publish extensions

1. Unpublish the existing system, test, and application symbols by using the Unpublish-NAVAPP cmdlet:

```
Unpublish-NAVApp -ServerInstance <ServerInstanceName> -Name <name> -Version <n.n.n.n>
```

2. Publish the system and test symbols.

   Symbols are a prerequisite for extensions. If you installed the **AL Development Environment**, you can find the symbol files where your installed the environment, which by default is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160. Otherwise, you can find the files in the **ModernDev** folder on the installation media.

   To publish the symbols, open the Business Central Administration Shell as an administrator, and run the following command for each of the symbol files:

```
Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <SymbolFilePath> -PackageType SymbolsOnly
```

3. Generate the application symbol references by using the finsql.exe file as follows:

   a. Make sure that **Enable loading application symbol references at server startup** (EnableSymbolLoadingAtServerStartup) is set on the Business Central Server instance.

      For more information, see Configuring Dynamics NAV Server.

   b. Open a command prompt as an administrator, change to the directory where the `finsql.exe` file has been installed as part of Dynamics NAV Development Environment, and then run the following command:

```
finsql.exe Command=generatesymbolreference, Database="<MyDatabaseName>", ServerName=
<DatabaseServerName>\<DatabaseInstance>
```

      Replace values for the `Database` and `ServerName` settings to suit.

      If the application database contains test objects (ID 130000-139999), then make sure to exclude these objects when generating symbols. You can do this by using the `-Filter` parameter and running the command twice:

```
finsql.exe command=generatesymbolreference, ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database=<MyDatabaseName>, filter="Object ID=1..129999"
```

```
finsql.exe command=generatesymbolreference, ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database=<MyDatabaseName>, filter="Object ID=140000..1999999999"
```

      > **NOTE**
      > This command does not generate a file. It populates the **Object Metadata** table in the database.

   c. When you run the command, the console returns to an empty command prompt, and does not display or provide any indication about the status of the run. However, the finsql.exe may still be

running in the background. It can take several minutes for the run to complete, and the symbols will not be generated until such time. You can see whether the finsql.exe is still running by using Task Manager and looking on the **Details** tab for **finsql.exe**.

When the process ends, a file named **navcommandresult.txt** is saved to the Dynamics NAV Client connected to Business Central installation folder. If the command succeeded, the file will contain text like `[0] [06/12/17 14:36:17] The command completed successfully in '177' seconds.` If the command failed, another file named **naverrorlog.txt** will be generated. This file contains details about the error(s) that occurred.

For more information about generation symbols, see Running C/SIDE and AL Side-by-Side.

4. Publish new versions of the Microsoft extensions.

   The Business Central installation media (DVD) includes several new versions of Microsoft extensions (that is, extensions that have **Microsoft** as the publisher). If your old deployment uses these extensions, you have to upgrade the old versions to the new versions.

   IMPORTANT

   For Denmark (DK) and German (DE) versions. Some of the local functionality has been moved from the base application to extensions.

   If you are upgrading from a Denmark (DK) version of Dynamics NAV 2017 or earlier, you must publish and install the following extensions to get the local functionality:

NAME	EXTENSION PACKAGE
OIOUBL	OIOUBL.app
Payroll Data Import Definitions (DK)	ImportDKPayroll.app
Payment and Reconciliation Formats (DK)	FIK.app
Tax File Formats (DK)	VATReportsDK.app

   If you are upgrading from a German (DE) version of Dynamics NAV or Business Central October 2018 (Cumulative Update 2 or earlier), you must publish and install the following extensions to get the local functionality:

NAME	EXTENSION PACKAGE
ELSTER VAT Localization for Germany	Elster.app

   The new versions are found in the `\Extensions` folder of the installation media.

   To publish the new extension version, run the Publish-NAVApp cmdlet:

   ```
   Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <ExtensionFileName>
   ```

See Also

Upgrading the Data
Upgrading to Business Central

Upgrading the Data to Business Central: Single-Tenant Deployment

2/17/2021 • 20 minutes to read • Edit Online

See print-friendly quick reference

This article describes the tasks required for upgrading the data of a Dynamics NAV or Business Central database to a Business Central major version or cumulative update.

This article pertains to a single-tenant deployment. For upgrade instructions for a multitenant deployment, see Upgrading the Data: Multitenant Deployment.

About Data Upgrade

You use data conversion tools provided with Business Central to convert the old data with the old version's table and field structure, so that it functions together with the new version's table and field structure. Mainly, only table objects and table data are modified during the data upgrade process. Other objects, such as pages, reports, codeunits, and XMLports are upgraded as part of the application code upgrade process.

The data upgrade process described in this article leads you through the database conversion (technical upgrade) and then the upgrade of the actual data, which is achieved by using the upgrade toolkit/upgrade codeunits.

Prerequisites

Before you start the upgrade tasks, make sure you meet the following prerequisites:

1. Your computer uses the same codepage as the data that will be upgraded.

   If you use conflicting codepages, some characters will not display in captions, and you might not be able to access the upgraded database. This is because Dynamics NAV must remove incorrect metadata characters to complete the data upgrade. In this case, after upgrade, you must open the database in the development environment on a computer with the relevant codepage and compile all objects. This adds the missing characters again.

   Optionally, you can export the captions before the upgrade. For more information, see How to: Add Translated Strings for Conflicting Text Encoding Formats.

2. (Upgrading from Dynamics NAV only) Custom V1 extensions used in the old deployment have been converted to V2 extensions.

   For more information, see Converting Extensions V1 to Extensions V2.

3. You have upgraded the application code, and have the FOB files that contain the upgraded application code.

   For more information about upgrading the application code, see Upgrading the Application Code.

4. You have the upgrade toolkit FOB files for the version that you are upgrading from.

   The upgrade toolkit includes upgrade codeunits for handling the data upgrade. The upgrade toolkit can be in the same FOB file as the application code or in a separate FOB file.

   For W1 versions, you can find the default upgrade toolkit objects in the **UpgradeToolKit\Data**

Conversion Tools folder on the Business Central installation media (DVD). Choose the FOB that matches the Dynamics NAV version from which you are upgrading:

FROM	TO BUSINESS CENTRAL APRIL 2019	TO BUSINESS CENTRAL OCTOBER 2018
Microsoft Dynamics NAV 2015	Upgrade80014x.FOB	Upgrade800130.FOB
Microsoft Dynamics NAV 2016	Upgrade90014x.FOB	Upgrade900130.FOB
Microsoft Dynamics NAV 2017	Upgrade100014x.FOB	Upgrade1000130.FOB
Microsoft Dynamics NAV 2018	Upgrade110014x.FOB	Upgrade1100130.FOB
Business Central Fall 2018	Upgrade13x14x.FOB	Not applicable

For local versions, you will find the upgrade toolkit objects in the **UpgradeToolKit\Local Objects** folder. The files follow the same naming convention except they include the 2-letter local version, such as **Upgrade110014x.DK.fob** for Denmark or **Upgrade110014x.DE.fob** for Germany.

5. You have exported the customized permission sets (except SUPER) and permissions from the old database that you want to reuse in the upgraded database.

   - When upgrading from Dynamics NAV

     To export permission sets and permissions, you run running XMLPort 9171 and 9172.

     It is important that you exclude the SUPER permission set when running XMLPort 9171. You can do this by adding the filter `Role ID is <>SUPER`.

     For more information, see Exporting and Importing Permission Sets and Permissions.

   - When upgrading from an earlier version of Business Central

     In the client, search for and open the **Permission Sets** page, select the user-defined permission sets that you want to keep, and then choose **Export Permission Sets**. This action runs XMLPort **9173 Export Permission Sets**.

6. If the old deployment uses data encryption, you have exported the encryption key file that it used for the data encryption.

   For more information, see How to: Export and Import Encryption Keys.

7. (Optional) Make a copy of the configuration file (web.config or navsettings.json) for all Business Central Web Server instances in the old deployment.

8. Business Central has been installed.

   As a minimum, you must install the following components:

   - Server
   - SQL Server Components
   - Business Central Web Server components
   - Dynamics NAV Development Environment(**DISCONTINUED AFTER:** Business Central Spring 2019).
   - AL Development Environment
   - (optionally) Business Central Server Administration tool

   For more information, see Installing Business Central Using Setup.

> **NOTE**
> If the old Dynamics NAV application uses Payment Services for Microsoft Dynamics ERP, be aware that this was discontinued in Microsoft Dynamics NAV 2017. This means that most of the objects that are associated with this feature will be deleted during the upgrade. Some objects you will have to manually delete.

Task 1: Create full SQL backup of old database

Create a full backup of the old database in the SQL Server. Alternatively, you can make a copy of the old database and perform the upgrade tasks on the copy.

For more information, see Create a Full Database Backup (SQL Server).

Task 2 Uninstall all extensions in old database

Open the Dynamics NAV Administration Shell or Business Central Administration Shell that matches to old database, and run these commands:

1. To get a list of the extensions that are installed, run this command:

   ```
   Get-NAVAppInfo -ServerInstance <ServerInstanceName> -Tenant default
   ```

   Replace `<ServerInstanceName>` with the name of the Dynamics NAV Server instance that the database connects to.

2. For each extension, run this command to uninstall it:

   ```
   Uninstall-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
   ```

   Replace `<ServerInstanceName>` with the name of the Dynamics NAV Server instance that the database connects to.

   Replace `<Name>` and `<N.N.N.N>` with the name and version of the Extension V1 as it appeared in the previous step.

   Alternately, to remove them all at once, you can run this command:

   ```
   Get-NAVAppInfo -ServerInstance <ServerInstanceName> -Tenant default | % { Uninstall-NAVApp -
   ServerInstance <ServerInstanceName> -Name $_.Name -Version $_.Version }
   ```

Task 3: Upload Business Central partner license to old database

By using the Dynamics NAV Development Environment that matches the old database, upload the Business Central license to the database.

For more information, see Uploading a License File for a Specific Database.

Task 4: Delete all objects except tables in old database

In the development environment version that matches the database, open the old database, open Object Designer, select all objects except tables, and then choose **Delete**.

You can also use the DeleteObjects command of the finsql.exe.

Task 5: Clear server instance and debugger breakpoint records in old database

Clear all records from the **dbo.Server Instance** and **dbo.Debugger Breakpoint** tables in the database in SQL Server.

1. Make sure that you stop the old server instance, and close any tools that connect to the database, such as the Dynamics NAV Administration Tool and development environment.
2. Using SQL Server Management Studio, open and clear the **dbo.Server Instance** and **dbo.Debugger Breakpoint** tables of the old database. For example, you can run the following SQL query:

```
DELETE FROM [<My NAV Database Name>].[dbo].[Server Instance]
DELETE FROM [<My NAV Database Name>].[dbo].[Debugger Breakpoint]
```

Task 6: Convert old database to Business Central

If the database is on Azure SQL Database, you must first add your user account to the **dbmanager** database role on master database. This membership is only required for converting the database, and can be removed afterwards.

To convert the old database to the Business Central format, run the new Dynamics NAV Development Environment for Business Central, open the old database, and follow the conversion instructions.

If you do not run the development environment as an administrator, you will get an error and the conversion will be stopped.

For more information about how to open a database, see Open a Database.

> **IMPORTANT**
> Do not run schema synchronization at this time. Choose to run it **later**.

Task 7: Import upgraded application objects to converted database

Using Dynamics NAV Development Environment for Business Central, import the application objects that you want in the database. This includes the application objects FOB file (from the application code upgrade) and the upgrade toolkit objects FOB file.

1. Import the application objects FOB file first, and then import the upgrade toolkit FOB file.

   For more information, see Importing Objects.

2. **IMPORTANT** When prompted about table synchronization, set the **Synchronize Schema** option to **Later**.

3. When you import the FOB file, if you experience metadata conflicts, the **Import Worksheet** windows appears.

   Review the Worksheet page. For more information, see Import Worksheet.

   Choose **Replace All**, and then **OK** to continue.

Task 8: Connect a Business Central Server instance to converted database

You use the Business Central Server Administration tool or Set-NAVServerConfiguration cmdlet in the Business Central Administration Shell to connect a Business Central Server instance to the converted database.

The service account that is used by the Business Central Server instance must be a member of the **db_owner** role in the Business Central database on SQL Server or Azure SQL Database.

For more information, see Connecting a Server Instance to a Database and Giving the account necessary database privileges in SQL Server.

> **IMPORTANT**
> When upgrading a large database, you should increase the **SQL Command Timeout** setting for the Business Central Server instance, to avoid timeouts during schema synchronization. The default setting is 30 minutes.

Task 9: Compile all objects in converted database

1. In the Dynamics NAV Development Environment, set it to use the server instance that connects to the database.

   For more information, see Changing the Server Instance.

2. Use the Dynamics NAV Development Environment or finsql.exe to compile all objects. This includes the imported application objects, data tables, and system tables.

   > **IMPORTANT**
   > Choose to run schema synchronization later. For example, in Object Designer, choose **Tools**, choose **Compile**, set the **Synchronize Schema** option to **Later**, and then choose **OK**. For more information, see Compiling Objects.

3. (Upgrade from Microsoft Dynamics NAV 2016 and earlier only) If you get errors on the following table objects, use the Object Designer to delete the objects because they are no longer used.

   - Table 470 Job Queue (replaced by the Task Scheduler)
   - Table 824 DO Payment Connection Details
   - Table 825 DO Payment Connection Setup
   - Table 827 DO Payment Credit Card
   - Table 828 DO Payment Credit Card Number
   - Table 829 DO Payment Trans. Log Entry
   - Table 1510 Notification Template

   > **IMPORTANT**
   > When you delete a table object, in the **Delete** confirmation dialog box that appears, set the **Synchronize Schema** option to **Force**.
   >
   > In this step, it's very important that you do not use the **Sync. Schema For All Tables** option from the **Tools** menu.

4. (Upgrade from Microsoft Dynamics NAV 2016 and earlier only) If the old database includes test runner codeunits, you will get errors on these codeunits that the OnBeforeTestRun and OnAfterTestRun trigger signatures are not valid. To fix these issues, you change the signature of the OnBeforeTestRun and OnAfterTestRun triggers to include the *TestPermission* parameter.

   For more information, see Resolving OnBeforeTestRun and OnAfterTestRun Trigger Errors When

Converting a Database.

The triggers for codeunit **130400 CAL Test Runner** and **130402 CAL Command Line Test Runner** will be updated for you during the data upgrade.

Task 10: (Upgrade from Dynamics NAV 2018 or Business Central Fall 2018 only) Increase the application version of converted database

You must increase the application version that is assigned to the database.

Use the Set-NAVApplication cmdlet of the Business Central Administration Shell to increase the application version number of the database from its current version.

To see the current version, use the following command:

```
Get-NAVApplication -ServerInstance <ServerInstanceName>
```

To increase the version by 1, run the following command:

```
Set-NAVApplication -ServerInstance <ServerInstanceName> -IncrementApplicationVersion
```

Or, to specify change to another version, run the following command:

```
Set-NAVApplication -ServerInstance <ServerInstanceName> -ApplicationVersion <N.N.N.N> -Force
```

For example, if the old version was `11.0.24279.0`, then you could change the version to `14.0.24279.0`.

Task 11: Run the schema synchronization on converted database

Synchronize the database schema with validation.

For example, run the Sync-NAVTenant cmdlet from the Business Central Administration Shell.

```
Sync-NAVTenant -ServerInstance <ServerInstanceName>
```

When completed, the tenant (database) should have the status **OperationalDataUpgradePending**. To verify this, run the following cmdlet:

```
Get-NAVTenant -ServerInstance <ServerInstanceName> -tenant default
```

For more information, see Synchronizing the Tenant Database and Application Database.

Task 12: Run data upgrade on converted database

A data upgrade runs the upgrade toolkit objects, such as upgrade codeunits and upgrade tables, to migrate business data from the old table structure to the new table structure. You can start the data upgrade from the Dynamics NAV Development Environment or Business Central Administration Shell.

1. Open the Business Central Administration Shell as an administrator, and then run Start-NavDataUpgrade cmdlet as follows:

```
Start-NavDataUpgrade -ServerInstance <ServerInstanceName> -FunctionExecutionMode Serial -
ContinueOnError
```

Replace `<ServerInstanceName>` with the name of the Business Central Server instance that is connected to the database.

> **NOTE**
> In the last phase of data upgrade, all companies will be initialized by running codeunit 2 Company Initialization. This is done automatically. If you want to skip company initialization, then use the `Start-NavDataUpgrade` with the `-SkipCompanyIntitialization` parameter.

To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

The data upgrade process runs `CheckPreconditions` and `Upgrade` functions in the upgrade codeunits. If any of the preconditions are not met or an upgrade function fails, you must correct the error and resume the data upgrade process. If CheckPreconditions and Upgrade functions are executed successfully, codeunit 2 is automatically run to initialize all companies in the database unless you set the `-SkipCompanyIntitialization` parameter.

2. Check for and resolve upgrade errors.

   Run the following command to get a list of any errors that have occurred:

   ```
   Get-NAVDataUpgrade -ServerInstance <ServerInstanceName> -ErrorOnly
   ```

   Resolve the errors before going to the next task.

Task 13: Upgrade Javascript-based control add-ins to new versions

The Business Central Server installation includes new versions of Microsoft-provided Javascript-based control add-ins, such as the Business Chart control add-in. If you application is using any of these add-ins, you must upgrade them to the new versions as follow:

1. Open the Business Central client.

   If the application uses the Business Chart control add-in, you will get an error about **High Charts**. Upgrading the Business Chart control add-in will clear this error.

2. Search for and open the **Control Add-ins** page.

   The page lists all the registered control add-ins.

3. Choose **Actions** > **Control Add-in Resource** > **Import**.

4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There is a sub-folder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\140\Service\Add-
   ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .

5. After you have imported all the new control add-in versions, restart Web Server instance.

Task 14: Publish and generate symbols for extensions

Complete this task if you are upgrading one of the following:

- Microsoft Dynamics NAV 2018 deployment that uses V2 extensions.
- Denmark (DK) version of Microsoft Dynamics NAV 2017 or earlier
- German (DE) version of Dynamics NAV or Business Central October 2018 (Cumulative Update 2 or earlier)

In this task, you will publish and generate symbols that are required for using V2 extensions. Symbols are the application programming interface between AL code and C/AL code. Symbols enable the ability to reference C/AL objects from AL objects. Symbols are provided as an extension package, and are published to the server instance, but not installed on tenants. There are three types of symbols: system, application, and test. System symbols contained references to the platform system objects. The application symbols contained references to the business application objects. The test symbols contained references to the test libraries used by Microsoft extensions.

1. To get list of the extensions currently published symbols, run the following command from the Business Central Administration Shell:

   ```
   Get-NAVAppinfo -ServerInstance <ServerInstanceName> -SymbolsOnly
   ```

2. Unpublish the existing system, test, and application symbols by using the Unpublish-NAVAPP cmdlet:

   ```
   Unpublish-NAVApp -ServerInstance <ServerInstanceName> -Name <name> -Version <n.n.n.n>
   ```

3. Publish the system.app and test.app symbol files.

   If you installed the **AL Development Environment**, you can find the symbol files where your installed the environment, which by default is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160. Otherwise, you can find the files in the **ModernDev** folder on the installation media.

   To publish the symbols, open the Business Central Administration Shell as an administrator, and run the following command for each of the symbol files:

   ```
   Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <SymbolFilePath> -PackageType SymbolsOnly
   ```

4. Generate the application symbol references by using the finsql.exe file as follows:

   a. Make sure that **Enable loading application symbol references at server startup** (EnableSymbolLoadingAtServerStartup) is set on the Business Central Server instance.

      For more information, see Configuring Business Central Server.

   b. Open a command prompt as an administrator, change to the directory where the `finsql.exe` file has been installed as part of Dynamics NAV Development Environment, and then run the following command:

      ```
      finsql.exe Command=generatesymbolreference, Database="<MyDatabaseName>", ServerName=
      <DatabaseServerName>\<DatabaseInstance>
      ```

      Replace values for the `Database` and `ServerName` settings to suit.

      If the application database contains test objects (ID 130000-139999), then make sure to exclude these objects when generating symbols. You can do this by using the `-Filter` parameter and

running the command twice:

```
finsql.exe command=generatesymbolreference, ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database="<MyDatabaseName>", filter="Object ID=1..129999"
```

```
finsql.exe command=generatesymbolreference, ServerName=<DatabaseServerName>\
<DatabaseInstance>, Database="<MyDatabaseName>", filter="Object ID=140000..1999999999"
```

> **NOTE**
> This command does not generate a file. It populates the **Object Metadata** table in the database.

c. When you run the command, the console returns to an empty command prompt, and does not display or provide any indication about the status of the run. However, the finsql.exe may still be running in the background. It can take several minutes for the run to complete, and the symbols will not be generated until such time. You can see whether the finsql.exe is still running by using Task Manager and looking on the **Details** tab for **finsql.exe**.

When the process ends, a file named **navcommandresult.txt** is saved to the Dynamics NAV Client connected to Business Central installation folder. If the command succeeded, the file will contain text like `[0] [06/12/17 14:36:17] The command completed successfully in '177' seconds.` If the command failed, another file named **naverrorlog.txt** will be generated. This file contains details about the error(s) that occurred.

For more information about generation symbols, see Running C/SIDE and AL Side-by-Side.

d. Restart the Business Central Server instance.

Task 15: Upgrade or repair V2 extensions

Complete this task if you are upgrading from a Microsoft Dynamics NAV 2018 deployment that uses V2 extensions. In this step, you will upgrade the V2 extensions that you previously uninstalled. The upgrade process will reinstall these extensions.

The Business Central installation media (DVD) includes several new versions of Microsoft extensions (that is, extensions that have **Microsoft** as the publisher). If your old deployment uses these extensions, you have to upgrade the current versions to the new versions.

1. To get list of the extensions currently published on the application, run the following command from the Business Central Administration Shell:

```
Get-NAVAppinfo -ServerInstance <ServerInstanceName>
```

2. Upgrade the Microsoft extensions that were published in the old deployment to new versions.

The new extension versions are found in the `\Extensions` folder of the installation media (DVD). Follow these steps for each extension by using the Business Central Administration Shell:

a. Publish the new extension version by running the Publish-NAVApp cmdlet:

```
Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <ExtensionFileName>
```

b. Synchronize the schema with the database by running the Sync-NAVApp cmdlet:

```
Sync-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

c. Upgrade the data of the extensions.

To run the data upgrade, run the Start-NAVAppDataUpgrade cmdlet:

```
Start-NAVAppDataUpgrade -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

Apart from upgrading the data, this command will install the new extension version.

For more information about publishing extensions, see Publish and Install an Extension.

3. Repair, synchronize, and install other currently published extension versions that you still want to use, but have not been upgraded to new versions.

For each extension, complete the following steps from the Business Central Administration Shell:

a. Compile the extension to make it work with the new platform by running the Repair-NAVApp cmdlet.

```
Repair-NAVApp -ServerInstance <ServerInstanceName> -Name <Extension Name> -Version <N.N.N.N>
```

b. Synchronize the schema with the database by running the Sync-NAVApp cmdlet:

```
Sync-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

c. Install the extension by running the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

4. (Optional) Unpublish unused extension versions by running the Unpublish-NAVApp:

```
Unpublish-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

Task 16: Publish and install local functionality extensions

Complete this task if you are upgrading one of the following:

- Denmark (DK) version of Microsoft Dynamics NAV 2017 or earlier
- German (DE) version of Dynamics NAV or Business Central October 2018 (Cumulative Update 2 or earlier)

With these language versions, some of the local functionality has been moved from the base application to extensions. These extensions will have to be published and installed to maintain the functionality. The extensions are avilable on the installation media (DVD).

If you are upgrading from a Denmark (DK) version, you must publish and install the following extensions to get the local functionality:

NAME	EXTENSION PACKAGE
OIOUBL	OIOUBL.app

NAME	EXTENSION PACKAGE
Payroll Data Import Definitions (DK)	ImportDKPayroll.app
Payment and Reconciliation Formats (DK)	FIK.app
Tax File Formats (DK)	VATReportsDK.app

If you are upgrading from a German (DE) version you must publish and install the following extensions to get the local functionality:

NAME	EXTENSION PACKAGE
ELSTER VAT Localization for Germany	Elster.app

Follow these steps for each extension by using the Business Central Administration Shell:

1. Publish the new extension version by running the Publish-NAVApp cmdlet:

   ```
   Publish-NAVApp -ServerInstance <ServerInstanceName> -Path <ExtensionFileName>
   ```

   The new extension versions are found in the `\Extensions` folder of the installation media (DVD).

2. Synchronize the schema with the database by running the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
   ```

3. Install the newly published local functionality extensions by running the Install-NAVApp cmdlet:

   ```
   Install-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
   ```

Task 17: Import permission sets and permissions

Import the permission sets and permissions XML files that you exported from the old database as follows:

- Upgrade from Dynamics NAV:

  1. Open table **2000000004 Permission Sets** in the client, and delete all permission sets except SUPER.

     > **NOTE**
     > You are only required to delete those permission sets that are also included in the permission sets XML file that you will import. Because if you try to import a permission set with the same name as one already in the database, you will get an error.

  2. Run XMLport **9171** and XMLport **9172** to import the permission sets and permission XML files.

     For more information, see How to: Export and Import Permission Sets and Permissions.

- Upgrade from an earlier Business Central version:

  1. In the client, search for and open the **Permission Sets** page.

2. Delete all user-defined permissions.
3. Choose **Import Permission Sets**, then select the permissions set file that you exported previously.

   This action runs the XMLPort **9174 Import Tenant Permissions**.

Task 18: (Optional) Import data encryption key

If you want to use data encryption as before, you must import the data encryption key file that was exported previously.

For more information, see Exporting and Importing Encryption Keys.

Task 19: Set the language of customer database

In the Dynamics NAV Development Environment, choose **Tools**, choose **Language**, and then select the language of the original customer database.

Task 20: (Optional) Update Web Server instance configuration file

If you have installed the Business Central Web Server, populate the navsettings.json file for the Business Central Web Server instance with the settings of the old web.config file or navsettings.json.

- If the old deployment used a web.config file, then you have to manually change the settings in the navsetting.json file that is used on the new Business Central Web Server instance.
- If you upgraded from Business Central October 2018, you can replace the navsettings.json file on the new Business Central Web Server instance with the old file. However, as of Business Central April 2019, the following settings are now configured under a root element called `ApplicationIdSettings` instead of the root element `NAVWebSettings`.
  - `AndroidPrivacy`
  - `AndroidSoftwareLicenseTerms`
  - `AndroidThirdPartyNotice`
  - `BaseHelpUrl`
  - `BaseSettingsSectionName`
  - `CommunityLink`
  - `FeedbackLink`
  - `IosPrivacy`
  - `IosSoftwareLicenseTerms`
  - `IosThirdPartyNotice`
  - `KeyboardShortcutsLink`
  - `PrivacyLink`
  - `LegalLink`
  - `SignInHelpLink`

  If the old navsettings.json file uses any of these settings, then you will have to move them from the `NAVWebSettings` element to the `ApplicationIdSettings` element.

For more information about the navsettings.json file, see Configuring Business Central Web Server Instances.

(Optional) Task 21: Delete upgrade objects

At this point, you have upgraded the database to Business Central. Now, you can delete the upgrade codeunits and upgrade table objects that you imported in task 9. This task is recommended but not required.

When you delete tables, on the **Delete** dialog box, set the **Synchronize Schema** option to **Force**.

See Also

Upgrading the Application Code
Upgrading to Business Central

Business Central Single-Tenant Full Upgrade Quick Reference

2/17/2021 • 2 minutes to read • Edit Online

This article provides an overview of the full upgrade process for Business Central in a single-tenant deployment. For more detailed steps, see Upgrading the Data: Single-Tenant Mode.

Prerequisite tasks on old database

STEP	MORE INFO	DONE
Upgrade application code.	See...	
Convert custom V1 extensions to V2 extensions.	See...	
Export permissions and permission sets. **Important:** Make sure your computer uses the same codepage as the data.	See...	
Export encryption keys from the old deployment.	See...	
Prepare for transitioning from codeunit 1. **Note:** Dynamics NAV upgrade only	See...	
Install Business Central components.	See...	

Prepare the old database for data upgrade

STEP	MORE INFO	DONE
Backup the database.	See...	
Uninstall all extensions.	See...	
Upload a Business Central partner license.	See...	
Delete all objects except tables. **Important** Do not synchronize schema at this point.	See...	
Clear server instance and debugger breakpoint tables.	See...	

Run the data upgrade

STEP	MORE INFO	DONE
Open Dynamics NAV Development Environment for Business Central as an administrator		
Connect to and convert the database.	See...	
Import upgraded application and upgrade toolkit objects (.fob files). **Important:** Select to synchronize **later**.	See...	
Connect a Business Central Server instance to the converted database.	See...	
Compile all objects. **Important:** Choose to synchronize schema **later**.	See...	
Increase the application version of the database, **Note:** Dynamics NAV 2018 upgrade only	See...	
Synchronize the database.	See...	
Run the data upgrade.	See...	
Update Javascipt control add-ins the data upgrade.	See...	

Publish, upgrade, and install extensions

STEP	MORE INFO	DONE
Publish system and test symbols, generate application symbols.	See...	
Publish, synchronize, and upgrade to new versions of Microsoft extensions from installation media.	''	
Repair, synchronize, and install old extension versions that were not upgraded in previous step.	''	
Run the data upgrade on the new extension versions.	''	
Repair other custom extensions to work on new platform.	''	

Post-upgrade tasks

STEP	MORE INFO	DONE
Import permissions and permission sets.	See...	
Import encryption keys	See...	
Upload the customer license.	See...	

Upgrading the Data to Business Central: Multitenant Deployment

2/17/2021 • 8 minutes to read • Edit Online

See print-friendly quick reference

This article describes the tasks required for upgrading data to the latest Business Central in a multitenant deployment.

About Data Upgrade

In this scenario, you already have an upgraded application that is mounted on a Business Central Server. You will then mount the old tenants on the server instance and perform the data upgrade.

You use data conversion tools provided with Business Central to convert the old data with the old version's table and field structure, so that it functions together with the new version's table and field structure. Mainly, only table objects and table data are modified during the data upgrade process. Other objects, such as pages, reports, codeunits, and XMLports are upgraded as part of the application code upgrade process.

The data upgrade process described in this article leads you through the database conversion (technical upgrade) and then the upgrade of the actual data, which is achieved by using the upgrade toolkit/upgrade codeunits.

> **IMPORTANT**
> Before you begin, read the article Important Information and Considerations for Before Upgrading. This article contains information about limitations and things that might require you to perform extra tasks before you upgrade, such as the use of extensions V1 and the deprecation of codeunit 1.

Prerequisites

Before you start the upgrade tasks, make sure you have the following prerequisites:

1. Your computer uses the same codepage as the data that will be upgraded.

   If you use conflicting codepages, some characters will not display in captions, and you might not be able to access the upgraded database. This is because Business Central must remove incorrect metadata characters to complete the data upgrade. In this case, after upgrade, you must open the database in the development environment on a computer with the relevant codepage and compile all objects. This adds the missing characters again.

2. (Upgrading from Dynamics NAV only) Custom V1 extensions used in Dynamics NAV have been converted to V2 extensions.

   For more information, see Converting Extensions V1 to Extensions V2.

3. Business Central has been installed with the upgraded application and upgrade toolkit.

   As a minimum, you must install the following components:

   - Business Central Server instance connected to the application database.
   - Dynamics NAV Development Environment for Business Central

- AL Development Environment

  This installs the required system and test symbols for V2 extensions.

- Business Central Server Administration tool (optional)
- Business Central Web Server components (not required for upgrade).

For more information about upgrading the application code, see Upgrading the Application Code.

4. Permission sets (except SUPER) and permissions have been exported from the old tenant database.

   - When upgrading from Dynamics NAV

     To export permission sets and permissions, you run running XMLPort 9171 and 9172.

     It is important that you exclude the SUPER permission set when running XMLPort 9171. You can do this by adding the filter `Role ID is <>SUPER`.

     For more information, see Exporting and Importing Permission Sets and Permissions.

   - When upgrading from an earlier version of Business Central

     In the client, search for and open the **Permission Sets** page, select the user-defined permission sets that you want to keep, and then choose **Export Permission Sets**.

5. If the old application uses data encryption, you have the encryption key file that it used for the data encryption.

   For more information, see Export and Import Encryption Keys.

> **NOTE**
> If the old Dynamics NAV application uses Payment Services for Microsoft Dynamics ERP, be aware that this was discontinued in Microsoft Dynamics NAV 2017. This means that most of the objects that are associated with this feature will be deleted during the upgrade. Some objects you will have to manually delete.

Prepare the tenant database for data upgrade

You perform these tasks on each tenant that you want to upgrade.

1. Backup the tenant database.

   Create a full backup of the old database in the SQL Server. Alternatively, you can make a copy of the old database and perform the upgrade tasks on the copy.

   For more information, see Create a Full Database Backup (SQL Server).

2. (Dynamics NAV upgrade only) Uninstall all V1 extensions.

   Make sure that all V1 extensions are uninstalled. Open the Dynamics NAV Administration Shell that matches to old database, and run these commands:

   a. To get a list of the V1 extensions that are installed, run this command:

   ```
   Get-NAVAppInfo -ServerInstance <OldServerInstanceName> -Tenant <TenantID>
   ```

   V1 extensions are indicated by `CSIDE` in the `Extension Type` column. 2. For each extension, run the Uninstall-NAVApp cmdlet to uninstall it:

```
Uninstall-NAVApp -ServerInstance <OldServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

3. Dismount the tenant.

   Before you upgrade the tenant, you must dismount it from the old server instance. To dismount the tenant, run the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <OldServerInstanceName> -Tenant <TenantID>
   ```

Run the data upgrade on the tenant

You perform these tasks on each tenant that you want to upgrade.

1. Mount the tenant.

   Mount the tenant on the new Business Central Server instance that connects to the newly application database. To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <ServerInstanceName> -DatabaseName <Database name> -DatabaseServer
   <server\instance> -Tenant <TenantID> -AllowAppDatabaseWrite
   ```

   IMPORTANT

   You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you will get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   NOTE

   For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

2. Synchronize the tenant.

   Synchronize the tenant database schema with validation by running the Sync-NAVTenant cmdlet from the Business Central Administration Shell.

   ```
   Sync-NAVTenant -ServerInstance <ServerInstanceName> -Tenant <TenantID>
   ```

   When completed, the tenant should have the status **OperationalDataUpgradePending** or, if there are published extensions with newer versions than on the tenant, **OperationalSyncPending**. To verify this, run the following cmdlet:

   ```
   Get-NAVTenant -ServerInstance <ServerInstanceName> -tenant default -ForceRefresh
   ```

3. If there are published extensions with newer versions than on the tenant, synchronize all published extensions with the tenant database.

   Synchronize the schema with the database by running the Sync-NAVApp cmdlet for each extension version:

```
Sync-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N> -Tenant <TenantID>
```

Or, to synchronize all published extensions using one command:

```
Get-NAVAppInfo -ServerInstance <ServerInstanceName> -Tenant <TenantID> | % { Sync-NAVApp -
ServerInstance <ServerInstanceName> -Name $_.Name -Version $_.Version }
```

When completed, the tenant should have the status **OperationalDataUpgradePending**.

4. Run the data upgrade.

   A data upgrade runs the upgrade toolkit objects, such as upgrade codeunits and upgrade tables, to migrate business data from the old table structure to the new table structure. It will also upgrade the published extensions.

   a. Open the Business Central Administration Shell as an administrator, and then run Start-NavDataUpgrade cmdlet as follows:

```
Start-NavDataUpgrade -ServerInstance ServerInstanceName> -FunctionExecutionMode Serial -
ContinueOnError
```

   Replace `<ServerInstanceName>` with the name of the Business Central Server instance that is connected to the database.

   > **NOTE**
   > In the last phase of data upgrade, all companies will be initialized by running codeunit 2 Company Initialization. This is done automatically. If you want to skip company initialization, then use the `Start-NavDataUpgrade` with the `-SkipCompanyIntitialization` parameter.

   To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

   The data upgrade process runs `CheckPreconditions` and `Upgrade` functions in the upgrade codeunits. If any of the preconditions are not met or an upgrade function fails, you must correct the error and resume the data upgrade process. If CheckPreconditions and Upgrade functions are executed successfully, codeunit 2 is automatically run to initialize all companies in the database unless you set the `-SkipCompanyIntitialization` parameter.

   b. Check for and resolve upgrade errors.

      Run the following command to get a list of any errors that have occurred:

```
Get-NAVDataUpgrade ServerInstanceName> -ErrorOnly
```

      Resolve the errors before going to the next step.

5. Install extensions on the tenant.

   Install the desired extensions on the tenant by running the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <ServerInstanceName> -Name <Name> -Version <N.N.N.N>
```

Post-upgrade tasks

1. Import permission sets and permissions

   Import the permission sets and permissions XML files that you exported from the old database as follows:

   - For upgrade from Dynamics NAV:

     a. Open table **2000000004 Permission Sets** in the client, and delete all permission sets except SUPER.

     > **NOTE**
     > You are only required to delete those permission sets that also included in the permission sets XML file that you will import. Because if you try to import a permission set with the same name as on already in the database, you will get an error.

     b. Run XMLport **9171** and XMLport **9172** to import the permission sets and permission XML files.

     For more information, see How to: Export and Import Permission Sets and Permissions.

   - For upgrade from an earlier Business Central version:

     a. In the client, search for and open the **Permission Sets** page.
     b. Delete all user-defined permissions.
     c. Choose **Import Permission Sets**, then select the permissions set file that you exported previously.

2. Import encryption keys.

   For more information, see Exporting and Importing Encryption Keys.

3. (Optional) Update Web Server instance configuration file

   If you have installed the Business Central Web Server, populate the navsettings.json file for the Business Central Web Server instance with the settings of the old web.config file or navsettings.json.

   - If the old deployment used a web.config file, then you have to manually change the settings in the navsetting.json file that is used on the new Business Central Web Server instance.
   - If you upgraded from Business Central October 2018, you can replace the navsettings.json file on the new Business Central Web Server instance with the old file. However, as of Business Central April 2019, the following settings are now configured under a root element called `ApplicationIdSettings` instead of the root element `NAVWebSettings`.
     - `AndroidPrivacy`
     - `AndroidSoftwareLicenseTerms`
     - `AndroidThirdPartyNotice`
     - `BaseHelpUrl`
     - `BaseSettingsSectionName`
     - `CommunityLink`
     - `FeedbackLink`
     - `IosPrivacy`
     - `IosSoftwareLicenseTerms`

- `IosThirdPartyNotice`
- `KeyboardShortcutsLink`
- `PrivacyLink`
- `LegalLink`
- `SignInHelpLink`

If the old navsettings.json file uses any of these settings, then you will have to move them from the `NAVWebSettings` element to the `ApplicationIdSettings` element.

For more information about the navsettings.json file, see Configuring Business Central Web Server Instances.

4. Upload the customer license.

   For more information, see Uploading the License File

See Also

Upgrading the Application Code
Upgrading to Business Central

Business Central Multitenant Full Upgrade Quick Reference

2/17/2021 • 2 minutes to read • Edit Online

This article provides an overview of the full upgrade process for Business Central in a multitenant deployment. For more detailed steps, see Upgrading the Data: Multitenant Mode.

Prerequisite tasks

STEP	MORE INFO	DONE
In the old deployment, convert custom V1 extensions to V2 extensions.	See...	
Export permissions and permission sets from the old deployment. **Important:** Make sure computer uses the same codepage as the data.	See...	
Export encryption keys from the old deployment.	See...	
Prepare for transitioning from codeunit 1.	See...	
Install Business Central components.	See...	

Upgrade the application and prepare it for data upgrade

STEP	MORE INFO	DONE
Upgrade the application code.	See...	
Mount the upgraded application on the Business Central Server instance.	See...	
Import upgrade toolkit (.fob)	See...	
Publish system and test symbols from the installation media, and generate application symbols.	See...	
Publish the new Microsoft extension versions from the installation media.	See...	
Upload a Business Central partner license.	See...	

Prepare the tenant database for data upgrade

STEP	MORE INFO	DONE
Backup the tenant database.	See...	
Uninstall all V1 extensions.	See...	
Dismount the tenant from the old server instance.	See...	

Run the data upgrade on the tenant

STEP	MORE INFO	DONE
Mount the tenant on the Business Central Server instance. **Important:** Use the `-AllowAppDatabaseWrite` parameter.	See...	
Synchronize the tenant.	See...	
Synchronize all extensions.	See..	
Run the data upgrade. **Important:** If there are V2 extensions, you must use the `-FunctionExecutionMode Serial` parameter.	See...	
Install the new V2 extensions that were not installed in the old tenant.	See...	

Post-upgrade tasks

STEP	MORE INFO	DONE
Import permissions and permission sets.	See...	
Import encryption keys	See...	
Upload the customer license.	See...	

Installing a Business Central Spring 2019 Cumulative Update

2/17/2021 • 7 minutes to read • Edit Online

This article describes how to install a cumulative update for Business Central on-premises. A cumulative update is a set of files that include hotfixes and regulatory features released for Business Central.

Download the cumulative update package

The first thing to do is to download the Cumulative Update package that matches your Business Central deployment.

1. Go to the relevant list of available updates for your on-premises version of Business Central. Then, choose the Cumulative Update you want.

   For a list of supported versions of Business Central on-premises, see the See Also section.

2. From the cumulative update page, under the **Resolution** section, select the link for downloading the update and follow the instructions.

3. On the computer where you downloaded cumulative update, extract files from all .zip files.

The cumulative update includes files that are separated into the following folders:

- APPLICATION folder

  Used for updating your application with the new or modified application objects that form the cumulative update.

- DVD

  Contains the full Business Central product, including the Business Central installation program (setup.exe) and tools for upgrading to the platform.

When this step is done, you can continue to update your Business Central deployment to the new platform and application.

Update the platform

The following components are part of the Business Central platform:

- Business Central Web Server
- Business Central Server
- SQL Server components
- Dynamics NAV Client connected to Business Central

To upgrade to the latest platform, the database must be converted by using the Dynamics NAV Development Environment.

1. (Single-tenant deployment only) Uninstall all extensions.

   If the Business Central Server is configured as a single-tenant instance, you must uninstall all extensions from tenant. For example, using the Uninstall-NAVApp cmdlet from the Business Central Administration Shell, you can run the following command:

```
Get-NAVAppInfo -ServerInstance <ServerInstanceName> -Tenant default | % { Uninstall-NAVApp -
ServerInstance <ServerInstanceName> -Name $_.Name -Version $_.Version }
```

2. Install Business Central components of the cumulative update.

   There are two ways to install components. You can use the setup.exe file from the installation media or manually patch files to the existing installation:

   - Using setup.exe

     a. Uninstall the current version of Business Central.

        You can uninstall it by running setup.exe found in the DVD folder of the download or from Control Panel.

     b. From the DVD folder, run setup.exe to install Business Central.

        As a minimum, install the following components: Server, Web Server Components, SQL Server Components, and the Dynamics NAV Development Environment. For more information, see Installing Business Central Using Setup.

   - Manually patch the components

     With this way, you copy component files from installation media (DVD) to the current installation folders as described in the following table:

COMPONENT	COPY FROM DVD	TO CURRENT INSTALLATION FOLDER
Server	DVD\ServiceTier\program files\Microsoft Dynamics NAV\140	C:\Program Files\Microsoft Business Central<140>\Service
WebClient	DVD\WebClient\Microsoft Dynamics NAV\140\Web Client	C:\Program Files\Microsoft Dynamics 365 Business Central<140>\Web Client
Windows Client and Development Environment	DVD\RoleTailoredClient\program files\Microsoft Dynamics NAV\140\RoleTailored Client	C:\Program Files (x86)\Microsoft Dynamics 365 Business Central<140>\RoleTailored Client
Microsoft Office Outlook Integration	• DVD\Outlook\program files\Microsoft Dynamics NAV\140\OutlookAddin • Microsoft.Dynamics.NAV.OLSync.NAVSyncAddIn.dll	• C:\Program Files(x64)\Microsoft Office\Office <version> • C:\Program Files (x64)\Microsoft Office\Office\xx-XX

TIP

Before you install, save a copy of the configuration files for the Business Central Server (CustomSettings.config), the Business Central Web Server (navsettings.json), and Dynamics NAV Client connected to Business Central (ClientUserSettings.config). You can then refer to settings the copies to configure the components again.

3. Start the new development environment as an administrator

4. Open the application database.

   For more information, see Open a Database.

5. If prompted, follow instructions to convert it to the new platform.

6. (Multitenant deployment only) Mount an old tenant to the Business Central Server instance.

   you'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

7. Synchronize the database (tenant).

   For more information about syncing, see Synchronizing the Tenant Database and Application Database.

Update the application

The APPLICATION folder includes the following files:

- AccumulatedChangeLog...txt
- Changelog...txt
- CUObjects...fob
- Objects..Objects...txt

Using Dynamics NAV Development Environment for Business Central, complete one of the following tasks.

> **NOTE**
> If a license update is required for a regulatory feature, customers can download an updated license from CustomerSource (see How to Download a Microsoft Dynamics 365 Business Central License from CustomerSource), and partners can download their customers' updated license from VOICE (see How to Download a Microsoft Dynamics 365 Business Central Customer License from VOICE).

Update an unmodified application to the Business Central cumulative update objects

1. Open the database in the development environment.
2. Import the CUObjects...fob into the application database.

   For more information, see Importing Objects.

3. Replace the existing objects in the database with the cumulative update objects.

Update a modified application to the Business Central cumulative update objects

1. Import the CUObjects...fob into the application database.

   For more information, see Importing Objects.

2. When prompted, open the **Import Worksheet**, and review the information.

   a. Replace objects in the database that have NOT been modified.

   b. If a table in the cumulative update has a new field and the same table in your database has been modified, use the **Merge: Existing<-New** or **Merge: New<-Existing** option in the **Import Worksheet** to import the new fields.

   c. For other objects that have been modified, use the CUObjects...txt file to compare and merge the cumulative update objects with the objects in your database.

      For more information about the worksheet, see Import Worksheet.

> **NOTE**
> If you don't want to use the Worksheet, you can use the Changelog...txt file to manually apply the changes to the objects in your database.

Updating from Business Central October 2018 DE Cumulative Update 2 or earlier

Starting with Cumulative Update 03, the ELSTER local functionality is contained in an extension instead of the base application. If you want the latest updates to the ELSTER functionality, you must publish and install the ELSTER extension. Publishing the extension is described later. To prepare for this step, delete the following objects from the application database: page 11016, page 11017, page 11019, and report 11016. These objects will be replaced by the ELSTER extension.

Make sure you synchronize the tenant after you delete the objects.

Set the application version of database to the application version of the cumulative update

Increase the application version of the application database in order to do a data upgrade. We recommend that you change to the application version to that of the cumulative update. You can get this version from the cumulative update release page. For more information, see Version numbers in Business Central.

To set the application version, use the Set-NAVApplication cmdlet of the Business Central Administration Shell to set the application version as follows:

```
Set-NAVApplication -ServerInstance <ServerInstanceName> -ApplicationVersion
Major.CU.ApplicationBuild.Revision -Force
```

Synchronize and upgrade tenants

1. (Multitenant deployment only) Mount the tenant.

   For more information, see Mount or Dismount a Tenant .

2. Synchronize the tenant.

3. Run a data upgrade.

   Run Start-NavDataUpgrade cmdlet of the Business Central Administration Shell:

   ```
   Start-NavDataUpgrade -ServerInstance <ServerInstanceName> -Tenant <TenantID>
   ```

   Replace `<TenantID>` with the tenant ID of the database. If you don't have a multitenant server instance, use `default` or omit this parameter.

Publish and install/upgrade extensions

Complete this task if you're updating:

- A Business Central October 2018 DE version earlier than Cumulative Update 3 and you want the latest updates to the Elster functionality. Use this procedure to install the following extension:

NAME	EXTENSION PACKAGE
ELSTER VAT Localization for Germany	Elster.app

- A Business Central application that uses extensions.

  The Business Central installation media (DVD) includes several new versions of Microsoft extensions (that

is, extensions that have **Microsoft** as the publisher). If your old deployment uses these extensions, you have to upgrade the current versions to the new versions.

Also, repair other extensions used in the old deployment that you still want to use so the extensions work on the new platform.

The general steps for this task are listed below. For detailed steps, see Publishing, Upgrading, and Installing Extensions During Upgrade .

1. Publish new system, test, and application symbols.

   Unpublish the old versions first.

2. Publish the new extension versions from the DVD.

   The extensions include new versions of Microsoft extensions that you already have used on your application, and the ELSTER app for the DE version. The new extension versions are found in the `\Extensions` folder of the installation media (DVD).

3. Synchronize the tenant database with the schema changes of the extensions.

4. Upgrade the data associated with the Microsoft extensions. This step isn't required the first time you publish the ELSTER extension.

5. Install the newly published extensions on tenants.

6. Repair, synchronize, and install any custom extensions (third-party) that are currently published. Do this step for old extensions that you still want to use.

   This step ensures that the extensions work on the new platform and application versions.

See Also

Dynamics 365 Business Central On-Premises October'18 Updates
Dynamics 365 Business Central On-Premises April'19 Updates
Upgrading the Application Code
Upgrading to Business Central
Synchronizing the Tenant Database and Application Database
Version numbers in Business Central
Publish and Install an Extension

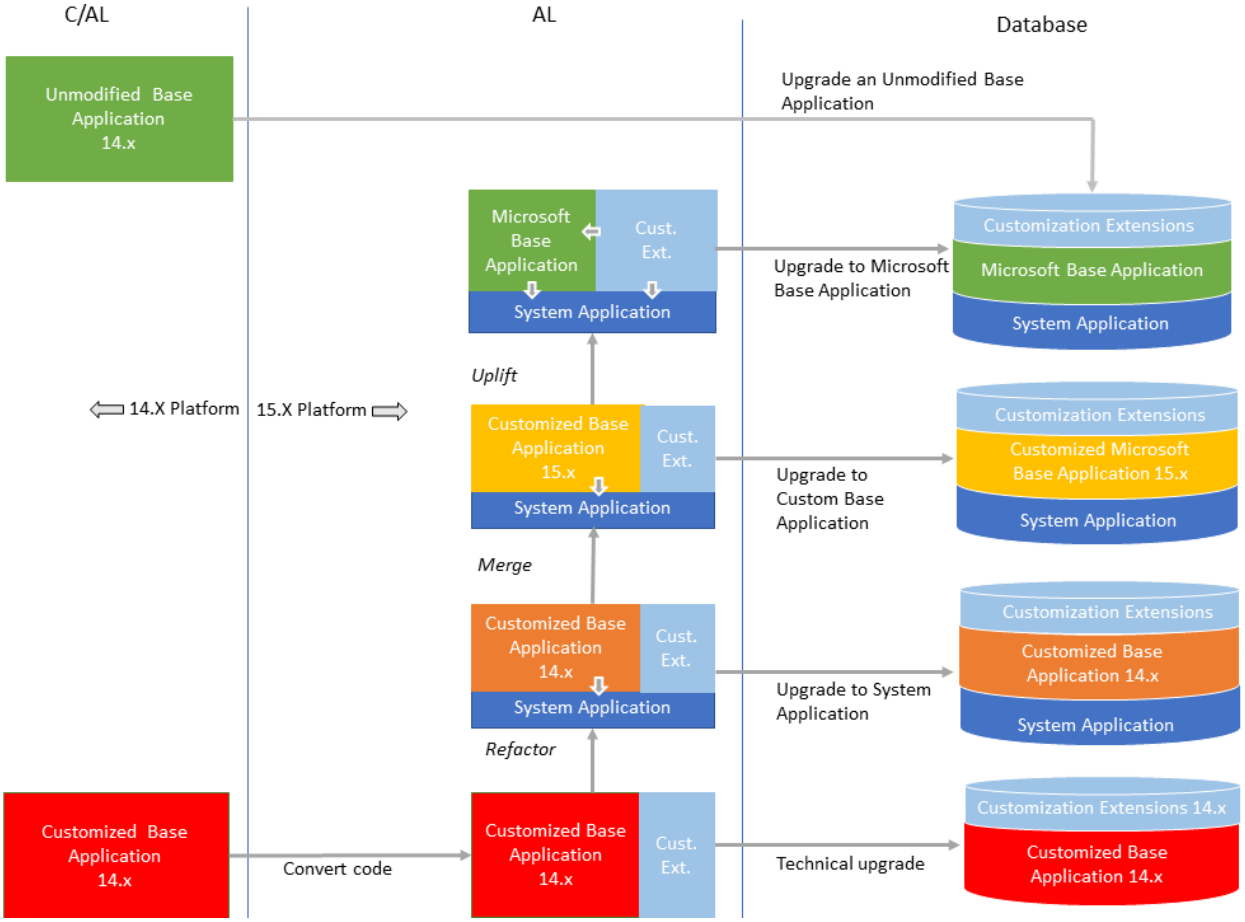
Upgrading to Dynamics 365 Business Central 2019 Release Wave 2

2/17/2021 • 3 minutes to read • Edit Online

Business Central 2019 release wave 2 (version 15) marks the release where C/AL has been completely replaced by AL. The 2019 release wave 2 is the first version that does not include the classic development environment (also known as C/SIDE). From an application perspective, this means that Business Central 2019 Wave 2 is completely extension-based. The Business Central base application is now delivered as an AL extension instead of C/AL. Additionally, application functionality that is not related to the business logic has been moved into separate modules that are combined into an extension known as the System Application. This change will influence how you perform the upgrade compared to earlier releases.

Upgrade paths

When upgrading your Business Central Spring 2019 (version 14) solution to version 15, the goal is to move towards a full uptake of the Business Central base and system applications, as they are, and migrating code customizations to add-on extensions. There are different upgrade paths that you follow to get to this state, as illustrated in the following figure.



As mentioned, the recommended upgrade path for a customized solution is to uptake the version 15 Microsoft Base Application and System Application, and move all code customizations to extensions. However, we realize that the complexity of some solutions will make this path very difficult. If this path is not currently realistic for your solution, then we recommend as a minimum to upgrade to a version 15 customized base application.

IMPORTANT

Be aware that for each path, once the database has been synchronized and data upgraded, it becomes more difficult to bring your solution to the next path. This will require significant manual work until tooling is available in a future release.

For details about each path, see the following articles:

- Upgrade of an Unmodified Application
- Technical Upgrade of Customized Application

The following articles are currently being formalized and will be available as soon as possible

- Upgrade to the System Application
- Upgrade to a Customized Version 15 Base Application
- Upgrade to the Microsoft Base Application

NOTE

Upgrading to Dynamics 365 Business Central 2019 Release Wave 2 requires that you first upgrade to Dynamics 365 Business Central Spring 2019 (version 14).

New and changed application features

There a several new and changed platform application features available in Business Central April 2019 release wave2 for users, administrators, and developers. For an overview of these features, see Overview of Dynamics 365 Business Central 2019 release wave 2.

To take advantage of these all these features, you will have to perform an application code upgrade, not just a technical (platform) upgrade.

Components

Base Application

The base application contains the objects (such as table, pages, codeunits, and reports) that define the business logic and functionality of the solution. In version 14 and earlier, the base application also contained system objects that were not specifically related to the business logic. In version 15, the standard business objects are now included in the Microsoft Base Application extension, and the system objects have been moved to the System Application extension.

System Application

In version 15.0, application functionality that is not related to the business logic has been moved into separate modules that are combined into an extension known as the System Application. For an introduction to the System Application, see For more information, see Overview of the System Application.

Symbols

Symbols are the application programming interface between AL code and C/AL code. Symbols enable the ability to reference C/AL objects from AL objects. Symbols are provided as an extension package, and are published to the server instance similar to application extensions, but not installed on tenants.

In version 14.0, with the base application being C/AL, there are three types of symbols: system, application, and test. System symbols contained references to the platform system objects. The application symbols contained references to the business application objects. The test symbols contained references to the test libraries used by Microsoft extensions.

In version 15, with the move to AL, the only symbols required are the system symbols, which are still provided on the version 15.0 installation media (DVD).

Customization extensions

Customization extensions are AL extensions that add functionality to the base application or system application. These extensions can be Microsoft (1st party) or 3rd party extensions. 3rd party extensions are extensions that your organization provides or extensions that are provided by others, such as from ISVs or from App Source.

See Also

Upgrade of an Unmodified Application
Technical Upgrade of Customized Application
Dynamics 365 Business Central Upgrade Compatibility Matrix

Dynamics 365 Business Central Upgrade Compatibility Matrix

2/17/2021 • 2 minutes to read • Edit Online

You can upgrade Business Central Spring 2019 (version 14) directly to 2019 release wave 2 (version 15) or to 2020 release wave 1 (version 16). And, of course, you can upgrade 2019 release wave 2 directly to 2020 release wave 1 (version 16). For an overview of the supported paths, see Supported Upgrade Paths.

However, minor updates are regularly made available for each major release, like 14.1 or 15.2. When upgrading, it's important to target an update version that's compatible with your current version.

Before choosing the target version

Before you choose the target version for your upgrade, read the Some Known Issues article. This article will describe issues in Business Central versions that affect upgrade.

Version 14 compatibility

The following table lists the Business Central 14 versions and the minimum 15, 16, and 17 version that's compatible for upgrade.

VERSION 14	VERSION 15	VERSION 16	VERSION 17
14.0 (GA and cumulative update 01)	15.0	16.0	17.0
14.3 (cumulative update 02)	15.0	16.0	17.0
14.4 (cumulative update 03)	15.0	16.0	17.0
14.5 (cumulative update 04)	15.0	16.0	17.0
14.6 (cumulative update 05)	15.1	16.0	17.0
14.7 (cumulative update 06)	15.2	16.0	17.0
14.8 (cumulative update 07)	15.3	16.0	17.0
14.9 (cumulative update 08)	15.3	16.0	17.0
14.10 (cumulative update 09)	15.4	16.0	17.0
14.11 (cumulative update 10)	15.5	16.0	17.0
14.12 (cumulative update 11)	15.6	16.1	17.0

VERSION 14	VERSION 15	VERSION 16	VERSION 17
14.13 (cumulative update 12)	15.7	16.2	17.0
14.14 (cumulative update 13)	15.8	16.3	17.0
14.15 (cumulative update 14)	15.9	16.4	17.0
14.16 (cumulative update 15)	15.10	16.5	17.0
14.17 (cumulative update 16)	15.11	16.6	17.0
14.18 (cumulative update 17)	15.12	16.7	17.1
14.19 (cumulative update 18)	15.13	16.8	17.2
14.20 (cumulative update 19)	15.14	16.9	17.3
14.21 (cumulative update 20)	15.15	16.10	17.4
14.22 (cumulative update 21)[1]	15.16	16.11	17.5

For example, you can upgrade version 14.0 to any 15 or 16 version. You can only upgrade version 14.11 to version 15.5 (or later) or version 16.0 (or later).

To see the available updates for Business Central 2019 Release Wave 2, see Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises.

[1]The compatible upgrade versions for this version aren't available yet. If you are currently operating on this version, you'll have to wait until the next round of updates before you upgrade.

Version 15 compatibility

The following table lists the Business Central 15 versions and the minimum 16 and 17 version that's compatible for upgrade.

VERSION 15	VERSION 16	VERSION 17
15.0 to 15.4	16.0	17.0
15.5	16.1	17.0
15.6	16.2	17.0
15.7	16.3	17.0

VERSION 15	VERSION 16	VERSION 17
15.8	16.4	17.0
15.9	16.5	17.0
15.10	16.6	17.0
15.11	16.7	17.1
15.12	16.8	17.2
15.13	16.9	17.3
15.14	16.10	17.4
15.15[1]	16.11	17.5

To see the available updates for Business Central 2020 Release Wave 1, see Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 1 on-premises.

Version 16 compatibility

The following table lists the Business Central 16 versions and the minimum 17 version that's compatible for upgrade.

VERSION 16	VERSION 17
16.0 to 16.5	17.0
16.6	17.1
16.7	17.2
16.8	17.3
16.9	17.4
16.10[1]	17.5

To see the available updates for Business Central 2020 Release Wave 2, see Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

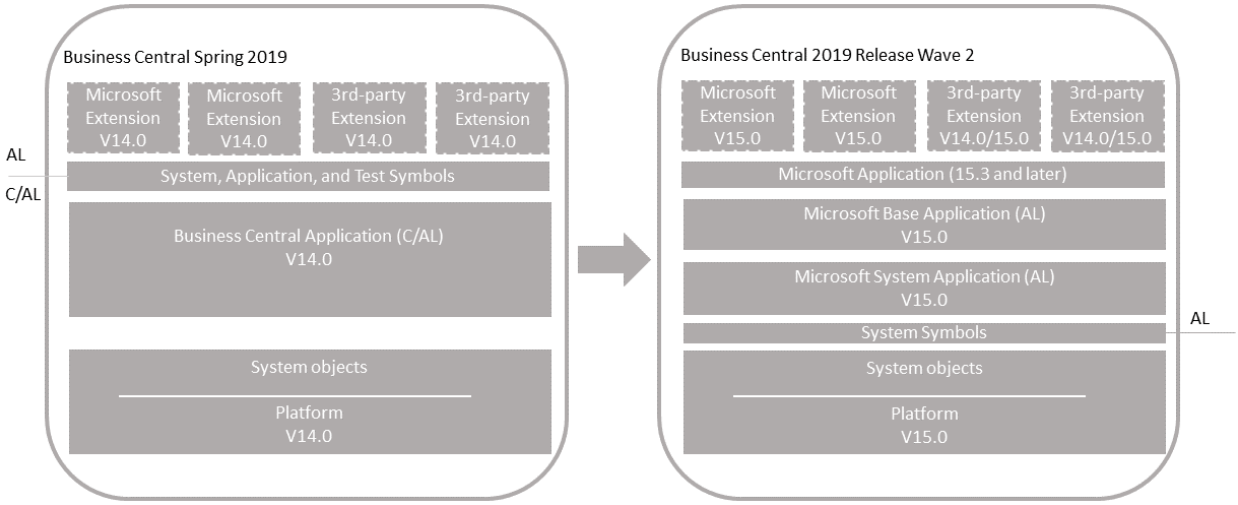
See Also

Upgrading to Business Central

Upgrading an Unmodified Application to Dynamics 365 Business Central 2019 Release Wave 2

2/17/2021 • 13 minutes to read • Edit Online

Use this scenario if you have a Business Central Spring 2019 (referred to as version 14) application that doesn't include any code customization. Your solution might include Microsoft (first party) extensions and customization extensions (3rd-party). With this upgrade, you'll replace the C/AL base application with the new Business Central 2019 release wave 2 base application extension. The result will be a fully upgraded Business Central 2019 release wave 2 (referred to as version 15) application and platform.



Single-tenant and multitenant deployments

The process for upgrading the similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application code and business data are in the same database. In a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisite

1. Upgrade to Business Central Spring 2019 (version 14).

   There are several updates for version 14.

   - If you're upgrading from Business Central Fall 2018 (version 13) or Dynamics NAV, we recommend you upgrade to the latest update for version 14 that has a compatible update for version 15.
   - If your solution is already on version 14, then you don't have to upgrade to the latest version 15 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   To download the latest update, go to Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

   For information about how to do the upgrade, see Upgrading to Dynamics 365 Business Central On-Premises.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

For more information, see Managing Encryption and Encryption Keys.

Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install Business Central (version 15)

1. Download the latest available update for Business Central 2019 version 15 that is compatible with your version 14.

   To download the latest update, go to Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises.

   The guidelines in this article assume that you're running the latest available update.

2. Before you install version 15, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 15 tools.

3. Install Business Central version 15 components.

   If you don't uninstall version 14, then you must either specify different port numbers for components (like the Business Central Server instance and web services) during installation, or you must stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 2: Prepare the version 14 databases for upgrade

1. Make backup of the databases.

2. Start Business Central Administration Shell for version 14 as an administrator.

3. Uninstall all extensions from the old tenants.

   In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>
      ```

      For a single-tenant deployment, set the `<tenant ID>` to default.

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Name <extensions name> -Tenant
      <tenant ID> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the

published System Application.

For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

```
Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-
NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
$_.Version -Force}
```

4. Unpublish all extensions from the application server instance.

   To unpublish an extension, use the Unpublish-NAVApp cmdlet:

   ```
   Unpublish-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension
   version>
   ```

   Together with the Get-NAVAppInfo cmdlet, you can unpublish all extensions by using a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
   server instance> -Name $_.Name -Version $_.Version }
   ```

5. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

6. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

7. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 3: Convert the version 14 database to the version 15 platform

This task runs a technical upgrade on the application database to convert it from the version 14 platform to the version 15 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 15 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (14-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 4: Connect and configure the version 15 server for app migration

When you installed version 15 in **Task 1**, a version 15 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 14 to version 15.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Configure the server instance for migrate extensions to the use the new base application and system application extensions.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"63ca2fa4-4f03-4f2b-a480-172fef340d3f",
   "name":"System Application", "publisher": "Microsoft"},{"appId":"437dbf0e-84ff-417a-965d-
   ed2bb9650972", "name":"Base Application", "publisher": "Microsoft"}]'
   ```

   This setting serves the following purposes:

   - When you run the data upgrade on a tenant, the server will run the data upgrade for the base and system application extensions. The base and system applications will be automatically installed on the tenant also.
   - Lets you republish extensions that haven't been built on version 15. The extensions typically include the third-party extensions that were used in your version 14. When you publish the extensions, the extension manifests are automatically modified with a dependency on the base and system applications.

3. Disable task Scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

4. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 5: Increase the application version

This task is optional, but it's recommended. You can choose to skip it for now and do it later. In this task, you'll

increase the application_version that's stored in ndodbproperty table of the application database. The application version isn't changed automatically. The application version serves two purposes:

- Enables running the Start-NAVDataUpgrade cmdlet later in Task 8 of this article. The application version is compared with the tenant's version. If the application version is greater, a data upgrade can be run. If you skip this task, you'll have to use the `-SkipAppVersionCheck` switch with Start-NAVDataUpgrade cmdlet in Task 8.
- The application version is shown in the client on the **Help and Support** page. This task ensures that page displays the latest application version.

The version has the format `major.minor.build.revision`, such as, '14.3.14824.1'. As a minimum, you increase the revision by 1. However, we recommend setting the value to application build number for the version 15 update. You can get this number from the Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises page.

To change the application version, run the Set-NAVApplication cmldet:

```
Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application version> -
Force
```

For example:

```
Set-NAVApplication -ServerInstance BC150 -ApplicationVersion 15.1.38071.0 -Force
```

Later, when you synchronize and upgrade the tenant(s), the new application version will be updated in the tenant database (ndotenantproperty table).

Task 6: Publish the symbols and extensions

In this task, you'll publish the platform symbols and extensions. As minimum, you publish the new base application and system application extensions from the installation media (DVD). You also publish new versions of any Microsoft extensions and third-party extensions that were used on your old deployment.

Publishing an extension adds the extension to the application database that is mounted on the server instance. Once published, it's available for installing on tenants. This task updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 15 that you started in the previous task.

1. Publish version 15 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\150\AL Development Environment.

   ```
   Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
   SymbolsOnly
   ```

   What are symbols?

2. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source**

folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
Application.app>"
```

What is the System Application?

3. Publish the Business Central base application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the (Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
   Application.app>"
   ```

4. Publish the Microsoft_Application extension (update to 15.3 and later only)

   The Microsoft_Application extension is a new extension introduced in 15.3. For more information about this extension, see The Microsoft_Application.app File.

   a. Get the Microsoft_Application extension package and source code.

      With update 15.4 or later, the extension package and source code are on the installation media. Complete this step only if you're installing update 15.3. The Microsoft_Application extension isn't on the 15.3 installation media.

      With update 15.3, you must download extension package and source code from the Microsoft Download Center. To download, go to https://download.microsoft.com/download/9/9/1/991764bc-5f99-4be7-957a-f132ac0633ef/Application.zip.

   b. Publish the Microsoft_Application.app package.

      ```
      Publish-NAVApp -ServerInstance <server instance name> -Path "<folder
      path>\Microsoft_Application.app"
      ```

5. Publish the new versions of Microsoft extensions.

   In this step, you publish new versions of Microsoft extensions that were used on your version 14 deployment. You find the extensions in the **Applications** folder of the installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft extension>"
   ```

   For example:

   ```
   Publish-NAVApp -ServerInstance BC150 -Path
   "C:\W1DVD\Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app"
   ```

6. Publish 3rd-party extensions.

   Publish 3rd-party extensions that were used on your version 14 solution. If you have new versions of these extensions, built on the Business Central version 15, then publish the new versions. Otherwise, republish the same versions that were previously published in the old deployment.

```
Publish-NAVApp -ServerInstance BC150 -Path "<path to extension>"
```

Task 7: Restart the server instance

Restart the Business Central Server to free up resources for completing the upgrade.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

This step is important, otherwise you might experience issues when you run the data upgrade.

Task 8: Synchronize the tenant

In this task, you'll synchronize the tenant's database schema with any schema changes in the application database and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 15 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
   <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

   IMPORTANT

   You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   NOTE

   For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

   At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the **System Application** extension.

   Use the Sync-NAVApp cmdlet:

```
Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
Version <extension version>
```

Replace `<extension version>` with the exact version of the published System Application. To get the version, you can use the Get-NAVAppInfo cmdlet.

4. Synchronize the tenant with the Business Central Base Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

5. Synchronize the tenant with the Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
   ```

6. Synchronize the tenant with Microsoft and 3rd-party extensions.

   For each extension, run the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance BC150 -Tenant default -Name "<extension name>" -Version <extension
   version>
   ```

> **TIP**
> When you synchronize an extension, the extension takes ownership of any tables that it includes. In SQL Server, you'll notice that the table names will be suffixed with the extension ID. For example, Base Application tables will have `437dbf0e-84ff-417a-965d-ed2bb9650972` in the name. In addition, the systemId column is added to application tables that are not already part of an extension.

Task 9: Upgrade the data

In this task, you run a data upgrade on tables to handle data changes made by platform and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. Upgrade the data to the platform, system application, and base application.

   a. To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

      ```
      Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -
      FunctionExecutionMode Serial [-SkipAppVersionCheck]
      ```

      You only need to use the -SkipAppVersionCheck if you didn't increase the application version in Task 5.

   b. To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

   This step will automatically install the base application and system application on the tenant.

2. Upgrade the new versions of Microsoft extensions and third-party extensions.

Complete this task to upgrade any Microsoft extensions used in the old deployment to new versions on the installation media. The new versions are in the **Application** folder of the DVD. There's a folder for each extension. The extension package (.app file) is in the **Source** folder.

To upgrade an extension, run Start-NAVAppDataUpgrade cmdlet:

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Name "<extension name>" -Version
<extension version>
```

This step will also automatically install the new extension version on the tenant.

3. (Multitenant only) Repeat steps 1 through 3 for each tenant.

Task 10: Install 3rd-party extensions

Complete this task to install third-party extensions for which a new version wasn't published. For each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
```

At this point, the upgrade is complete, and you can open the client.

Post-upgrade tasks

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

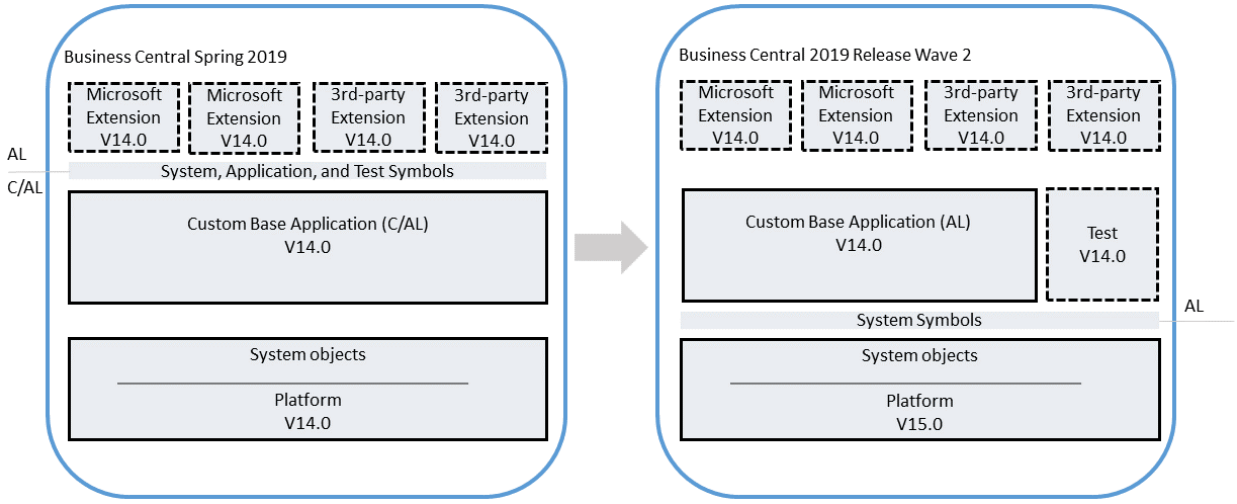
See Also

Upgrading the Data
Upgrading to Business Central

Technical Upgrade to Dynamics 365 Business Central 2019 Wave 2

2/17/2021 • 10 minutes to read • Edit Online

Use this process when you have a code customized Business Central application (version 14) that you want to upgrade to the Business Central 2019 release wave 2 platform (version 15). This process won't upgrade the application to the latest version. You'll convert the entire application from C/AL to an AL base application extension.



Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Upgrade to Business Central Spring 2019 (version 14).

   There are several updates for version 14. When upgrading from Business Central Fall 2018 (version 13) or Dynamics NAV, upgrade to the latest version 14 update that has a compatible version 15 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   If your solution is already on version 14, then you don't have to upgrade to the latest version 15 update.

   To download the latest update, go to Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

   For information, see Upgrading to Dynamics 365 Business Central On-Premises.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install Dynamics 365 Business Central 2019 release wave 2 (version 15.0)

1. Before you install version 15, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 15.0 tools.

2. Install all components of Business Central 2019 release wave 2 (version 15).

   If you didn't uninstall version 14.0, then you must either specify different port numbers for components during installation or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

3. Copy the version 14 **CodeViewer** add-in to the version 15.0 server installation.

   a. Find the **CodeViewer** folder in the **Add-ins** folder of the version 14 RoleTailored client installation. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client\Add-ins.

   b. Copy the folder to the **Add-ins** folder of the version 15 server installation. By default, the folder path is C:\Program Files\Microsoft Dynamics 365 Business Central\150\Service\Add-ins. Replace the existing folder and files, if any.

   CodeViewer is no longer used in version 15. But it's required because of references that exist in the converted application. If you omit this step, you might get compilation errors later.

Task 2: Convert your application from C/AL to AL

Convert your solution from C/AL code to AL code. For more information, see Code Conversion from C/AL to AL.

Task 3: Prepare the application database for technical upgrade

1. Make backup of the database.

2. Make sure that you have the extension packages for all published extensions.

   You'll need these packages later to publish and install the extensions again.

3. Uninstall all extensions from the old tenants.

   Run the Business Central Administration Shell for version 14.0 as an administrator. To uninstall an extension, you use the Uninstall-NAVApp cmdlet. For example, together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -Tenant <tenant ID> | % { Uninstall-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version -Tenant <tenant ID>}
   ```

   If you have a single tenant deployment, you can omit the `-Tenant` parameter and value.

4. Unpublish all extensions from the application server instance.

   To unpublish extensions, use the Unpublish-NAVAPP cmdlet. Together with the Get-NAVAppInfo cmdlet,

you can uninstall all extensions from the tenant using a single command:

```
Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
server instance> -Name $_.Name -Version $_.Version }
```

5. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet. You can unpublish all symbols by using the Get-NAVAppInfo cmdlet with the `-SymbolsOnly` switch as follows:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

6. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <BC14 server instance> -Tenant <tenant ID>
   ```

7. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <BC14 server instance>
   ```

Task 4: Convert the version 14.0 application database to the version 15.0 platform

This task runs a technical upgrade on the application database. A technical upgrade converts the database from the version 14.0 platform to the version 15.0 platform. This conversion updates the system tables of the database to the new schema (data structure). It also provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 15.0 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server>\<database instance> -
   DatabaseName "<BC14 database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (14-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 5: Connect and configure the version 15 server instance

When you installed version 15 in **Task 1**, a version 15 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are

only required for version 14 to version 15.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <BC15 server instance> -KeyName DatabaseName -KeyValue "
   <BC14 database name>"
   ```

   In a single tenant deployment, this command mounts the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <BC15 server instance> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <BC15 server instance>
   ```

Task 6: Publish system symbols, base application, and test library extensions

In this task, you'll publish extensions to the version 15.0 server instance. Publishing adds the extension to the application database that is mounted on the server instance. The extension is then available for installing on tenants later. It updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 15.0 that you started in the previous task.

1. Publish the system symbols extension for version 15.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default installation path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\150\AL Development Environment.

   ```
   Publish-NAVApp -ServerInstance <BC15 server instance> -Path "<path to the System.app file>" -
   PackageType SymbolsOnly
   ```

2. Publish the custom base application extension that you created in **Task 2**.

   ```
   Publish-NAVApp -ServerInstance <BC15 server instance> -Path "<path to the base application extension
   package file>"
   ```

3. Publish the test library extension if you created one in **Task 2**.

   ```
   Publish-NAVApp -ServerInstance <BC15 server instance> -Path "<path to the test library extension
   package file>"
   ```

Task 7: Synchronize tenant and synchronize/install base application and test library extensions

This task updates the tenant database schema with schema changes in system objects and application objects. You'll synchronize the tenant to the custom base application and test library extension (if any). When you synchronize the tenant, extensions take ownership of the tables in the SQL Database.

If you have a multitenant deployment, run these steps for each tenant (replacing `<tenant ID>` with the appropriate tenant ID).

1. (Multitenant only) Mount the tenant to the version 15 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <BC15 server instance> -DatabaseName "<BC14 database name>" -
   DatabaseServer <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

   IMPORTANT

   You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you will get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   NOTE

   For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

2. Synchronize the tenant to the system objects.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <BC15 server instance> -Tenant <tenant ID> -Mode Sync
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   At this stage, the tenant state is **Operational**.

3. Synchronize the tenant to the base application extension (Base Application).

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <BC15 server instance> -Name "Base Application" -Version <extension
   version> -tenant <tenant ID>
   ```

   With this step, the base app takes ownership of the database tables. When completed, in SQL Server, the table names will be suffixed with the base app extension ID.

4. Install custom base application extension on the tenant.

   To install the extension, you use the Install-NAVApp cmdlet.

```
Install-NAVApp -ServerInstance <BC15 server instance> -Name "Base Application" -Version <extension
version>
```

At this point, the base application is upgraded to the version 15 platform and is operational. You can open the application in the client.

5. Synchronize and install the test library extension.

   This step is like what you did for the custom base application in steps 3 and 4.

Task 8: Configure the version 15 server instance for migrating extensions

In this task, you configure the version 15 server so that the Microsoft and third-party extensions that were installed in the version 14 deployment can be reinstalled. You'll configure the `DestinationAppsForMigration` parameter of the server instance with information about the custom base application and test library. Specifically, you need the appId, name, and publisher assigned to these extensions. With the `DestinationAppsForMigration` parameter set, when you publish the Microsoft and third-party extensions, the server instance will automatically modify the manifest of the extensions to include the dependency on the base application and test library extension, allowing them to be published.

1. Get the appId, name, and publisher of the custom base application and test library.

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance>
   ```

2. Set the `DestinationAppsForMigration` parameter for the server instance configuration to include the information about the custom base application and test library (if used). For example:

   ```
   Set-NAVServerConfiguration -ServerInstance <BC15 server instance> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"437dbf0e-84ff-417a-965d-ed2bb9650972",
   "name":"Base Application", "publisher": "Microsoft"},{"appId":"<test library extension app ID>",
   "name":"<test library extension name>", "publisher": "<test library publisher>"}]'
   ```

3. Restart the server instance.

Task 9: Publish, synchronize, and install extensions on the tenants

Now, you can install the Microsoft and 3rd-party extensions that were installed on the tenant before the upgrade.

1. Publish the old extension versions.

   ```
   Publish-NAVApp -ServerInstance <BC15 server instance> -Path "<path to extension package file>"
   ```

   You only need to do this step once.

2. Synchronize the extension.

   ```
   Sync-NAVApp -ServerInstance <BC15 server instance> -Name "<extension name>" -Version <extension
   version> -tenant <tenant ID>
   ```

3. Install the extension:

```
Install-NAVApp -ServerInstance <BC15 server instance> -Name "<extension name>" -Version <extension version> -tenant <tenant ID>
```

4. Repeat steps 2 and 3 for each extension and on each tenant.

Now, your application is fully upgraded to the version 15 platform.

Task 10: Post-upgrade

1. Upgrade Javascript-based control add-ins to new versions.

   The Business Central Server installation includes new versions of the following Microsoft-provided Javascript-based control add-ins that must be upgraded.

   - Microsoft.Dynamics.Nav.Client.BusinessChart
   - Microsoft.Dynamics.Nav.Client.FlowIntegration
   - Microsoft.Dynamics.Nav.Client.OAuthIntegration
   - Microsoft.Dynamics.Nav.Client.PageReady
   - Microsoft.Dynamics.Nav.Client.PowerBIManagement
   - Microsoft.Dynamics.Nav.Client.RoleCenterSelector
   - Microsoft.Dynamics.Nav.Client.SocialListening
   - Microsoft.Dynamics.Nav.Client.SatisficationSurvey
   - Microsoft.Dynamics.Nav.Client.TimelineVisualization
   - Microsoft.Dynamics.Nav.Client.VideoPlayer
   - Microsoft.Dynamics.Nav.Client.WebPageViewer
   - Microsoft.Dynamics.Nav.Client.WelcomeWizard

   To upgrade the control add-ons, do the following steps:

   a. Open the Business Central client.

   b. Search for and open the **Control Add-ins** page.

   c. Choose **Actions** > **Control Add-in Resource** > **Import**.

   d. Locate and select the .zip file for the control add-in and choose **Open**.

      The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

      ```
      C:\Program Files\Microsoft Dynamics 365 Business Central\150\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
      ```

      .

   e. After you've imported all the new control add-in versions, restart Business Central Server instance.

2. Enable task scheduler on the server instance.

3. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.

4. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

See Also

Upgrading the Data
Upgrading to Business Central
Business Central 14.X to 15.X compatibility matrix

Installing a Business Central 2019 Release Wave 2 Update

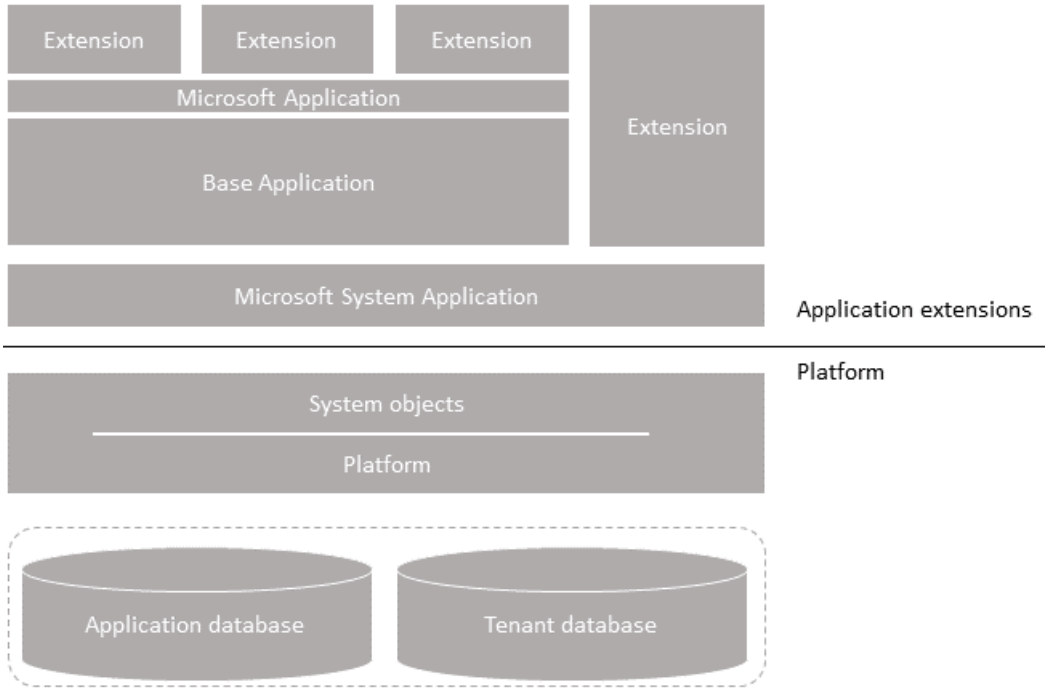
2/17/2021 • 17 minutes to read • Edit Online

This article describes how to install an update for Business Central on-premises. An update is a set of files that includes all hotfixes and regulatory features that have been released for Business Central.

You can choose to update only the platform or both the platform and application code. The installation guidelines are separated into PLATFORM tasks and APPLICATION tasks.

Overview

The following figure provides a high-level representation of a Business Central solution and the components that are involved in the installation of an update.



The databases store the application metadata and business data. If you have a single-tenant deployment, this data is stored in a single database. A multitenant deployment stores the application metadata in the application database and the business data in one or more tenant databases.

Application stack

The application includes AL extensions that define the objects and code that makes up the business logic. For example, objects include tables, report, pages, codeunits, and more. Each extension is compiled and delivered as an .app file, which is published to the Business Central Server instance.

- Base application

  As a minimum, the solution always includes the Base Application. The Base Application contains the objects (such as table, pages, codeunits, and reports) that define the business logic and functionality of the solution. The Base Application can be either the Microsoft Base Application or a customized Base Application. The Microsoft Base Application is the standard application that is on the installation media (DVD). A customized Base Application is an application that includes customized code.

The Microsoft System Application extension includes functionality that isn't directly related the business logic. For more information, see Overview of the System Application. When using the Microsoft Base Application, your solution uses the System Application. With a custom Base Application, your solution may or may not use the System Application. If it doesn't, you can skip any steps in this article related to the System Application.

- Extensions

  Extensions add functionality and features to the Base Application or System Application. Extensions can be either Microsoft extensions or third-party extensions. Microsoft extensions are available on the DVD. Third-party extensions are extensions developed by your organization or by another organization, like an ISV.

Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Platform versus application update

A platform update doesn't change the application. It involves converting your databases to the new platform and recompiling the existing extensions to ensure that they're compatible with the new platform.

An application update involves:

- Publishing new versions of extensions that include the latest application modifications
- Synchronizing the databases with any schema change introduced by the new extensions
- Updating affected data.

The installation media (DVD) includes new versions of Microsoft's Base Application, System Application, and extensions. The DVD also includes the AL source code for the Microsoft Base Application. This code useful if you have a custom base application. You can use the code to compare and merge updates into your application. You'll only have to recompile third-party extensions that you don't have a new version to publish.

PREPARATION

Download update package

The first thing to do is to download the update package that matches your Business Central solution.

1. Go to the list of available updates for your on-premises version of Business Central. Then, choose the update that you want.
2. From the update page, under the **Resolution** section, select the link for downloading the update, and follow the instructions.
3. On the computer where you downloaded the update .zip file, extract the all the files to a selected location.

   When extracted, the update includes the DVD folder. This folder contains the full Business Central product. For example, the folder includes the Business Central installation program (setup.exe), tools for upgrading to the platform, and the Microsoft extensions.

When this step is completed, you can continue to update your Business Central solution to the new platform and application.

Prepare existing databases

1. Back up your databases.
2. Run the Business Central Administration Shell as an administrator.
3. (Single-tenant only) Uninstall all extensions from the all tenants.

   In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name>
      ```

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extensions
      name> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the installed extension. For single-tenant deployment, set `<tenant ID>` to `default` or omit the `-Tenant` parameter.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID> | % { Uninstall-
      NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
      $_.Version -Force}
      ```

4. Unpublish the existing system symbols.

   To unpublish the system symbols, use the Unpublish-NAVApp cmdlet as follows:

   ```
   Unpublish-NAVApp -ServerInstance <server instance> -Name System -version <version>
   ```

   What are symbols?.

5. (Multitenant only) Dismount the tenants from the application database.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID>
   ```

Install Business Central components

From the installation media (DVD), run setup.exe to uninstall the current Business Central components and install the Business Central components included in the update.

1. Stop the Business Central Server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance>
   ```

2. Run setup.exe to uninstall your current version of Business Central.

3. Run setup.exe again to install components of the update.

   a. Follow setup pages until you get to the **Microsoft Dynamics 365 Business Central Setup** page.

   b. Select **Advanced installation options** > **Choose an installation option** > **Custom**.

   c. On the **Design customization the installation** page, select the following components as a minimum:

      - AL Development Environment (optional but recommended)
      - Server
      - Web Server Components.

   d. Select **Next**.

   e. On the **Specify parameters** page, set the fields as needed.

      > **IMPORTANT**
      > Clear the **SQL Database** field so that it is blank. At this time, do not set this to the database that you want to update; otherwise, the installation of the Business Central Server will fail. You will connect the database to the Business Central Server later after it is converted to the new platform.

   f. Select **Apply** to complete the installation.

For more information, see Installing Business Central Using Setup.

PLATFORM UPDATE

Follow the next few tasks to convert your database to the new platform of the update. A multitenant deployment includes the application and tenant databases. The conversion updates the system tables of the database to the new schema (data structure) and provides the latest platform features and performance enhancements.

Also, to ensure that the existing published extensions work on the new platform, you'll recompile the extensions.

Convert existing database to new platform

1. Run the Business Central Administration Shell as an administrator.

2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the database conversion to the new platform.

   In a multitenant deployment, run this cmdlet against the application database.

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>" [-Force]
   ```

   For example, in a single tenant deployment:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (15-
0)"
```

In a multitenant deployment, run this cmdlet against the application database and use the `-Force` parameter. For example:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer .\BCDEMO -DatabaseName "BC15 Application" -
Force
```

When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (15-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

NOTE

Depending on the update that you're installing, you might get a message similar to the following:

```
Invoke-NAVApplicationDatabaseConversion : A technical upgrade of database <database name> on
server '.\<database instance>' cannot be run, because the database's application version 'NNNNNN'
is greater than or equal to the platform version 'NNNNNN'
```

This is not an error, and you can continue installing the update. This message is recorded as a warning in the event log as well. This message indicates that the application database is already compatible with the new platform, which happens when the update does not make any schema changes to the system tables.

Connect server instance to database

1. (Multitenant only) Enable the server instance as a multitenant instance:

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance> -KeyName Multitenant -KeyValue true
   ```

2. Connect the server instance to connect to the database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a multitenant deployment, the database is the application database. For more information, see Connecting a Server Instance to a Database.

3. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Publish the new system symbols

Use the Publish-NAVApp cmdlet to publish the new symbols extension package. This package is called **System.app**. If you've installed the **AL Development Environment**, you find the file in the installation folder. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\150\AL

Development Environment. Or, it's also on the installation media (DVD) in the ModernDev\program files\Microsoft Dynamics NAV\150\AL Development Environment folder.

```
Publish-NAVApp -ServerInstance <server instance> -Path "<path to the System.app file>" -PackageType SymbolsOnly
```

Recompile published extensions

Compile all published extensions against the new platform.

> **NOTE**
> If you plan on updating the application you can skip this step for extensions for which you have new versions built on the new platform. For example, this includes Microsoft extensions that are on the DVD.

1. To compile an extension, use the Repair-NAVApp cmdlet, For example:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   To compile all published extensions at once, you can use this command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Synchronize tenant

1. (Multitenant only) Mount the tenant to the new Business Central Server instance.

   You'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

2. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID> -Mode Sync
   ```

   For a single-tenant deployment, you can either set the `<tenant ID>` to `default` or omit the `-Tenant <tenant ID>` parameter. For more information about syncing, see Synchronizing the Tenant Database and Application Database.

Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\150\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

> **NOTE**
> At this point, if you want to update the application, you can skip the next step and proceed APPLICATION.

(Single-tenant only) Reinstall extensions

In this task, you reinstall the same extensions that were installed on the tenant before. If you're planning on updating the application, then you only do this step on third-party extensions.

To install an extension, you use the Install-NAVApp cmdlet.

1. If your solution uses the System Application, install this first.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.
2. Install the Base Application.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Base Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.
3. Install the Application extension as needed.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Application extension.
4. Install other extensions, including Microsoft and third-party extensions.

   ```
   Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
   ```

At this point, your solution has been updated to the latest platform.

APPLICATION UPDATE

Follow the next tasks to update the application code to the new features and hotfixes. The tasks include publishing new versions of the System Application, Base Application, and add-on extensions.

You publish the System Application extension only if it was used in old solution. Add-on extensions include Microsoft and third- party extensions that were used in the old solution.

> **NOTE**
> If a license update is required for a regulatory feature, customers can download an updated license from CustomerSource (see How to Download a Microsoft Dynamics 365 Business Central License from CustomerSource), and partners can download their customers' updated license from VOICE (see How to Download a Microsoft Dynamics 365 Business Central Customer License from VOICE).

Upgrade System Application

Follow these steps if your existing solution uses the Microsoft System Application. Otherwise, you can skip this procedure.

1. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
   Application.app>"
   ```

2. Synchronize the tenant(s) with the **System Application** extension (Microsoft_System Application):

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

   > **TIP**
   > To get a list of all published extensions, along with their names and versions, use the Get-NAVAppInfo cmdlet.

3. Run the data upgrade on the System Application.

   To run the data upgrade, use the Start-NavAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System
   Application" -Version <extension version>
   ```

   Upgrading data updates the data in the tables of the tenant database to the schema changes made to tables of the System Application.

Upgrade Base Application

Microsoft Base Application

Follow these steps if your existing solution uses the Microsoft Base Application.

1. Publish the Business Central Base Application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
   Application.app>"
   ```

2. Synchronize the tenant with the Business Central Base Application extension (Microsoft_BaseApp):

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

   With this step, the base app takes ownership of the database tables. When completed, in SQL Server, the table names will be suffixed with the base app extension ID. This process can take several minutes.

3. Run the data upgrade on the Base Application.

   To run the data upgrade, use the Start-NavAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base
   Application" -Version <extension version>
   ```

   Upgrading data updates the data in the tables of the tenant database to the schema changes made to tables of the Base Application.

Upgrade custom Base Application

With a custom Base Application, you may want the new application features and hotfixes in the Microsoft Base Application. If so, you'll have to merge the modifications made in the Microsoft Base Application into your custom Base Application. Then, create a new version of your custom Base Application.

The source code for the new Microsoft Base Application version is in the **Base Application.Source.zip** file. This file is on the installation media (DVD), in the **Applications\BaseApp\Source** folder. You can compare this source code with the source code of the previous Microsoft Base Application and your custom application source. Then merge the code into a new custom application version.

After you've created the new version of your custom application, you publish it to the application server instance. Then, you synchronize and run the data upgrade on the tenants.

Upgrade Microsoft extensions

If your old solution used Microsoft extensions, then you upgrade these extensions to the new versions that are available on the Business Central installation media (DVD). The new versions are in the **Applications** folder, which contains a subfolder for each extension. The extension package (.app file) that you need for publishing the extension is in the **Source** folder, for example, **Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app**.

The general steps for this task are listed below. For detailed steps, see Publishing, Upgrading, and Installing Extensions During Upgrade.

Publish and install Microsoft_Application extension (update to 15.3 and later only)

The Microsoft_Application extension is a new extension introduced in 15.3. For more information about this extension, see The Microsoft_Application.app File.

1. Get the Microsoft_Application extension package and source code.

   With update 15.4 or later, the extension package and source code are on the installation media. Complete this step only if you're installing update 15.3. The Microsoft_Application extension isn't on the 15.3 installation media.

   With update 15.3, you must download extension package and source code from the Microsoft Download Center. To download, go to https://download.microsoft.com/download/9/9/1/991764bc-5f99-4be7-957a-f132ac0633ef/Application.zip.

2. If your base application has an ID other the Microsoft ID (437dbf0e-84ff-417a-965d-ed2bb9650972), you have to modify the Microsoft_Application extension. Skip this step if you're using the Microsoft Base Application or a custom base application that has the Microsoft ID.

   The Microsoft_Application extension must include a dependency reference to your custom base application instead of Microsoft's. To make this change, use the extension's source code that you downloaded. For example:

   a. Extract the source code from the Application.source.zip file.

   b. Create a Visual Studio Code project that includes the extracted files. You can give the project any valid name.

   c. In the extension's app.json file, change the base application dependency to match your base application.

      > **IMPORTANT**
      > Do not change the name of the extension. It must have the name **Application**; otherwise you won't be able to publish the other Microsoft extensions.

   d. Build the extension package.

3. Publish the Microsoft_Application.app package.

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<folder path>\Microsoft_Application.app"
```

4. Synchronize the tenant database with the schema changes of the Microsoft_Application extension.

```
Sync-NavApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension name> -Tenant
<tenant ID>
```

5. Install the Microsoft_Application extension on your tenants.

```
Install-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name Application
```

Publish and install Microsoft extensions

Complete these steps for each Microsoft extension that you want to upgrade.

1. Publish the new extension versions from the installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path <path to extension package file>
```

2. Synchronize the tenant database with the schema changes of the extensions.

```
Sync-NavApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension name> -
Version <extension version>
```

3. Upgrade the data associated with the Microsoft extensions. This step will automatically install the new version on the tenant

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension
name> -Version <extension version>
```

Upgrade Third-Party extensions

If the old solution used third-party extensions, and you still want to use them, they must be compiled to work on the new platform. Then, you reinstall the extensions on tenants.

As an alternative, if you have the source for these extensions, you can build and compile a new version of the extension in the AL development environment. Then, you upgrade to the new version as described in the previous task.

Post Upgrade - Change application version

(Optional) This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version.

On the **Help and Support** page in the client, you'll see an application version, such as 15.1.2345.6. For an explanation of the number, see Version numbers in Business Central. This version number isn't updated automatically when you install an update. If you want the application version to reflect the version of the update or your own version, you change it manually as described here.

We recommend setting the value to application build number for the version 15 update. You get the number from the Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises.

1. Run the Set-NAVApplication cmldet:

```
Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
version> -Force
```

   For example:

```
Set-NAVApplication -ServerInstance BC150 -ApplicationVersion 15.3.38071.0 -Force
```

2. Run the Sync-NAVTenant cmdlet to synchronize the tenant with the application database.

```
Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Run the Start-NavDataUpgrade cmdlet to change the version number:

```
Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID>
```

See Also

Dynamics 365 Business Central On-Premises Release Wave 2 Updates
Upgrading to Dynamics 365 Business Central 2019 Release Wave 2
Synchronizing the Tenant Database and Application Database
Version numbers in Business Central
Publish and Install an Extension
Getting Started in AL

Upgrading to Dynamics 365 Business Central 2020 Release Wave 1

2/17/2021 • 2 minutes to read • Edit Online

Business Central 2020 release wave 1 (version 16) is the second major release that is fully AL-based. Business Central 2019 release wave 2 (version 15) marked the release in which C/AL was replaced by AL. The classic development environment, known as C/SIDE, has been deprecated. From an application perspective, Business Central is now extension-based only. The Business Central base application is delivered as an AL extension. Also, application functionality that isn't related to the business logic has been moved into separate modules. These modules are combined into an extension known as the System Application. This change will influence how you do the upgrade compared to earlier releases.

Upgrade path

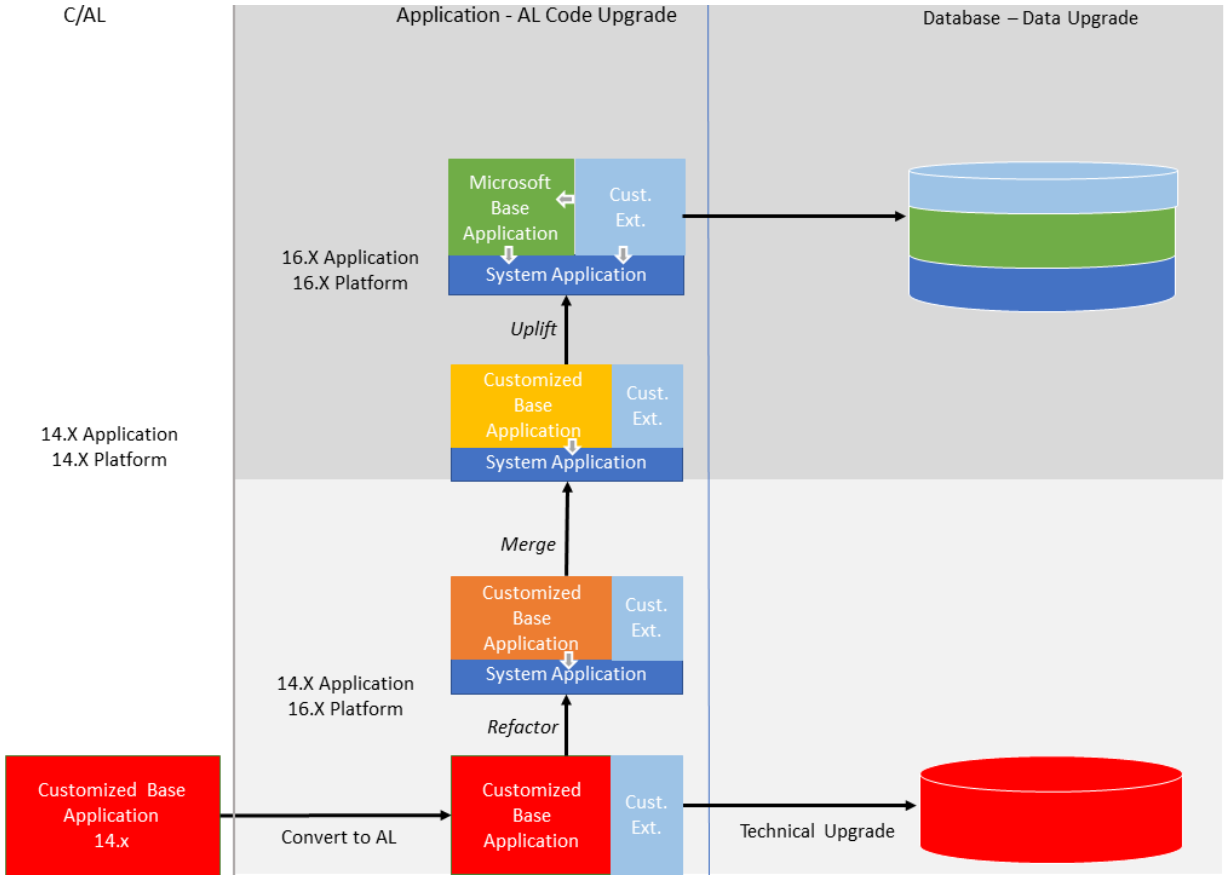
Depending on your current version, a direct upgrade to version 16 isn't always possible. You may have to first upgrade to an intermediate version. The following table describes the upgrade paths for supported versions:

SOURCE VERSION	PATH	COMMENT
• Microsoft Dynamics NAV 2015 (version 8) • Microsoft Dynamics NAV 2016 (version 9) • Microsoft Dynamics NAV 2017 (version 10) • Microsoft Dynamics NAV 2018 (version 11) • Business Central October 2018 (version 13)	1. Business Central Spring 2019 (version 14) 2. Business Central 2020 Release Wave 2 (version 16)	This path requires you convert your application from C/AL to AL.
• Business Central Spring 2019 (version 14) • Business Central 2019 Release Wave 2 (version 15)	Direct to version 16	

Your current version doesn't have to be the latest update for the version. However, for intermediate versions, use to the latest available update.

Upgrade overview

When upgrading your Business Central Spring 2019 (version 14) solution to version 16, the goal is to move towards a full uptake of the Business Central base and system applications, as they are, and migrating code customizations to addon extensions. There are different upgrade levels that you follow to get to this state, as illustrated in the following figure. We recommend that you refactor to the system application as a minimum.



Technical Upgrade

Although it's recommend to refactor to the system application as a minimum, you can do a technical upgrade only. A technical upgrade changes the database so that it works on the latest Business Central platform. The conversion updates the system tables of the old database to the new schema (data structure). It provides you with the latest platform features and performance enhancements.

When upgrading from version 14, part of the technical upgrade process includes converting your customized base application from C/AL to AL.

New and changed features

There are several new and changed platform and application features available in Business Central 2020 release wave 1. These changes affect users, administrators, and developers. For an overview of these features, see Overview of Dynamics 365 Business Central 2020 release wave 1.

To take advantage of these features, you'll have to do an application code upgrade, not just a technical (platform) upgrade.

Deprecated features

Before you upgrade, review the following articles to get an overview of features deprecated in this release:

- Deprecated Tables
- Deprecated Features in W1

  From this article, use the links in the table of content to view deprecated features specific to local versions

Migrating from on-premises to online

For information about migrating an on-premises solution to online, see Migrate to Business Central Online from Business Central On-premises.

See Also

Upgrading to Business Central
Upgrading Extensions
Dynamics 365 Business Central Upgrade Compatibility Matrix

Dynamics 365 Business Central Upgrade Compatibility Matrix

2/17/2021 • 2 minutes to read • Edit Online

You can upgrade Business Central Spring 2019 (version 14) directly to 2019 release wave 2 (version 15) or to 2020 release wave 1 (version 16). And, of course, you can upgrade 2019 release wave 2 directly to 2020 release wave 1 (version 16). For an overview of the supported paths, see Supported Upgrade Paths.

However, minor updates are regularly made available for each major release, like 14.1 or 15.2. When upgrading, it's important to target an update version that's compatible with your current version.

Before choosing the target version

Before you choose the target version for your upgrade, read the Some Known Issues article. This article will describe issues in Business Central versions that affect upgrade.

Version 14 compatibility

The following table lists the Business Central 14 versions and the minimum 15, 16, and 17 version that's compatible for upgrade.

VERSION 14	VERSION 15	VERSION 16	VERSION 17
14.0 (GA and cumulative update 01)	15.0	16.0	17.0
14.3 (cumulative update 02)	15.0	16.0	17.0
14.4 (cumulative update 03)	15.0	16.0	17.0
14.5 (cumulative update 04)	15.0	16.0	17.0
14.6 (cumulative update 05)	15.1	16.0	17.0
14.7 (cumulative update 06)	15.2	16.0	17.0
14.8 (cumulative update 07)	15.3	16.0	17.0
14.9 (cumulative update 08)	15.3	16.0	17.0
14.10 (cumulative update 09)	15.4	16.0	17.0
14.11 (cumulative update 10)	15.5	16.0	17.0
14.12 (cumulative update 11)	15.6	16.1	17.0

VERSION 14	VERSION 15	VERSION 16	VERSION 17
14.13 (cumulative update 12)	15.7	16.2	17.0
14.14 (cumulative update 13)	15.8	16.3	17.0
14.15 (cumulative update 14)	15.9	16.4	17.0
14.16 (cumulative update 15)	15.10	16.5	17.0
14.17 (cumulative update 16)	15.11	16.6	17.0
14.18 (cumulative update 17)	15.12	16.7	17.1
14.19 (cumulative update 18)	15.13	16.8	17.2
14.20 (cumulative update 19)	15.14	16.9	17.3
14.21 (cumulative update 20)	15.15	16.10	17.4
14.22 (cumulative update 21)[1]	15.16	16.11	17.5

For example, you can upgrade version 14.0 to any 15 or 16 version. You can only upgrade version 14.11 to version 15.5 (or later) or version 16.0 (or later).

To see the available updates for Business Central 2019 Release Wave 2, see Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises.

[1]The compatible upgrade versions for this version aren't available yet. If you are currently operating on this version, you'll have to wait until the next round of updates before you upgrade.

Version 15 compatibility

The following table lists the Business Central 15 versions and the minimum 16 and 17 version that's compatible for upgrade.

VERSION 15	VERSION 16	VERSION 17
15.0 to 15.4	16.0	17.0
15.5	16.1	17.0
15.6	16.2	17.0
15.7	16.3	17.0

VERSION 15	VERSION 16	VERSION 17
15.8	16.4	17.0
15.9	16.5	17.0
15.10	16.6	17.0
15.11	16.7	17.1
15.12	16.8	17.2
15.13	16.9	17.3
15.14	16.10	17.4
15.15[1]	16.11	17.5

To see the available updates for Business Central 2020 Release Wave 1, see Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 1 on-premises.

Version 16 compatibility

The following table lists the Business Central 16 versions and the minimum 17 version that's compatible for upgrade.

VERSION 16	VERSION 17
16.0 to 16.5	17.0
16.6	17.1
16.7	17.2
16.8	17.3
16.9	17.4
16.10[1]	17.5

To see the available updates for Business Central 2020 Release Wave 2, see Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

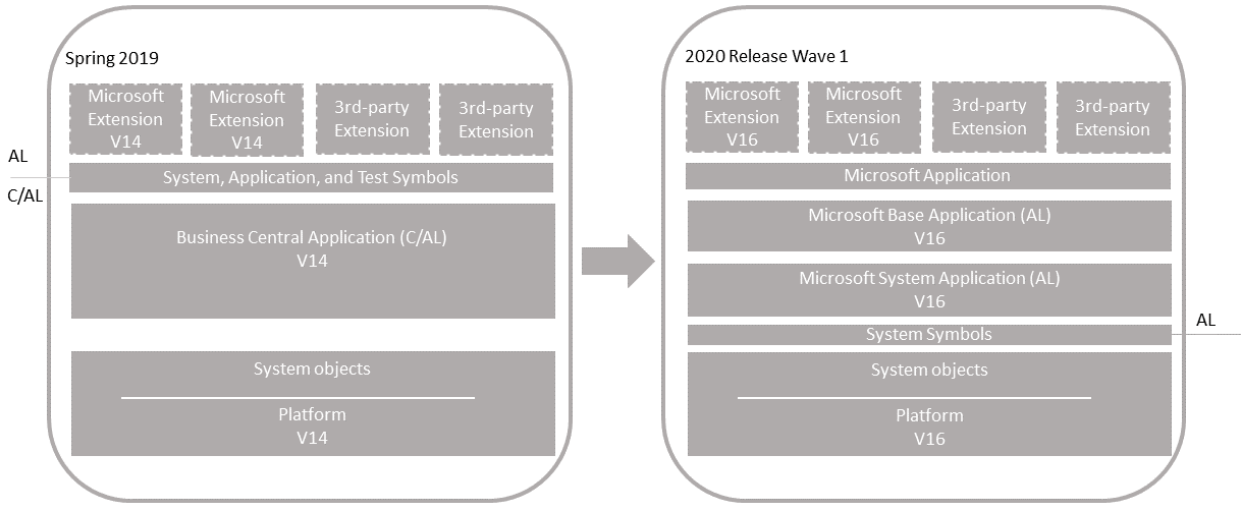
See Also

Upgrading to Business Central

Upgrading Unmodified C/AL Application to Version 16

2/17/2021 • 15 minutes to read • Edit Online

Use this scenario if you have a Business Central Spring 2019 (version 14) application or earlier that doesn't include any code customization. Your solution might include Microsoft (first party) extensions and customization extensions (3rd-party). With this upgrade, you'll replace the C/AL base application with the new system and base application extensions. The result will be a fully upgraded Business Central 2020 release wave 1 (version 16) application and platform.



Single-tenant and multitenant deployments

The process for upgrading the similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application code and business data are in the same database. In a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisite

1. Upgrade to Business Central Spring 2019 (version 14).

   - If your solution is already on version 14, then no action on this step is required.
   - If you're upgrading from Business Central Fall 2018 (version 13) or Dynamics NAV, we recommend you upgrade to the latest update for version 14 that has a compatible update for version 16. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   To download the latest update, go to Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

   For information about how to do the upgrade, see Upgrading to Dynamics 365 Business Central On-Premises.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 16

1. Download the latest available update for version 16 that is compatible with your version 14.

   To download the latest update, go to Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises.

   The guidelines in this article assume that you're running the latest available update.

2. Before you install version 16, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 16 tools.

3. Install Business Central version 16 components.

   You'll have to keep version 14 installed to complete some steps in the upgrade process. When you install version 16, you must either specify different port numbers for components (like the Business Central Server instance and web services) or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 2: Prepare version 14 databases

1. Make backup of the databases.

2. Start Business Central Administration Shell for version 14 as an administrator.

3. Uninstall all extensions from the old tenants.

   In this step, you uninstall any extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>
      ```

      For a single-tenant deployment, set the `<tenant ID>` to default.

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Name <extensions name> -Tenant
      <tenant ID> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the published System Application.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

```
Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-
NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
$_.Version -Force}
```

4. Unpublish all extensions from the application server instance.

   To unpublish an extension, use the Unpublish-NAVApp cmdlet:

   ```
   Unpublish-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension
   version>
   ```

   Together with the Get-NAVAppInfo cmdlet, you can unpublish all extensions by using a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
   server instance> -Name $_.Name -Version $_.Version }
   ```

5. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

6. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

7. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 3: Convert the version 14 database

This task runs a technical upgrade on the application database to convert it from the version 14 platform to the version 16 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 16 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

   When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (14-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 4: Configure version 16 server for DestinationAppsForMigration

When you installed version 16 in **Task 1**, a version 16 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 14 to version 16.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Configure the server instance for migrate extensions to the use the new base application and system application extensions.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"63ca2fa4-4f03-4f2b-a480-172fef340d3f",
   "name":"System Application", "publisher": "Microsoft"},{"appId":"437dbf0e-84ff-417a-965d-
   ed2bb9650972", "name":"Base Application", "publisher": "Microsoft"}]'
   ```

   This setting serves the following purposes:

   - When you run the data upgrade on a tenant, the server will run the data upgrade for the base and system application extensions. The base and system applications will be automatically installed on the tenant also.
   - Lets you republish extensions that haven't been built on version 16. The extensions typically include the third-party extensions that were used in your version 14. When you publish the extensions, the extension manifests are automatically modified with a dependency on the base and system applications.

   For more information about this setting, see DestinationAppsForMigration.

3. Disable task Scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

4. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 5: Increase application version

This task is optional, but it's recommended. You can choose to skip it for now and do it later. In this task, you'll

increase the application_version that's stored in ndodbproperty table of the application database. The application version isn't changed automatically. The application version serves two purposes:

- Enables running the Start-NAVDataUpgrade cmdlet later in Task 8 of this article. The application version is compared with the tenant's version. If the application version is greater, a data upgrade can be run. If you skip this task, you'll have to use the `-SkipAppVersionCheck` switch with Start-NAVDataUpgrade cmdlet in Task 9.
- The application version is shown in the client on the **Help and Support** page. This task ensures that page displays the latest application version.

The version has the format `major.minor.build.revision`, such as, '14.3.14824.1'. As a minimum, you increase the revision by 1. However, we recommend setting the value to application build number for the version 16 update. You can get this number from the Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises page.

To change the application version, run the Set-NAVApplication cmldet:

```
Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application version> -
Force
```

For example:

```
Set-NAVApplication -ServerInstance BC160 -ApplicationVersion 16.0.38071.0 -Force
```

Later, when you synchronize and upgrade the tenant(s), the new application version will be updated in the tenant database (ndotenantproperty table).

Task 6: Import version 16 license

1. Use the Import-NAVServerLicense to upload the version 16 license to the database.

   ```
   Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path and file name>
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 7: Publish symbols and extensions

In this task, you'll publish the platform symbols and extensions. As minimum, you publish the new base application and system application extensions from the installation media (DVD). You also publish new versions of any Microsoft extensions and third-party extensions that were used on your old deployment.

Publishing an extension adds the extension to the application database that is mounted on the server instance. Once published, it's available for installing on tenants. This task updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 14 that you started in the previous task.

1. Publish version 16 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The

symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment.

```
Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
SymbolsOnly
```

What are symbols?

2. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
   Application.app>"
   ```

   What is the System Application?

3. Publish the Business Central base application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the (Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
   Application.app>"
   ```

4. Publish the Microsoft_Application extension

   The Microsoft_Application extension is a new extension introduced in 15.3. For more information about this extension, see The Microsoft_Application.app File.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<folder path>\Microsoft_Application.app"
   ```

5. Publish the new versions of Microsoft extensions.

   In this step, you publish new versions of Microsoft extensions that were used on your version 14 deployment. You find the extensions in the **Applications** folder of the installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft extension>"
   ```

   For example:

   ```
   Publish-NAVApp -ServerInstance BC160 -Path
   "C:\W1DVD\Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app"
   ```

6. Publish 3rd-party extensions.

   Publish 3rd-party extensions that were used on your version 14 solution. If you have new versions of these extensions, built on the Business Central version 16, then publish the new versions. Otherwise, republish the same versions that were previously published in the old deployment.

   ```
   Publish-NAVApp -ServerInstance BC150 -Path "<path to extension>"
   ```

Task 8: Restart server instance

Restart the Business Central Server to free up resources for completing the upgrade.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

This step is important, otherwise you might experience issues when you run the data upgrade.

Task 9: Synchronize tenant

In this task, you'll synchronize the tenant's database schema with any schema changes in the application database and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 16 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
   <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

   > **IMPORTANT**
   > You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   > **NOTE**
   > For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

   At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the **System Application** extension.

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application. To get the version, you can use the Get-NAVAppInfo cmdlet.

4. Synchronize the tenant with the Business Central Base Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

5. Synchronize the tenant with the Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
   ```

6. Synchronize the tenant with Microsoft and 3rd-party extensions.

   For each extension, run the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance BC160 -Tenant default -Name "<extension name>" -Version <extension
   version>
   ```

TIP

When you synchronize an extension, the extension takes ownership of any tables that it includes. In SQL Server, you'll notice that the table names will be suffixed with the extension ID. For example, Base Application tables will have `437dbf0e-84ff-417a-965d-ed2bb9650972` in the name. In addition, the systemId column is added to application tables that are not already part of an extension.

Task 10: Upgrade data

In this task, you run a data upgrade on tables to handle data changes made by platform and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. Upgrade the data to the platform, system application, and base application.

   a. To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

      ```
      Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -
      FunctionExecutionMode Serial [-SkipAppVersionCheck]
      ```

      You only need to use the -SkipAppVersionCheck if you didn't increase the application version in Task 5.

   b. To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

   This step will automatically install the base application and system application on the tenant.

2. Upgrade the new versions of Microsoft extensions and third-party extensions.

   Complete this task to upgrade any Microsoft extension and third-party extension. Microsoft extensions used in the old deployment to new versions on the installation media. The new versions are in the **Application** folder of the DVD. There's a folder for each extension. The extension package (.app file) is in the **Source** folder.

   a. Install **Application** extension.

You'll have to install the **Application** extension first, otherwise you can't upgrade Microsoft extensions.

```
Install-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
```

b. For each extension, run Start-NAVAppDataUpgrade cmdlet:

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Name "<extension name>" -
Version <extension version>
```

This step will also automatically install the new extension version on the tenant.

3. (Multitenant only) Repeat steps 1 through 3 for each tenant.

Task 11: Install 3rd-party extensions

Complete this task to install third-party extensions for which a new version wasn't published. For each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
```

Task 12: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is `C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`.

5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC160'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

At this point, the upgrade is complete, and you can open the client.

Post-upgrade tasks

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
4. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 16 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - Assign the **EXCEL EXPORT ACTION** permission set to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate

permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

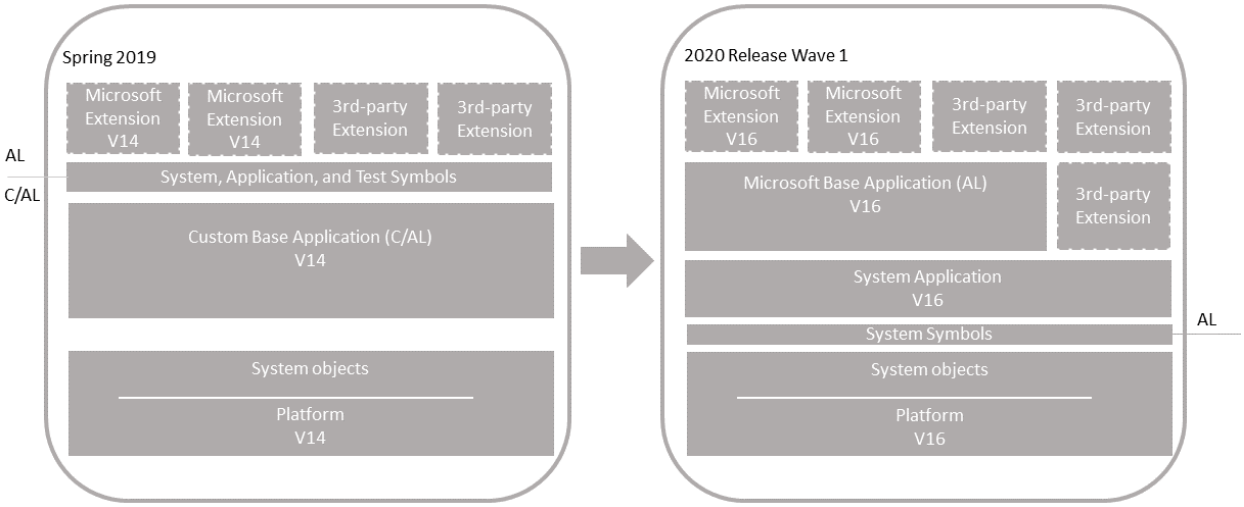
See Also

Publishing and Installing an Extension
Upgrading to Business Central
Upgrading Extensions

Upgrading Customized C/AL Application to Microsoft Base Application Version 16

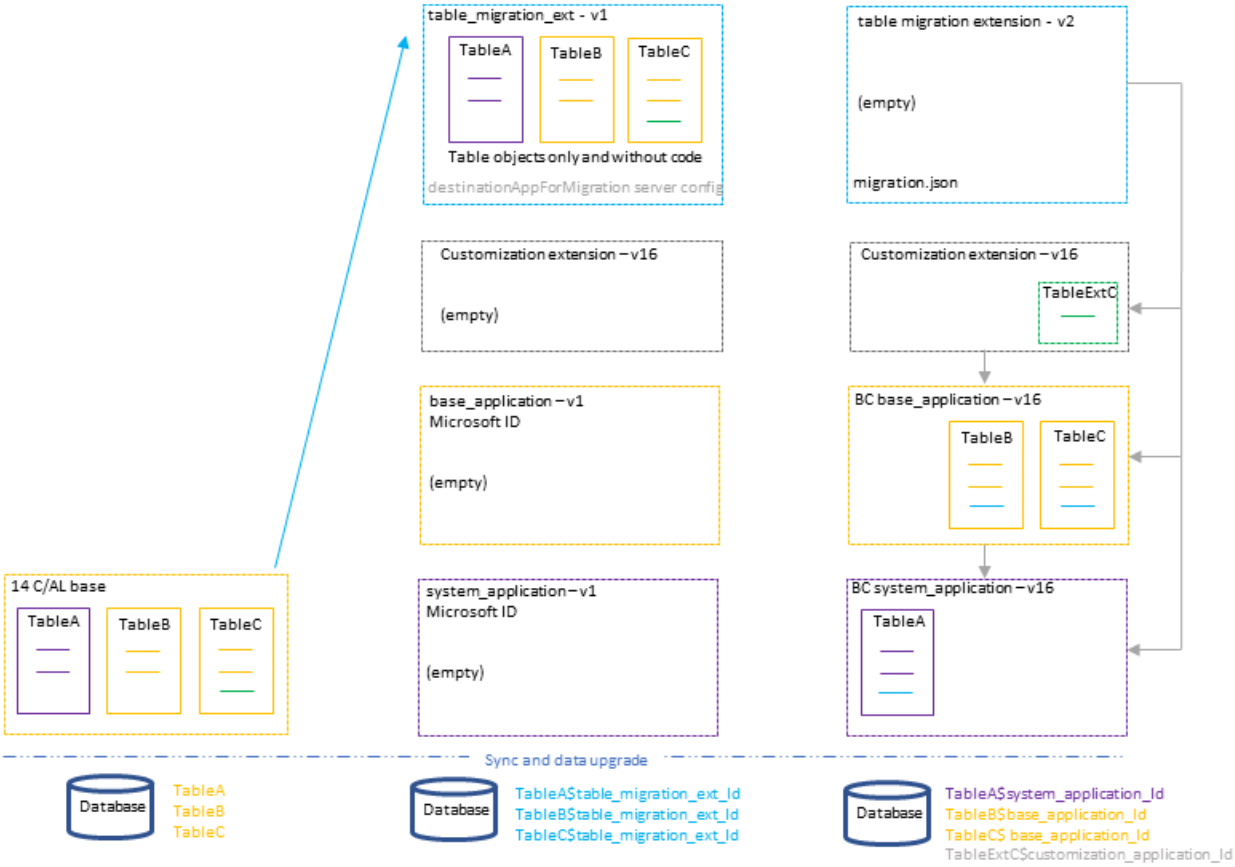
2/17/2021 • 20 minutes to read • Edit Online

This article describes how to upgrade a customized version 14 application to a version 16 solution that uses the Microsoft Base Application.



Overview

The upgrade is divided into two sections: Application Upgrade and Data Upgrade. The Application Upgrade section deals with upgrading the application code. For the application upgrade, you'll have to create several extensions. Some of these extensions are only used for upgrade purposes. The Data Upgrade section deals with upgrading the data on tenants - publishing, syncing, and installing extensions. For this scenario, the data upgrade consists of two phases for migrating data from the current tables to extension-based tables. The following figure illustrates the upgrade process.



The process uses two special features for migrating tables and data to extensions:

- destinationappsformigration server setting

  The *destinationappsformigration* setting is a configuration setting on the Business Central Server. In short, it's used to transfer ownership of the existing tables to the table migration extension. For more information, see DestinationAppsForMigration.

- migration.json file

  The *migration.json* file is used to migrate tables and fields from one extension to another. In this case, migration is from the table migration extension to system and base application tables. For more information about the migration.json, see The Migration.json File.

Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

- Get the required version of the txt2al conversion tool.

  During the upgrade, you'll use the txt2al conversion tool to convert existing tables to the AL syntax. You'll need to use a version of txt2al conversion tool that supports the `--tableDataOnly` parameter. This parameter was first introduced in version 14.12 (cumulative update 11, platform 14.0.41862). So if you're upgrading from version 14.11 (cumulative update 10) or earlier, you'll have to download the txt2al conversion tool from a later version 14 update. See Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

Install version 16

1. Download the latest available update for Business Central 2020 release wave 1, version **16.5** or later, that is compatible with your version 14.

   For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   > **IMPORTANT**
   > You must use version 16.5 or later, otherwise you'll run problems during upgrade because of missing enum objects later in this procedure. For more information, see Some Known Issues.

2. Before you install version 16, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 16 tools.

3. Install Business Central version 16 components.

   You'll have to keep version 14 installed to complete some steps in the upgrade process. When you install version 16, you must either specify different port numbers for components (like the Business Central Server instance and web services) or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

APPLICATION UPGRADE

This section describes how to upgrade the application code. This work involves creating various extensions.

Task 1: Move code customizations to extensions

The first step, and the largest step, is to create extensions for the customizations compared to the Microsoft base application.

Task 2: Create table migration extension

In this step, you create an extension that consists only of the non-system table objects from your custom base application. The table objects will only include the properties and field definitions. They won't include AL code on triggers or methods. This extension is an interim extension used only during the upgrade.

You'll create two versions of this extension. The first version contains the table objects. The second version, is an empty extension that contains a migrate.json file.

Create the first version

1. Create a folder where you'll store exported txt files for tables (for example, C:\export2al\bc14tablesonly).

2. Start the Dynamics NAV Development Shell for version 14.

3. Run Export-NAVApplicationObject cmdlet to export only tables from the database.

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "C:\export2al\bc14tablesonly\exportedbc14-tables.txt" -Filter
'Type=Table;Id=1..1999999999'
```

4. Use the txt2al conversion tool to convert the exported tables to the AL syntax. Use the `--tableDataOnly`

parameter to include table and field definitions only.

```
txt2al --source=C:\export2al\bc14tablesonly --target=C:\export2al\bc14tablesonly\al --tableDataOnly
```

For more information about this tool, see The Txt2Al Conversion Tool.

> **NOTE**
> If the `--tableDataOnly` parameter isn't available, you'll need a later version ot the txt2al conversion tool. See Prerequisites for more information.

5. Make sure you've installed the latest AL Extension for Visual Studio Code from the version 16 DVD.

   For more information, see Getting Started with AL.

6. In Visual Studio Code, create an AL project for table migration extension using the **AL: Go!** command.

   Set the target platform to **5.0 Business Central 2020 release wave 1**.

   For more information, see Getting Started with AL.

7. Configure the project's app.json file:

   - Set the `"name"`, `"publisher"`, and `"version"`. You can use any valid values.
   - Delete the `"application"` parameter.
   - Clear the `"dependencies"`.
   - Set the `"idRanges"` to cover the table object IDs or clear all values.
   - Add the `"target"` parameter and set it to `"Onprem"`.

   Make a note of the `"id"` setting value, which is the ID assigned to the table migration extension. You'll use this ID later in the process. **Don't** use the ID in the following example. It's for illustration purposes only.

```
{
  "id": "11111111-aaaa-2222-bbbb-333333333333",
  "name": "bc14tablesonly",
  "publisher": "My publisher",
  "version": "1.0.0.0",
  "brief": "",
  "description": "",
  "privacyStatement": "",
  "EULA": "",
  "help": "",
  "url": "",
  "logo": "",
  "dependencies": [],
  "screenshots": [],
  "platform": "16.0.0.0",
  "idRanges": [  ],
  "contextSensitiveHelpUrl": "https://bc14tablesonly.com/help/",
  "showMyCode": true,
  "runtime": "5.0",
  "target": "OnPrem"
}
```

8. Create an `.alpackages` folder in the root folder of the project and then copy the system (platform) symbols extension (System.app file) to the folder.

The System.app file is located where you installed the AL Development Environment. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment. This package contains the symbols for all the system tables and codeunits.

9. Add the AL files that you converted tables to the project.

   The folder used in this article is C:\export2al\bc14tablesonly\al.

   > **IMPORTANT**
   > If you're upgrading a **CH** or **NA** local version (14.18 or later), you'll have to rename the primary keys in some tables. For more information, see Known Issues.

10. Build the extension package for the first version.

    To build the extension package, press Ctrl+Shift+B. This step creates an .app file for your extension. The file name has the format <publisher>_<name>_<version>.app.

Create the second version

1. In Visual Studio Code, create a new file called migration.json file and add it to the project's root folder.
2. In the migration.json, include rules for the Microsoft base application, system application, and your customization extensions.

```
{
    "apprules": [
        {
            "id": "63ca2fa4-4f03-4f2b-a480-172fef340d3f"
        },
        {
            "id": "437dbf0e-84ff-417a-965d-ed2bb9650972"
        },
        {
            "id": "<NNNNNNNN-NNNN-NNNN-NNNN-NNNNNNNNNNNN>"
        },
        {
            "id": "<NNNNNNNN-NNNN-NNNN-NNNN-NNNNNNNNNNNN>"
        }
    ]
}
```

   In the example code:

   - `63ca2fa4-4f03-4f2b-a480-172fef340d3f` identifies the system application extension
   - `437dbf0e-84ff-417a-965d-ed2bb9650972` identifies the base application extension
   - The two other IDs are examples that identify other new customization extensions you might have. Replace or remove these entries as needed.

   For more information about the migration.json, see The Migration.json File.

3. Delete the AL objects.
4. Increase the `version` in the app.json file.
5. Build the extension package for the second version.

   To build the extension package, press Ctrl+Shift+B.

Task 3: Create empty extensions System, Base, and customization

extensions

Create two empty extensions: one for the Microsoft base application and another for the System Application. Also, create an empty extension for each new customization extension. The only file in the extension project that is required is an app.json.

You can create the empty extension like any other extension by adding an AL project in Visual Studio Code:

1. In Visual Studio Code, select **View** > **Command Palette** > **AL: Go!** and follow instructions.
2. Delete the HelloWorld.al sample file from the project.
3. Modify the app.json file.

   The important settings in the app.json file `"id"`, `"name"`, `"version"`, `"publisher"`, and `"dependencies"`.

   - The `id` and `name` must match the value used by Microsoft's extensions.
   - Set the `version` to any version lower than 16.0.0.0.
   - You'll also have to include the `"publisher"`. You can use your own publisher name or `"Microsoft"`.
   - Remove all other settings. It's important that there are no `"dependencies"` set.

   The app.json files for the **System Application** and **Base Application** extensions, should look similar to following examples:

   System Application

   ```
   "id": "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
   "name": "System Application",
   "publisher": "Microsoft",
   "version": "14.0.0.0",
   ```

   Base Application

   ```
   "id": "437dbf0e-84ff-417a-965d-ed2bb9650972",
   "name": "Base Application",
   "publisher": "Microsoft",
   "version": "14.0.0.0",
   ```

4. Build and compile the extension package. To build the extension package, press Ctrl+Shift+B.

> **TIP**
> This step is only required if you need to trigger a data upgrade on these extensions, which you'll do by running Start-NavAppDataUpgrade on these extensions in Task 15. For the scenario in this article, at a minimum this step is required for the System and Base Applications. You can skip this step for any customization extensions that do not not include upgrade code.

DATA UPGRADE

Task 4: Prepare databases

1. Make backup of the databases.
2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

For more information, see Managing Encryption and Encryption Keys.

Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

3. Start Business Central Administration Shell for version 14 as an administrator.
4. (Single-tenant only) Uninstall all extensions from the tenants.

   To uninstall an extension, you use the Uninstall-NAVApp cmdlet. For example, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-NAVApp -
   ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version $_.Version -Force}
   ```

5. (Single-tenant only) Unpublish all extensions from the application server instance.

   To unpublish an extension, use the Unpublish-NAVApp cmdlet. For example, you can unpublish all extensions by using a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
   server instance> -Name $_.Name -Version $_.Version }
   ```

6. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

7. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

8. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 5: Convert version 14 database

This task runs a technical upgrade on the application database. The task converts the database from the version 14 platform to the version 16 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 16 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (14-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 6: Configure version 16 server for DestinationAppsForMigration

In this step, you configure the version 16 server instance. In particular, you configure it to migrate the table migration extension that you created earlier. The migration is controlled by the `DestinationAppsForMigration` setting for the server instance. For more information about the `DestinationAppsForMigration` setting, see DestinationAppsForMigration.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Configure the `DestinationAppsForMigration` setting of the server instance to table migration extension.

   You'll need the ID, name, and publisher for the table migration extension that you created in **Task 2**.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"<table migration extension ID>", "name":"<table
   migration extension>", "publisher": "<publisher>"}]'
   ```

   > **NOTE**
   > You can add multiple extensions to this setting.

3. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

4. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 7: Import License

Import the version 16 partner license. To import the license, use the Import-NAVServerLicense cmdlet:

```
Import-NAVServerLicense $-ServerInstance <server instance name> -LicenseFile <path>
```

Restart the server instance.

Task 8: Increase application version

This task is optional, but it's recommended. You can choose to skip it for now and do it later. In this task, you'll increase the application_version that's stored in ndodbproperty table of the application database. The application version isn't changed automatically.

The application version serves two purposes:

- Enables running the Start-NAVDataUpgrade cmdlet later. The application version is compared with the tenant's version. If the application version is greater, a data upgrade can be run. If you skip this task, you'll have to use the `-SkipAppVersionCheck` switch with Start-NAVDataUpgrade cmdlet in Task 11.
- The application version is shown in the client on the **Help and Support** page. This task ensures that page displays the latest application version.

The version has the format `major.minor.build.revision`, such as, '14.3.14824.1'. As a minimum, you increase the revision by 1. However, we recommend setting the value to application build number for the version 16 update. You can get this number from the Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 1 on-premises page.

1. To see the current application version, run the Get-NAVApplication cmdlet:

   ```
   Get-NAVApplication -ServerInstance <server instance name>
   ```

2. To change the application version, run the Set-NAVApplication cmdlet:

   ```
   Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
   version> -Force
   ```

   For example:

   ```
   Set-NAVApplication -ServerInstance BC160 -ApplicationVersion 16.0.38071.0 -Force
   ```

Task 9: Publish symbols and DestinationAppsForMigrations

In this task, you'll publish the platform symbols and the extensions configured as DestinationAppsForMigration.

1. Publish version 16 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment.

   ```
   Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
   SymbolsOnly
   ```

   What are symbols?

2. Publish the first version of the table migration extension, which is the version that contains the table objects.

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to extension .app file>"
```

3. Publish the empty versions of the following extensions:

   - **System Application** extension
   - **Base Application** extension
   - Customization extensions (if any).

   This step publishes the extensions you created in Task 3. Publish the extensions using the Publish-NAVApp, like in the previous steps. Except if the extensions aren't signed, use the `-SkipVerification` switch parameter.

4. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 10: Synchronize tenant

In this task, you'll synchronize the tenant's database schema with any schema changes in the application database and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 15 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

```
Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
<database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
```

   IMPORTANT

   You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   NOTE

   For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

   At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

```
Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the table migration extension.

Use the Sync-NAVApp cmdlet:

```
Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "<table migration
extension>" -Version <extension version>
```

This step creates empty tables in the database for the table objects defined in the table migration extension. When completed, the table migration extension takes ownership of the table. In SQL Server, you'll notice that the table names will be suffixed with the extension ID. At this point, the tenant state is OperationalDataUpgradePending.

> **TIP**
> To verify the tenant state, run Get-NAVTenant cmdlet with the `-ForceRefresh` switch:
>
> `Get-NAVTenant <server instance> -Tenant <default> -ForceRefresh`

4. Synchronize the empty versions of system application, base application, and customization extensions that you published in Task 9.

Task 11: Install DestinationAppsForMigration and move tables

In this task, you run a data upgrade on tables to handle data changes made by platform and extensions. The step installs table migration extension. It moves data from the old tables to the new tables owned by the table migration extension.

1. To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -
   FunctionExecutionMode Serial -SkipCompanyInitialization [-SkipAppVersionCheck]
   ```

   You only need to use the -SkipAppVersionCheck if you didn't increase the application version in Task 8. This step will automatically install the table migration extension.

2. To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

3. Install the empty versions of the system, base, and custom extensions that you published in Task 9.

   To install the extension, you use the Install-NAVApp cmdlet.

   ```
   Install-NAVApp -ServerInstance <BC16 server instance> -Name "<name>" -Version <extension version>
   ```

4. (Multitenant only) Repeat steps 1 and 2 for each tenant.

Task 12: Publish final extensions

This step starts the second phase of the data upgrade. You'll publish the second version of the table migration extension and the production versions of extensions. The production extensions include the new versions of Microsoft System Application, Base Application extension, and customization extensions. The extension packages for Microsoft extensions are on the installation media (DVD). Customization extensions include the extension versions that you created in Task 1, not the empty versions.

Publish extensions using the Publish-NAVApp cmdlet like you did in previous steps.

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to extension .app file>"
```

Publish the extensions in the following order:

1. Second version of the table migration extension, which is the empty version with the migration.json file.
2. Microsoft System Application

   Publish the Microsoft_System Application.app extension package file that is in the **Applications\System Application\Source** folder of installation media (DVD).
3. Microsoft Base Application

   Publish the Microsoft_Base Application.app extension package file that is in the **Applications\BaseApp\Source** folder of installation media (DVD).

   > **NOTE**
   > The other .app files in this folder, like Microsoft_Danish language (Denmark).app, are extensions that add translations for a specific language. By publishing and installing these extensions, you add the capability of showing the base application in another language. These extensions aren't required to complete the upgrade and can be published and installed later.
4. Customization extensions.
5. Microsoft and third-party extensions.

   The Microsoft extensions are in the **Applications** folder of installation media (DVD).

Task 13: Synchronize final extensions

Synchronize the newly published extensions using the Sync-NAVApp cmdlet like you did in previous steps.

```
Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "<table migration extension>" -
Version <extension version>
```

Synchronize the extensions in the following order:

1. Microsoft System Application
2. Microsoft Base Application
3. Microsoft and third-party extensions
4. Customization extensions
5. Second version of the table migration extension

> **IMPORTANT**
> Synchronize extensions in the order of dependencies. The migration extension must be synchronized last. This step will change table ownership to the system and base application.

Task 14: Upgrade on table migration extension

Run data upgrade on the table migration extension by using the Start-NAVAppDataUpgrade cmdlet. For example:

```
Start-NAVAppDataUpgrade -ServerInstance <server instance> -Name "<table migration extension>" -version
<version 2>
```

Task 15: Upgrade final extensions

The final step is to upgrade to the new extension versions in the following order. Use the Start-NAVAppDataUpgrade cmdlet like you did in the previous task.

Run the data upgrade on the extensions in the following order:

1. Microsoft System Application.
2. Microsoft Base Application.
3. Customization, Microsoft, and third-party extensions.

   For customization extensions, only do this task for those extensions that have an empty version currently installed on the tenant (see Task 11). If you have a customization extension for which you didn't create and publish an empty version, complete the next task for these extensions.

Task 16: Install remaining customization extensions

Complete this task for cutomizations extension that you created in Task 1, but create and publish an empty version first.

To install each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <BC16 server instance> -Name "<name>" -Version <extension version>
```

Task 17: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins that must be upgraded.

- Microsoft.Dynamics.Nav.Client.BusinessChart
- Microsoft.Dynamics.Nav.Client.FlowIntegration
- Microsoft.Dynamics.Nav.Client.OAuthIntegration
- Microsoft.Dynamics.Nav.Client.PageReady
- Microsoft.Dynamics.Nav.Client.PowerBIManagement
- Microsoft.Dynamics.Nav.Client.RoleCenterSelector
- Microsoft.Dynamics.Nav.Client.SocialListening
- Microsoft.Dynamics.Nav.Client.SatisficationSurvey
- Microsoft.Dynamics.Nav.Client.TimelineVisualization
- Microsoft.Dynamics.Nav.Client.VideoPlayer
- Microsoft.Dynamics.Nav.Client.WebPageViewer
- Microsoft.Dynamics.Nav.Client.WelcomeWizard

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.

4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is `C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`.

5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC160'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Post-upgrade tasks

1. Uninstall the table migration extension.
2. Enable task scheduler on the server instance.
3. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
4. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

5. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 16 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - Assign the **EXCEL EXPORT ACTION** permission set to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate permission sets.

   For more information about working with permission sets and permissions, see Export and Import Permission Sets.

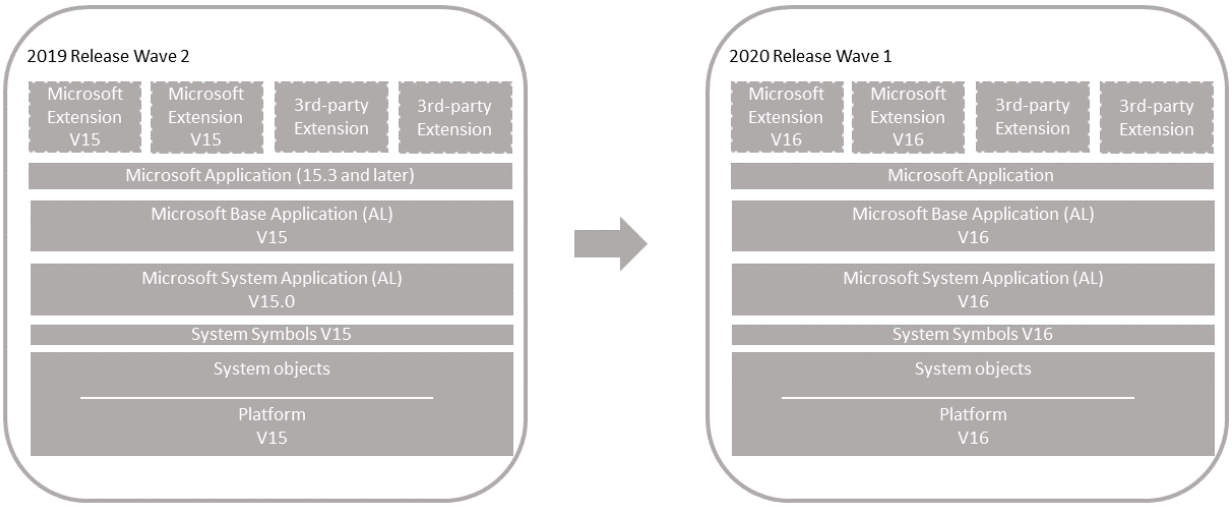
See Also

Publishing and Installing an Extension
Upgrading to Business Central
Signing an APP Package File

Upgrading Version 15 Base Application to Version 16

2/17/2021 • 13 minutes to read • Edit Online

Use this scenario if you have a Business Central 2019 release wave 2 solution that uses the Microsoft System and Base applications.



Single-tenant and multitenant deployments

The process for upgrading the similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application code and business data are in the same database. In a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Your version 15 is compatible with version 16.

   There are several updates for version 15. The updates have a compatible version 16 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix. For example, if your solution is currently running 15.5, you can't upgrade to 16.0. You must wait until 16.1 is available.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 16

1. Download the latest available update for Business Central 2020 (version 16) that is compatible with your version 15.

   For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

2. Before you install version 16, it can be useful to create desktop shortcuts to the version 15.0 tools, such

as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 16 tools.

3. Install Business Central version 16 components.

   You'll have to keep version 15 installed to complete some steps in the upgrade process. When you install version 16, you must either specify different port numbers for components (like the Business Central Server instance and web services) or you must stop the version 15.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 2: Prepare version 15 databases

1. Make backup of the databases.
2. Start Business Central Administration Shell for version 15 as an administrator.
3. (Single-tenant only) Uninstall all extensions from the old tenants.

   In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>
      ```

      For a single-tenant deployment, set the `<tenant ID>` to default.

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Name <extensions name> -Tenant
      <tenant ID> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the published System Application.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-
      NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
      $_.Version -Force}
      ```

4. Unpublish all system symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

```
Get-NAVAppInfo -ServerInstance <BC15 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
ServerInstance <BC15 server instance> -Name $_.Name -Version $_.Version }
```

What are symbols?

5. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

```
Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
```

6. Stop the server instance.

```
Stop-NAVServerInstance -ServerInstance <server instance name>
```

Task 3: Convert version 15 database

This task runs a technical upgrade on the application database to convert it from the version 15 platform to the version 16 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 16 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
instance> -DatabaseName "<database name>"
```

   When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (15-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 4: Configure version 16 server

When you installed version 16 in **Task 1**, a version 16 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 15 to version 16 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

```
Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
<database name>"
```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task Scheduler on the server instance for purposes of upgrade.

```
Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
KeyValue false
```

Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 5: Import version 16 license

1. Use the Import-NAVServerLicense to upload the version 16 license to the database.

```
Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path and file name>
```

2. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 6: Publish symbols and extensions

In this task, you'll publish the platform symbols and extensions. As minimum, you publish the new base application and system application extensions from the installation media (DVD). You also publish new versions of any Microsoft extensions and third-party extensions that were used on your old deployment.

Publishing an extension adds the extension to the application database that is mounted on the server instance. Once published, it's available for installing on tenants. This task updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 16 that you started in the previous task.

1. Publish version 16 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment.

```
Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
SymbolsOnly
```

   What are symbols?

2. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
Application.app>"
```

What is the System Application?

3. Publish the Business Central base application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the (Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
   Application.app>"
   ```

4. Publish the Microsoft_Application extension (when coming from 15.2 and earlier only).

   The Microsoft_Application extension is a new extension introduced in 15.3. For more information about this extension, see The Microsoft_Application.app File.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Application.app>"
   ```

5. Publish the new versions of Microsoft extensions.

   In this step, you publish new versions of Microsoft extensions that were used on your version 15 deployment. You find the extensions in the **Applications** folder of the installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft extension>"
   ```

   For example:

   ```
   Publish-NAVApp -ServerInstance BC150 -Path
   "C:\W1DVD\Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app"
   ```

6. Publish new versions of 3rd-party extensions.

   If you have new versions of these extensions, built on the Business Central version 16, then publish the new versions.

   ```
   Publish-NAVApp -ServerInstance BC160 -Path "<path to extension>"
   ```

7. Recompile extensions not build on version 16.

   This step pertains to any published extension versions that aren't built on version 16, which you want to reinstall on tenants. These extensions must be recompiled to work with version 16. To recompile the extensions, use the Repair-NAVApp cmdlet:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   For example, to recompile all extensions that are not published by Microsoft, you could run the following command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Where-Object {$_.Publisher -notlike 'Microsoft'} |
   Repair-NAVApp
   ```

   Restart the Business Central Server when completed.

Task 7: Synchronize tenant

You'll synchronize the tenant's database schema with any schema changes in the application database and extensions. If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 15 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
   <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

   > **IMPORTANT**
   > You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   > **NOTE**
   > For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

   At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the **System Application** extension.

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application. To get the version, you can use the Get-NAVAppInfo cmdlet.

4. Synchronize the tenant with the Business Central Base Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

5. Synchronize the tenant with the Application extension (when coming from 15.2 and earlier only).

```
Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
```

6. Synchronize the tenant with Microsoft and 3rd-party extensions.

   For each extension, run the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance BC150 -Tenant default -Name "<extension name>" -Version <extension
   version>
   ```

Task 8: Upgrade data

In this task, you run a data upgrade for extensions.

Single tenant

Run the data upgrade on extensions in order of dependency.

1. Run the data upgrade for the system application, followed by the base application.

   To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Name "<extension name>" -Version
   <extension version>
   ```

   This step will automatically install the new system application and base application versions on the tenant.

2. Upgrade the new versions of Microsoft extensions and third-party extensions.

   Complete this task to upgrade any Microsoft and third-party extension used in the old deployment to new versions on the installation media. The new versions are in the **Application** folder of the DVD. There's a folder for each extension. The extension package (.app file) is in the **Source** folder.

   a. Install **Application** extension (when comping from 15.2 and earlier only).

      You'll have to install the **Application** extension first, otherwise you can't upgrade Microsoft extensions.

      ```
      Install-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
      ```

   b. For each extension, run Start-NAVAppDataUpgrade cmdlet.

      This step will also automatically install the new extension version on the tenant.

Multitenant

On each tenant, run the Start-NavDataUpgrade cmdlet as follows to change the version number:

```
Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -FunctionExecutionMode
Serial -SkipAppVersionCheck
```

Task 9: Install 3rd-party extensions

Complete this task to install third-party extensions for which a new version wasn't published. For each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
```

Task 10: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is
   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```
   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC160'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

At this point, the upgrade is complete, and you can open the client.

Post-upgrade tasks

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
4. Change application version.

   (Optional) This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version.

   On the **Help and Support** page in the client, you'll see an application version, such as 16.0.2345.6. For an explanation of the number, see Version numbers in Business Central. This version isn't updated automatically when you install an update. If you want the version to reflect the version of the update or your own version, you change it manually.

We recommend setting the value to application build number for the version 15 update. You get the number from the Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 1 on-premises.

a. Run the Set-NAVApplication cmdlet:

```
Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
version> -Force
```

For example:

```
Set-NAVApplication -ServerInstance BC160 -ApplicationVersion 16.0.38071.0 -Force
```

b. Run the Sync-NAVTenant cmdlet to synchronize the tenant with the application database.

```
Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
```

With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

c. Run the Start-NavDataUpgrade cmdlet to change the version number:

```
Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID>
```

5. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 16 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - Assign the **EXCEL EXPORT ACTION** permission set to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate permission sets.

   For more information about working with permission sets and permissions, see Export and Import Permission Sets.

See Also

Upgrading to Business Central
Upgrading Extensions

Code Conversion from C/AL to AL

2/17/2021 • 11 minutes to read • Edit Online

This article explains how to convert a Business Central (version 14) C/AL code-customized on-premises solution to AL code.

You'll use this procedure as part of the upgrade process when going from version 14 to a later version, like 15, 16, or 17.

Before you start

Get familiar with the basics of setting up and developing in Visual Studio Code and AL, see Developing Extensions in AL.

> **NOTE**
> Moving on-premise C/AL code customizations to Dynamics 365 Business Central online, requires converting these to AL extensions. This could include converting the C/AL deltas to AL extension code as a starting point, as outlined in The Txt2Al Conversion Tool.

Also, review the Known Issues for information about issues that may affect the upgrade.

Breaking changes

When converting from C/AL to AL, it's important that you don't introduce any breaking schema changes to the database. Otherwise, you won't be able to synchronize the new extension with the database.

Task 1: Import the test library into your C/AL solution

If your solution uses Microsoft (1st-party) extensions, you'll have to convert the test library from C/AL to AL. The reason for this is that the Microsoft extensions rely on the test symbols. The easiest way is to import the **CALTestLibraries.W1.fob** file into the old database. This file is available on the version 14 installation media (DVD) in the **TestToolKit** folder.

You can do this using the (Dynamics NAV Development Environment). For more information, see Exporting and Importing Objects.

Task 2: Compile all the objects in your C/AL solution

Compiling all the objects is a prerequisite for a successful and complete export. To compile objects, you can use either of the following tools:

- C/SIDE (Dynamics NAV Development Environment). See Compiling Objects.
- Compile-NAVApplicationObject cmdlet of the Dynamics NAV Development Shell. Make sure to run this cmdlet as an administrator.

Task 3: Export the application objects to the new TXT syntax

Once the application compiles, you must export all C/AL application objects, except system tables and codeunits (IDs in the 2000000000 range), to the new TXT format. The exported objects will be used as input to the Txt2AL conversion tool. To export objects, use the Export-NAVApplicationObject cmdlet of the Dynamics NAV Development Shell. it's important to:

- Omit omit all system objects, which have IDs in the 2000000000 range.
- Use the `ExportToNewSyntax` switch to export the objects in a syntax that is compatible with the Txt2Al conversion tool.

The Export-NAVApplicationObject cmdlet will export all objects to a single .txt file. If you imported the test library objects into the database, then you'll export the base application objects and the test library separately. Later, you'll create a separate AL project for each set of files.

For example, do the following steps:

1. Export the custom base application objects.
   a. Create a folder for storing the exported base application objects to TXT files (for example, c:\export2al\baseapplication).

   b. Run the following commands to export tha application objects, but omitting the system objects and test library objects.

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\baseapplication\exportedbc14app-part1.txt" -Filter
'Id=1..129999'
```

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\baseapplication\exportedbc14app-part2.txt" -Filter
'Id=140000..1999999999'
```

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\baseapplication\exportedbc14testobjects.txt" -Filter
'Id=130400..130416'
```

2. Export the test library objects.
   a. Create a folder for storing the exported test library objects to TXT files (for example, c:\export2al\testlibrary).

   b. Run the following commands to export the test library objects only

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\testlibrary\bc14testlibrary-part1.txt" -Filter
'Id=130000..130399'
```

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\testlibrary\bc14testlibrary-part2.txt" -Filter
'Id=130440..139999'
```

Task 4: Create a declaration file for custom .NET assemblies (optional)

If your solution contains .NET interoperability code and control add-ins, you can create a file that contains the declarations to the assemblies. This file will be used when you convert the C/AL TXT files to AL in the next step. Alternatively, after the conversion, you'll have to manually add the declarations to objects that use the assemblies.

To create the file, use a text editor or Visual Studio code to create a file that contains the assembly declarations as follows:

```
dotnet

{
    assembly("Microsoft.Dynamics.Nav.Client.BusinessChart")
    {

type("Microsoft.Dynamics.Nav.Client.BusinessChart.BusinessChartAddIn";"Microsoft.Dynamics.Nav.Client.Busines
sChart")
        {
            IsControlAddIn = true;
        }

        // Other control add-ins in this assembly
    }

    // Other assemblies containing control add ins
}
```

Save the file with any name and the extension **.al**, for example **mydotnet.al**. Make a note of the path because you'll use it in the next step.

Task 5: Convert the C/AL TXT files to AL

With C/AL exported to the new TXT format, you now convert the code to AL using the The Txt2Al Conversion Tool. The Txt2Al creates .al files for each object in the TXT files. Similar to **Task 3**, if you imported the test library objects into the database, then you'll convert the base application objects and the test library separately.

Get the Txt2AL conversion tool

The Txt2Al conversion tool (txt2al.exe) is only available with version 14, which is the last version to support C/AL. Use this version of the tool no matter what later version you may eventually be upgrading to. The AL objects created by the tool will be compatible with later versions.

You find the txt2al.exe on the installation media (DVD) in the "DVD\RoleTailoredClient\program files\Microsoft Dynamics NAV\140\RoleTailored Client" folder. Or, it's installed locally with Dynamics NAV Development Environment, for example, in the "C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client" folder.

Run the Txt2AL conversion tool

1. Convert the base application TXT files to AL.

    a. Create a folder for storing the AL files for base application objects (for example, c:\export2al\baseapplication\al).

    b. Start a command prompt as administrator, and navigate to the folder that contain txt2al.exe file.

    By default, the location is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client.

    c. Run the txt2al command:

    ```
    txt2al --source=C:\export2al\baseapplication --target=C:\export2al\baseapplication\al --
    injectDotNetAddIns --dotNetAddInsPackage=C:\export2al\dotnet\mydotnet.al --dotNetTypePrefix=BC
    --rename
    ```

    If your solution contains .NET interoperability code, the following Txt2Al command line parameters are used to achieve a conversion that requires less manual intervention:

    - `--injectDotNetAddIns` injects the definition of standard .NET add-ins in the resulting .NET

package. The standard .NET add-ins are a set of add-ins that are embedded into the platform.

- `--dotNetAddInsPackage` should be used to point the conversion tool to an AL file containing declarations for the .NET types that represent .NET control addins. Use this to inject a custom set of .NET control add-in declarations. This parameter is only required if you completed **Task 4**, and you set it to point to the location of the dotnet.al file.

  > **NOTE**
  > If you are interested in migrating your localization resources, you should use the `--addLegacyTranslationInfo` switch to instruct Txt2Al to generate information about the legacy IDs of the translation code.

- `--dotNetTypePrefix` specifies a prefix to be used for all .NET type aliases created during the conversion. This will ensure that no naming conflicts occur with existing types. In the example, `BC` is the prefix.

- `--rename` renames the output files to prevent clashes with the source .txt files.

When completed, there will be an .al file for each object.

2. Convert the test library TXT files to AL.

   This is similar to the previous step.

   a. Create a folder for storing the AL files for base application objects (for example, c:\export2al\baseapplication\al).

   b. Run the txt2al command:

   ```
   txt2al --source=C:\export2al\testlibrary --target=C:\export2al\testlibrary\al --
   injectDotNetAddIns --dotNetTypePrefix=BCTest --rename
   ```

   Use a different value for the `--dotNetTypePrefix` than you did for the base application.

Task 6: Create a new application database for development

To build your base application, you'll create a new application database on the Business Central version 15, 16, or 17 platform. This will only be used during development.

1. Start the Business Central Administration Shell for version 16 as an administrator.
2. Run the New-NAVApplicationDatabase cmdlet to create the database. For example:

   ```
   New-NAVApplicationDatabase -DatabaseServer .\BCDEMO -DatabaseName MyDBforupgrade
   ```

3. Connect your Business Central Server instance to the database. See Connecting a Business Central Server Instance to a Database.

   ```
   Set-NAVServerConfiguration -ServerInstance BC -KeyName DatabaseName -KeyValue "MyDBforupgrade"
   ```

4. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance BC
```

Task 7: Create and build an AL project for custom base application

In this task, you'll create a AL project in Visual Studio code that you'll use for building your custom base application extension based on your converted C/AL application.

1. If you haven't already, install Visual Studio Code and the latest AL Language extension for version 15, 16, or 17 as outlined in Getting Started with AL.

2. Configure Visual Studio Code for optimal performance with AL projects.

   This step is optional, but recommended. For more information, see Optimize Visual Studio Code for Editing and Building.

3. In Visual Studio Code, from the **Command Palette**, select the **AL Go!** command to create a new project.

   Specify the path for the project, and set the **Target Platform** to **4.0 Business Central 2019 release wave 2** or **5.0 Business Central 2020 release wave 1**. When prompted to select your server, choose **Your own server**.

4. Create a **.alpackages** folder in the root folder of the project and then copy the system (platform) symbols extension (System.app file) to the folder.

   The System.app file is located where you installed the AL Development Environment, which by default is the C:\Program Files (x86)\Microsoft Dynamics 365 Business Central<150, 160, or 170>\AL Development Environment folder. This package contains the symbols for all the system tables and codeunits.

5. Delete the **HelloWorld.al** sample file from the project.

6. Modify the `settings.json` file of Visual Studio Code to configure the assembly probing path.

   Change `"al.assemblyProbingPaths": ["./.netpackages"]` to point to all the folders that contain .NET assemblies that are used by your project. Here is an example that contains the most typical paths:

   ```
   "al.assemblyProbingPaths": [
   "C:\\Program Files\\Microsoft Dynamics 365 Business Central\\150",
   "C:\\Program Files (x86)\\Microsoft Dynamics 365 Business Central\\150\\RoleTailored Client",
   "C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\Framework\\.NETFramework\\v4.7.2",
   "C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\WindowsPowerShell\\3.0"
   ]
   ```

   For more information about the settings.json, see User and Workspace Settings.

   > **NOTE**
   > Adding assemblies to the folders in your assembly probing paths is not automatically detected by the compiler. You must restart Visual Studio Code for the changes to be detected.

7. Modify the `app.json` for the project as follows:

   - **Important** The ID, name, and publisher, and version of the custom base application must match the Business Central base application. Set the parameters to the following values`:

```
"appId": "437dbf0e-84ff-417a-965d-ed2bb9650972",
"name": "Base Application",
"publisher": "Microsoft",
"version": "14.5.0.0"
```

We recommend that you set the "version" to the same version as the C/AL application.

- Add the setting `"target": "OnPrem"` somewhere in the file.
- Change the `idRange` to include all the IDs used by your base application (or leave blank).
- Delete the values in the `dependencies` parameter.

8. Copy all of the base application AL files generated in the previous task (**Task 5**) to the root folder of your project.
9. Open the **dotnet.al** file for the project, and make the following changes:
   - Delete all instances of `Version = '14.0.0.0';` for **Microsoft.Dynamics.Nav** assembly declarations.
   - For the `DocumentFormat.OpenXml` assembly declaration, remove the `version` and `culture` keys and set `PublicKeyToken = '8fb06cb64d019a17'`.

   ```
   assembly("DocumentFormat.OpenXml")
   {
       PublicKeyToken = '8fb06cb64d019a17';
       ...
   ```
10. Delete objects that are related to the client debugger client.

    Debugging from the client has been discontinued, and replaced by AL Debugger. The version 14 debugger objects are not supported on version 15, 16, or 17. To avoid compilation errors, delete the following objects:

    - Debugger.Page.al
    - DebuggerBreakpointCondition.Page.al
    - DebuggerBreakpointList.Page.al
    - DebuggerCallstackFactBox.Page.al
    - DebuggerCodeViewer.Page.al
    - DebuggerManagement.Codeunit.al
    - DebuggerVariableList.Page.al
    - DebuggerWatchValueFactBox.Page.al
    - SessionList.Page.al

    You might also have to remove references to `SessionList` in ChangeGlobalDimensions.Codeunit.al.
11. Build and compile your project (press Ctrl+Shift+B).

    The AL compiler will issue errors for constructs that are not valid. Fix any errors that occur, and build again.

> **TIP**
> If you are maintaining your C/AL solution going forward, we recommend that you fix errors in C/AL objects and convert to AL again. This makes it future changes easier to foward push changes because code bases will be similar.

When all errors are fixed, the custom base application package (.app) will be created.

Task 8: Create and build an AL project for the test library

If you converted the test library form C/AL to AL, you'll now create and build a project for test library, similar to what you did for the base application.

a. Follow steps 1 through 5 in **Task 7** to create an AL project for the test library.

b. As with base application project, you have to modify the `app.json` file, but in this case, you have to change the version and add a dependency on the base application that you created.

- Set the `"version"` to the old application version, such as `14.5.0.0`.
- Set the `"dependencies"` to include information about your custom the base application.

```
"dependencies": [
{
    "appId": "437dbf0e-84ff-417a-965d-ed2bb9650972",
    "publisher": "Microsoft",
    "name": "Base Application",
    "version": "14.5.0.0"
}
],
```

- Set the `target` to `OnPrem`.
- Change the `idRange` to include all the IDs used by your test application (or leave blank).

c. Copy all of the AL files that you generated for the test library in **Task 5** to the root folder of your project.

d. Open the **dotnet.al** file for the project, and make the following changes:

- Delete all instances of `Version = '14.0.0.0';` for **Microsoft.Dynamics.Nav** assembly declarations.

e. Build the project.

Next Steps

If you are performing a technical upgrade from version 14.0 to version 15, 16, or 17, return to the technical upgrade step where you left off.

- Technical Upgrade to version 15.0
- Technical Upgrade to to version 16.0
- Technical Upgrade to to version 17.0

See Also

The Txt2Al Conversion Tool
Developing Extensions

Migrating Tables and Fields Between Extensions

2/17/2021 • 4 minutes to read • Edit Online

INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

Data migration allows you to move table and field data between extensions. The concepts and processes in this article apply to large-scale and small-scale data migrations. A large-scale migration is typically upgrade scenario, like upgrading from Business Central version 14 to version 16. Small-scale migrations are scenarios where you want to move a limited number of objects from one extension to another.

Overview

The move is divided into two phases: development and deployment. However, before you begin, you have to determine the direction of the migration within the dependency graph.

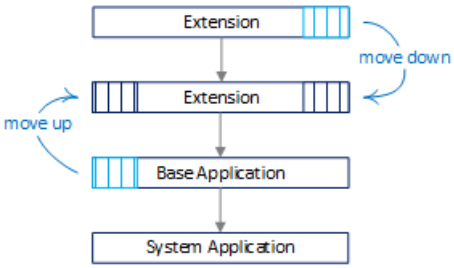
Limitations

- The concepts discussed in this article pertain only to AL. They're not supported in C/AL.
- You can only move fields from a table object to a table extension or move entire tables from one extension to another.
- You can't:
  - Move table extensions to other extensions.
  - Merge multiple table extensions into one extension.
  - Split table extensions into multiple extensions.

  Although you can't use this feature for these scenarios, you can achieve the scenarios by other means, which require more manual work. Examples include obsoleting, copying/re-IDing/renaming, moving data, and more.

Determining the migration direction in the dependency graph

The process to migrate tables and fields to another extension depends on the migration's direction in the dependency graph. The following figure illustrates a simplified extension dependency graph. From top to bottom, an extension is dependent on any extension below it in the graph.



When to move down

Typical move-down scenarios include:

- Moving to a shared extension.

  You want to move common tables to a separate extension that other extensions can have a dependency on.

- Transitioning from a customized base application extension with its own ID to an extension on top of the

Microsoft Base Application.

You have a customized base application extension with its own ID. You want to transition to the Microsoft Base Application. In this case, customizations remain in the current extension. Base objects are removed and ownership transferred to Microsoft Base Application.

This scenario is typical for embed ISV apps and on premises solutions moving to the cloud. It's also relevant for solutions that will remain on-premises for the foreseeable future. Use it to refactor code customizations into cleaner, standard base with extensions as part of upgrading.

For more information, see Moving Tables and Fields to Extension Down the Dependency Graph.

When to move up

Typical move-up scenarios include:

- Splitting an extension in two, with one dependent on the other.
- Extracting the system application from the base application.
- Transitioning from a customized base application extension with Microsoft ID to the Microsoft Base Application.

  You have a customized base application extension that reuses the Microsoft application ID. You want to transition to the standard Microsoft Base Application. In this case, customizations are moved out of the base application up into new extensions. The new extensions have new application IDs and dependencies to the standard Microsoft base application. The customization objects are removed from the custom base application and ownership transferred to the new extensions.

For more information, see Moving Tables and Fields to Extension Up the Dependency Graph.

Development

Development involves making application code changes required for the move. In short, the work involves:

- Creating the *releasing extension* version

  You create a new version of the original extension. This new version contains the tables or fields you want to keep in the extension. The tables and fields that you want to move are deleted from this extension. They're moved to the *receiving extension*.

- Creating the *receiving extension*

  You create new extension that includes the table and fields that you want moved. It essentially includes those tables and fields deleted from the releasing extension.

The key to the move is the *migration.json* file. You add the file to the project for the releasing extension. This file provides a pointer to the ID of new receiving extension where tables and fields are to be moved to. The *migration.json* is used in the deployment phase. Its purpose is to transfer ownership of tables and fields in the database from one extension to another. For more information, see Migration.json File.

Deployment

The deployment phase is when the data is migrated to new tables in the database. In this phase, ownership of tables and fields is switched from one extension to another. Deployment involves publishing, syncing, upgrading, and installing extensions.

The order that you synchronize extensions is important:

- The receiving extensions must be synchronized first.

- The releasing extensions, which include those extensions that include the migration.json file, must be synchronized last.

  These extensions are synchronized last because the tables are moved during the synchronization of the releasing extensions. At the end of the synchronization process, the system checks for breaking changes introduced by the extension. If the extension isn't synchronized last, breaking changes will be detected.

See Also

Publishing and Installing an Extension
JSON Files
Upgrading Extensions

Moving Tables and Fields to Extensions Down the Dependency Graph

2/17/2021 • 3 minutes to read • Edit Online

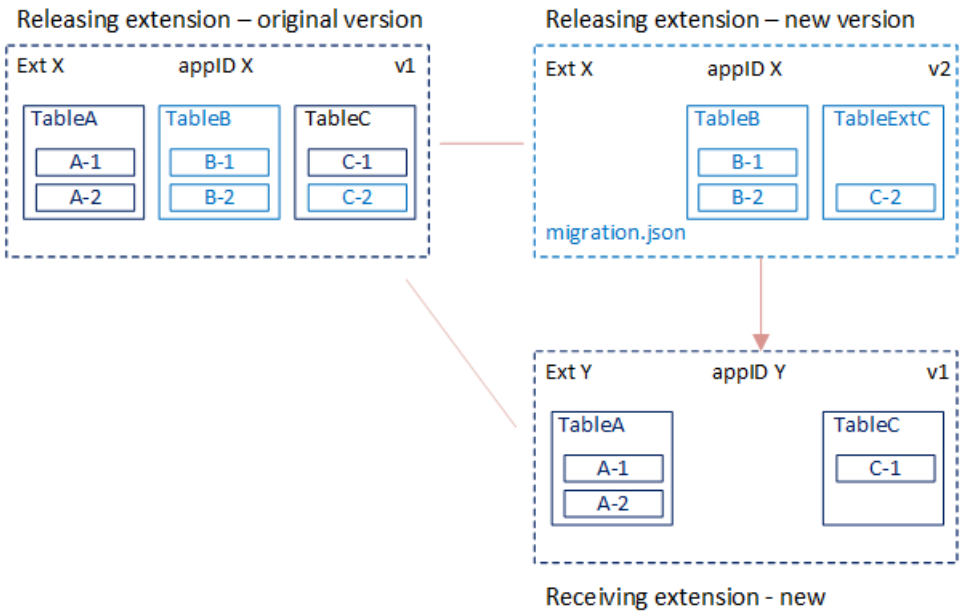
INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

This article explains how to move tables and fields from an extension to another extension that is down the dependency graph.

> **TIP**
> For more information about the general concepts for moving tables and fields between extensions, see Migrating Tables and Fields Between Extensions. This article explains the the difference between moving up and down the dependency graph.

Overview

The steps in this article are based on the example illustrated in the following figure. Although your scenario is different, the concept and process are much the same.



In the example, **TableB** and **Field C-2** are customizations. You'll keep these elements in the original extension, but create a new version without **TableA** and **TableC**. You'll move **TableA** and **TableC** down the dependency chain to a new, separate extension.

Prerequisite

If you're moving enum type fields, then your solution must be running on Business Central 2020 release wave 1, version 16.5 or later. For more information, see Known Issues.

Create receiving extension (Ext Y)

The receiving extension will contain the table and fields that you want to move. In this example, these objects

include **TableA** and **TableC**.

1. Create an AL project for the receiving extension.
2. Add a table object definition for **TableA**.

   The table definition (schema) must include the full schema of the releasing extension **Ext X**, with the same field definitions. You can add new fields.
3. Add a table object definition for **TableC**.

   The table definition (schema) must include the full schema of the releasing extension **Ext X**, with the same field definitions, except don't include field **C-2**. You can add new fields.
4. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `11111111-aaaa-2222-bbbb-333333333333`.
5. Compile the extension package.

Create new version of releasing extension (Ext X v2)

1. In the releasing extension AL project, add a migration.json file that points to the ID of the target extension.

   ```
   {
   "apprules": [
       {
           "id": "11111111-aaaa-2222-bbbb-333333333333"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.
2. Modify the app.json file as follows:

   - Ensure that `"target": "OnPrem"`.
   - Increase the `"version"` value.
   - In the `"dependencies"` parameter, set up a dependency on the new receiving extension **Ext Y**.

   For more information, see App.json file.
3. Complete the following steps for **TableC**.

   a. Add a table extension object **TableExtC**.
   b. In table extension object **TableExtC**, add a field definition for field **C-2** that matches its definition in the original **TableC** object.
   c. Delete the original **TableC** object.
4. Delete the entire table object for **TableA**.
5. Compile a new version of the extension package.

Deploy extensions

1. Uninstall the old version of the releasing extension **Ext X**.
2. Publish the new receiving extension **Ext Y** and releasing extension version **Ext X v2**.
3. Synchronize the receiving extension **Ext Y**.

This step creates empty database tables for **TableA** and **TableC** that are owned by the receiving extension **Ext Y**.

> **IMPORTANT**
>
> The receiving extension must always be synchronized first.

4. Synchronize the new version of the releasing extension **Ext X v2**.

   This step first reads rules in the migration.json file of the extension, then does the following operations in the database:

   - Creates a companion table for field **C-2** of the table extension object **TableExtC**.
   - Copies data from column **C-2** in the original **TableC** to new companion table **TableExtC**.
   - Temporarily renames the new empty tables **TableA** and **TableC** made by receiving extension **Ext Y**.
   - Renames the original tables **TableA** and **TableC** that include the data. Instead of including the ID of the releasing extension **Ext X**, the names are changed to include the ID of the receiving extension **Ext Y**. This step essentially transfers ownership from **Ext X** to **Ext Y**.
   - Deletes the unused column for **C-2** in the original table **TableC**.
   - Deletes the empty, renamed tables of **Ext Y**.
5. Install the receiving extension **Ext Y**.
6. Run Start-NAVAppDataUpgrade cmdlet on the new releasing extension version **Ext X v2**.

   This step basically installs the new extension version. You run a data upgrade because an earlier version has been installed and is still published.

See Also

Migrating Tables and Fields Between Extensions
Moving Tables and Fields to Extension Up the Dependency Graph
Upgrading Extensions
Publishing and Installing an Extension
JSON Files

Moving Tables and Fields to Extensions Up the Dependency Graph

2/17/2021 • 5 minutes to read • Edit Online

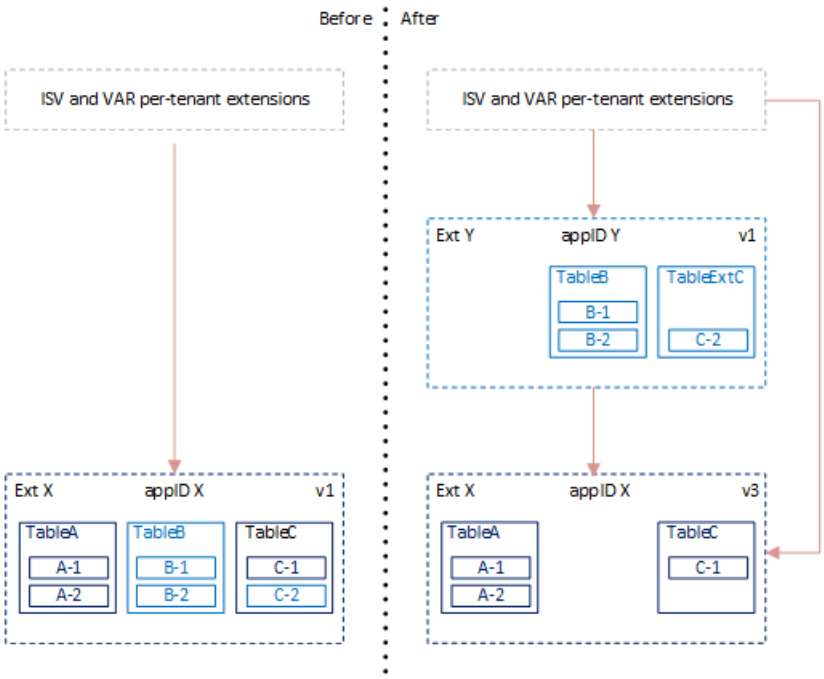
INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

This article explains how to move tables and fields from an extension to another extension that is up the dependency graph.

> **TIP**
> For more information about the general concepts for moving tables and fields between extensions, see Migrating Tables and Fields Between Extensions. This article explains the the difference between moving up and down the dependency graph.

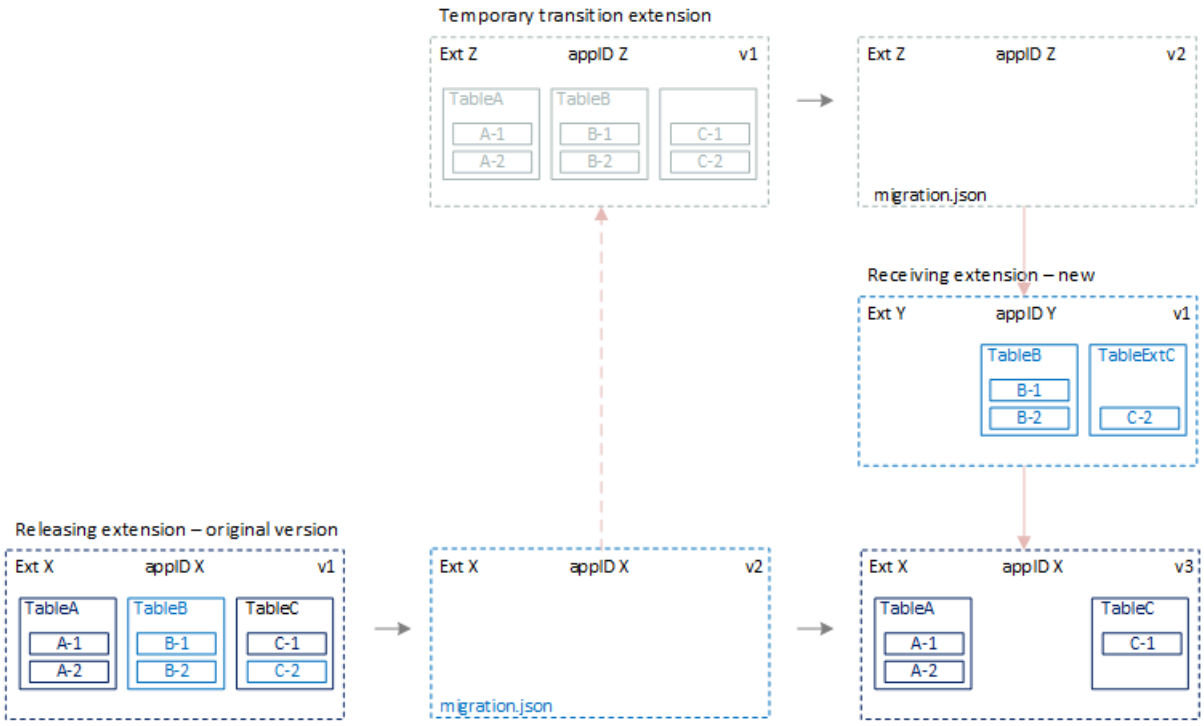
Overview

The steps are based on the example illustrated in the following figure. Although your scenario is different, the concept and process are much the same.



In the example, **TableB** and **Field C-2** are customizations. You'll move these elements from the original extension up to a new extension. This new extension will have a dependency on the original extension. You'll keep **TableA** and **TableC** in the original extension.

To accommodate data migration, you'll have to create an extension that is only used for deployment. This extension is **Ext Z** in the figure. There are two stages of deployment:



- In the first stage, **Ext Z** temporarily takes ownership of tables and fields from **Ext X**.
- In the second stage, **Ext Z** releases ownership to extensions **Ext X** and **Ext Y**. You uninstall and unpublish transition extension **Ext Z** when you finish deployment.

This process is a two-step process because we only support moving down the dependency graph. So instead, the concept is to first move the tables' ownership to an extension above the receiving extensions in the dependency graph. Then, the extensions are moved down. This concept essentially turns the process into a two-step, move-down process.

Ext Z is used just as a temporary extension for moving ownership. So, it only includes schema objects. As a result, customers can't run on the tenant until both steps have been done.

> **NOTE**
> You can only use this process for on-premise solutions.

Prerequisite

If you're moving enum type fields, then your solution must be running on Business Central 2020 release wave 1, version 16.5 or later. For more information, see Known Issues.

Create transition extension (Ext Z v1)

The transition extension will contain replicas of all table object definitions in the releasing extension, except logic code. In the illustration, these objects include **TableA**, **TableB**, and **TableC** and current their field definitions. The transition extension is **Ext Z**.

1. Create an AL project for the transition extension.
2. Add a table object that exactly matches the table object definitions for **TableA**, **TableB**, and **TableC** in the releasing extension.
3. Compile the extension package.
4. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

For purposes of the example, the ID is `11111111-aaaa-2222-bbbb-333333333333`. The value for your extension will be different.

Create empty version of releasing extension (Ext X v2)

In this step, you create a new version of the releasing extension that doesn't contain any objects. It only contains a `migration.json` file that points to **Ext Z**.

1. In the releasing extension AL project, add a migration.json file that points to the ID of the transition extension **Ext Z**.

   ```
   {
   "apprules": [
       {
           "id": "11111111-aaaa-2222-bbbb-333333333333"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.

2. Delete all objects from the extension. The objects include **TableA**, **TableB**, and **TableC**.
3. In the app.json file, increase the `version` value. Ensure that `"target": "OnPrem"`.
4. Compile a new version of the extension package.

Create receiving extension (Ext Y v1)

You now create a new extension that contains the customization you want to move from the releasing. In this example, the customizations include **TableB** and a **TableExtC**.

1. Create an AL project for **Ext Y**.
2. In the app.json file, set up a dependency on the releasing extension **Ext X**.
3. Add a table definition and code for **TableB** that exactly matches the definition in the original releasing extension.
4. Add a table extension object called **TableExtC**. Then, add a field definition for field **C-2** that matches its definition in the original **TableC** object of the releasing extension.
5. Compile the extension package.
6. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `44444444-cccc-5555-dddd-666666666666`. The value for your extension will be different.

Create final version of releasing extension (Ext X v3)

In this step, you create another version of the releasing extension **Ext X**. This version will contain the objects and code that you want to finally publish.

1. Create an AL project for **Ext X**.
2. Add a table definition and code for **TableA** that exactly matches the definition in the original releasing extension.
3. Add a table object for **TableC** and field definition for **C-1** that matches the definitions in the original

TableC object of the releasing extension.

4. In the app.json file, increase the `version` value.
5. Compile the extension package.
6. Make a note of the `ID` of the extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `77777777-eeee-8888-ffff-999999999999`. The value for your extension will be different.

Create new empty version of transition extension (Ext Z v2)

In this step, you create a new version of **Ext Z** that only contains a `migration.json` file. This `migration.json` file points the IDS of **Ext X** and **Ext Y**. The file is used to release ownership.

1. In the extension AL project, add a migration.json file that points to the IDs of the releasing extension **Ext X** and receiving extension **Ext Y**.

   ```
   {
   "apprules": [
       {
           "id": "77777777-eeee-8888-ffff-999999999999"
       }
       {
           "id": "44444444-cccc-5555-dddd-666666666666"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.

2. Delete the object definitions for **TableA**, **TableB**, and **TableC**.
3. In the app.json file, increase the `version` value.
4. Compile the extension package.

Deploy the extensions

1. Uninstall the current version of the releasing extension **Ext X**.
2. Complete the following steps for the first stage of deployment:

   a. Publish the transition extension **Ext Z** and empty version of **Ext X v2**.

   b. Synchronize the transition extension **Ext Z**.

      This step creates empty database tables **TableA**, **TableB**, and **TableC** that are owned by **Ext Z**.

      > **IMPORTANT**
      > Extensions receiving table objects must be synced first. Extension releasing/giving away table objects must be synced last.

   c. Synchronize the releasing extension **Ext X v2**.

      This step will read the migration.json of the extension. Then transfer ownership of the original tables **TableA**, **TableB**, and **TableC** to **Ext Z**.

3. Complete the following steps for the second stage of deployment:

    a. Publish the next version for **Ext Z v2** and **Ext X v3**, and the first version of **Ext Y**.

    b. Synchronize the extensions in the following order: **Ext X v3**, **Ext Y v1**, and **Ext Z v2**.

        Synchronize **Ext Z v2** last. When you synchronize **Ext Z v2**, ownership of the tables is transferred from **Ext Z** to **Ext X** and **Ext Y**.

4. Run Start-NAVAppDataUpgrade cmdlet on the new releasing extension version**Ext X v3**.

    This step basically installs the new extension version. You run a data upgrade because an earlier version has been installed and is still published.

5. Install the new receiving extension **Ext Y v1**.

6. Unpublish both versions of **Ext Z**.

See Also

Migrating Tables and Fields Between Extensions
Moving Tables and Fields to Extension Down the Dependency Graph
Upgrading Extensions
Publishing and Installing an Extension
JSON Files

DestinationAppsForMigration

2/17/2021 • 3 minutes to read • Edit Online

DestinationAppsForMigration is a Business Central Server instance setting. It's primarily used when upgrading from version 14. It's part of data upgrade process for transitioning application features from C/AL to AL extensions.

What this setting does

The DestinationAppsForMigration setting serves the following purposes:

1. Supports moving entire tables from C/AL to AL.

   When you transition your solution from C/AL to AL, you can move entire tables from the C/AL application to separate AL extensions. In the tenant database, the ownership of the tables must be transferred to the extensions. The DestinationAppsForMigration setting is used to do this transfer, which is done when the tenant is synchronized. As a result, in the database, the table names will be suffixed with the ID of the extension. For example, `[dbo].[CRONUS International Ltd_$VAT Clause]` changes to `[dbo].[CRONUS International Ltd_$VAT Clause$437dbf0e-84ff-417a-965d-ed2bb9650972]`. In this case, `437dbf0e-84ff-417a-965d-ed2bb9650972` is the extension's ID.

   To successfully move a table automatically: This scenario requires that:

   - Table ID in AL must be the same ID as was in the C/AL.
   - The table definition (schema) must match the existing schema. Or, if there are schema changes, they must be additive only.

     If this condition isn't met, it won't prohibit the move to the extension. But, the data upgrade may fail because of the breaking changes.

2. Runs upgrade code and installs extensions automatically during upgrade.

   The server instance essentially runs the Start-NAVDataUpgrade cmdlet. However, unlike normally, it doesn't invoke the OnInstall codeunit. It invokes the upgrade codeunit directly instead.

   When you run the data upgrade on a tenant, the server runs the upgrade code for the base and system application extensions. These extensions will then be automatically installed on the tenant.

3. Enables republishing and reinstalling extensions that haven't been built on the latest platform. For example, extensions that haven't been compiled on version 15 or 16.

   These extensions don't include base or system application dependencies in their manifests (app.json file). These extensions typically include the third-party extensions used on your version 14 solution. The DestinationAppsForMigration setting will resolve references to the Microsoft base application and system application for the extensions. When an extension is published, the system will automatically modify its manifest to include a dependency on the base and system applications.

   The destinationappsformigration process only inserts dependencies, which allows you to resolve dependencies. If the extension introduces breaking changes, it will fail.

TIP

If you moving a table and fields from one extension to another, this move is considered an AL to AL move. For this move, you use the migration.json file. For more information, see Migrating Tables and Fields Between Extensions.

How to set it up

You can only set the DestinationAppsForMigration by using the Set-NAVServerConfiguration cmdlet. To specify an extension, you must know the extension's ID, name, and publisher. The following code snippet shows the syntax for setting the DestinationAppsForMigration:

```
Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName "DestinationAppsForMigration" -
KeyValue '[{"appId":"<GUID>, "name":"<Extension name>", "publisher": "<Publisher>"},{"appId":"<GUID>,
"name":"<Extension name>", "publisher": "<Publisher>"}]'
```

For example, the following command specifies the Microsoft system and base applications and a custom extension named My Extension.

```
Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName "DestinationAppsForMigration" -
KeyValue '[{"appId":"63ca2fa4-4f03-4f2b-a480-172fef340d3f", "name":"System Application", "publisher":
"Microsoft"},{"appId":"437dbf0e-84ff-417a-965d-ed2bb9650972", "name":"Base Application", "publisher":
"Microsoft"},{"appId":"e3d1b010-7f32-4370-9d80-0cb7e304b6f0", "name":"My Extension", "publisher": "Me"}]'
```

IMPORTANT

The order of the extensions is important. You specify the extensions according to the dependency graph. For example, the system application must be before the base application.

FAQ

What extensions should I include?

In short, include any new extension, or in other words, any extensions that have never published and installed on the tenant. Typically:

- New extensions for existing objects that were moved from C/AL. The base application and system application are good examples.
- New extensions that add new objects to the application.
- Old extensions that are still compiled on version 14. Third-party extensions for which you don't have access to source code would typically fall into this category.

My extension only includes new objects that were never in C/AL. Can I include this extension?

Yes. You can use this mechanism to install any extension together with the Microsoft system or base application. For example, Microsoft publishes extensions for upgrade tests and the library assert that had no objects in the C/AL base application.

My extension is standalone, that is, it's not dependent on the system or base application. Can or should I include it?

It depends. For the setting to have any effect, you also include the system and base applications. If you use global triggers, you can add it to the list so it can listen to the triggers.

How many extensions can I include?

As many as you like. There's no limit or performance impact.

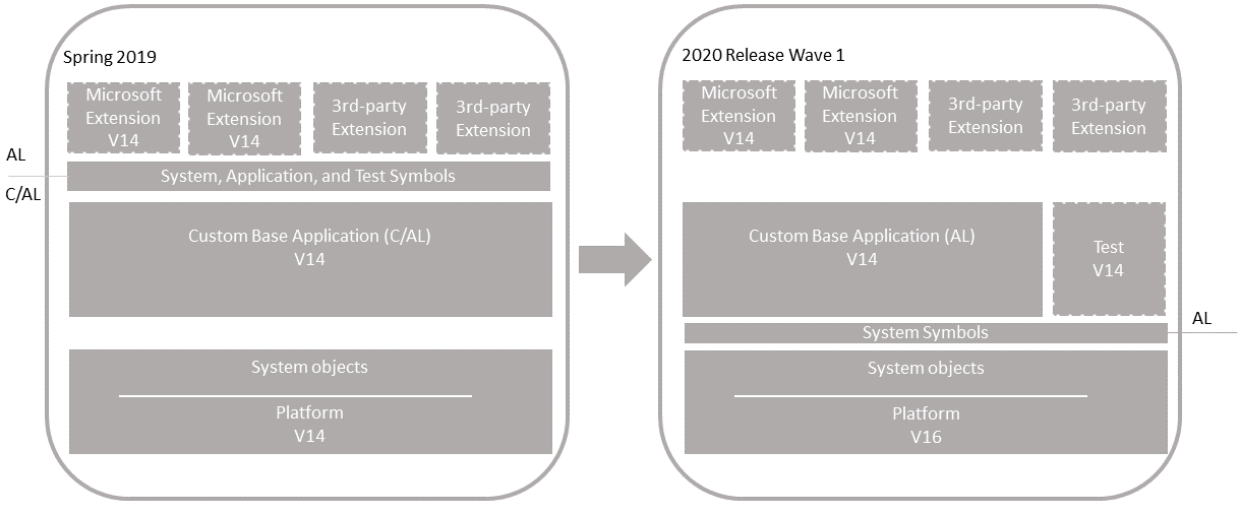
See Also

JSON Files
Migrating Tables and Fields Between Extensions

Technical Upgrade From Version 14 to Version 16

2/17/2021 • 12 minutes to read • Edit Online

Use this process when you have a code customized Business Central application (version 14) that you want to upgrade to the Business Central 2020 release wave 1 platform (version 16). This process won't upgrade the application to the latest version. You'll convert the entire application from C/AL to an AL base application extension.



Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Upgrade to Business Central Spring 2019 (version 14).

   There are several updates for version 14. When upgrading from Business Central Fall 2018 (version 13) or Dynamics NAV, upgrade to the latest version 14 update that has a compatible version 16 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   If your solution is already on version 14, then you don't have to upgrade to the latest version 16 update.

   To download the latest update, go to Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

   For information, see Upgrading to Dynamics 365 Business Central On-Premises.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 16

1. Before you install version 16, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 16.0 tools.

2. Install all components of version 16.

   You'll have to keep version 14.0 installed, because you'll need the it to complete the C/AL to AL conversion in this process. Therefore, when you install version 16, you must either specify different port numbers for components during installation or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

3. Copy the version 14 **CodeViewer** add-in to the version 16.0 server installation

   a. Find the **CodeViewer** folder in the **Add-ins** folder of the version 14 RoleTailored client installation. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client\Add-ins.

   b. Copy the folder to the **Add-ins** folder of the version 16 server installation. By default, the folder path is C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins. Replace the existing folder and files, if any.

   CodeViewer is no longer used in version 16. But it's required because of references that exist in the converted application. If you omit this step, you might get compilation errors later.

Task 2: Convert your v14 C/AL application to AL

For more information, see Code Conversion from C/AL to AL.

Task 3: Rewrite code for obsoleted system tables

In version 17, a number of tables have been deprecated and replaced by new tables compared to version 14. You must rewrite code that uses the deprecated tables to use the new tables. For a list of the deprecated tables and new tables, see Deprecated Tables.

Task 4: Prepare databases

In this task, you prepare the application and tenant databases for the upgrade.

1. Make backup of the database.

2. Make sure that you have the extension packages for all published extensions.

   You'll need these packages later to publish and install the extensions again.

3. Uninstall all extensions from the old tenants.

   Run the Business Central Administration Shell for version 14.0 as an administrator. Use the Uninstall-NAVApp cmdlet to uninstall an extension. For example, together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions with a single command:

```
Get-NAVAppInfo -ServerInstance <BC14 server instance> -Tenant <tenant ID> | % { Uninstall-NAVApp -
ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version -Tenant <tenant ID>}
```

If you have a single tenant deployment, you can omit the `-Tenant` parameter and value.

4. Unpublish all extensions from the application server instance.

   To unpublish extensions, use the Unpublish-NAVAPP cmdlet. Together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions from the tenant using a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
   server instance> -Name $_.Name -Version $_.Version }
   ```

5. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet. You can unpublish all symbols by using the Get-NAVAppInfo cmdlet with the `-SymbolsOnly` switch as follows:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

6. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <BC14 server instance> -Tenant <tenant ID>
   ```

7. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <BC14 server instance>
   ```

Task 5: Convert version 14.0 application database

This task runs a technical upgrade on the application database. A technical upgrade converts the database from the version 14.0 platform to the version 16.0 platform. This conversion updates the system tables of the database to the new schema (data structure). It also provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 16.0 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server>\<database instance> -
   DatabaseName "<BC14 database name>"
   ```

   When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (14-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 6: Configure version 16 server

When you installed version 16 in **Task 1**, a version 16 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 14 to version 16.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <BC15 server instance> -KeyName DatabaseName -KeyValue "
   <BC14 database name>"
   ```

   In a single tenant deployment, this command mounts the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <BC16 server instance> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <BC16 server instance>
   ```

Task 7: Publish system symbols, base application, and test library extensions

In this task, you'll publish extensions to the version 16.0 server instance. Publishing adds the extension to the application database that is mounted on the server instance. The extension is then available for installing on tenants later. It updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 16.0 that you started in the previous task.

1. Publish the system symbols extension for version 16.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default installation path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment.

   ```
   Publish-NAVApp -ServerInstance <BC16 server instance> -Path "<path to the System.app file>" -
   PackageType SymbolsOnly
   ```

2. Publish the custom base application extension that you created in **Task 2**.

```
Publish-NAVApp -ServerInstance <BC16 server instance> -Path "<path to the base application extension
package file>"
```

3. Publish the test library extension if you created one in **Task 2**.

```
Publish-NAVApp -ServerInstance <BC16 server instance> -Path "<path to the test library extension
package file>"
```

Task 8: Synchronize tenant

This task updates the tenant database schema with schema changes in system objects and application objects.

If you have a multitenant deployment, run these steps for each tenant (replacing `<tenant ID>` with the appropriate tenant ID).

1. (Multitenant only) Mount the tenant to the version 16 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

```
Mount-NAVTenant -ServerInstance <BC16 server instance> -DatabaseName "<BC14 database name>" -
DatabaseServer <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
```

> **IMPORTANT**
> You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you will get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

> **NOTE**
> For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

2. Synchronize the tenant to the system objects.

   Use the Sync-NAVTenant cmdlet:

```
Sync-NAVTenant -ServerInstance <BC16 server instance> -Tenant <tenant ID> -Mode Sync
```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   At this stage, the tenant state is **Operational**.

Task 9: Install base and test applications

You'll synchronize the tenant to the custom base application and test library extension (if any). When you synchronize the tenant, extensions take ownership of the tables in the SQL Database.

If you have a multitenant deployment, run these steps for each tenant (replacing `<tenant ID>` with the appropriate tenant ID).

1. Synchronize the tenant to the base application extension (Base Application).

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <BC16 server instance> -Name "Base Application" -Version <extension
   version> -tenant <tenant ID>
   ```

   With this step, the base app takes ownership of the database tables. When completed, in SQL Server, the table names will be suffixed with the base app extension ID.

2. Install custom base application extension on the tenant.

   To install the extension, you use the Install-NAVApp cmdlet.

   ```
   Install-NAVApp -ServerInstance <BC16 server instance> -Name "Base Application" -Version <extension
   version>
   ```

   At this point, the base application is upgraded to the version 16 platform and is operational. You can open the application in the client.

3. Synchronize and install the test library extension.

   This step is like what you did for the custom base application in steps 3 and 4.

Task 10: Configure version 16 server for app migration

In this task, you configure the version 16 server so that the Microsoft and third-party extensions that were installed in the version 14 deployment can be reinstalled. You'll configure the `DestinationAppsForMigration` parameter of the serve instance with information about the custom base application and test library. Specifically, you need the ID, name, and publisher assigned to these extensions. With the `DestinationAppsForMigration` parameter set, when you publish the Microsoft and third-party extensions, the server instance will automatically modify the manifest of the extensions to include the dependency on the base application and test library extension, allowing them to be published. For more information about this setting, see DestinationAppsForMigration.

1. Get the ID, name, and publisher of the custom base application and test library.

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance>
   ```

2. Set the `DestinationAppsForMigration` parameter for the server instance configuration to include the information about the custom base application and test library (if used). For example:

   ```
   Set-NAVServerConfiguration -ServerInstance <BC16 server instance> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"437dbf0e-84ff-417a-965d-ed2bb9650972",
   "name":"Base Application", "publisher": "Microsoft"},{"appId":"<test library extension app ID>",
   "name":"<test library extension name>", "publisher": "<test library publisher>"}]'
   ```

3. Restart the server instance.

Task 11: Publish and install extensions

Now, you can install the Microsoft and 3rd-party extensions that were installed on the tenant before the upgrade.

1. Publish the old extension versions.

```
Publish-NAVApp -ServerInstance <BC16 server instance> -Path "<path to extension package file>"
```

You only need to do this step once.

2. Synchronize the extension.

```
Sync-NAVApp -ServerInstance <BC16 server instance> -Name "<extension name>" -Version <extension
version> -tenant <tenant ID>
```

3. Install the extension:

```
Install-NAVApp -ServerInstance <BC16 server instance> -Name "<extension name>" -Version <extension
version> -tenant <tenant ID>
```

4. Repeat steps 2 and 3 for each extension and on each tenant.

Now, your application is fully upgraded to the version 16 platform.

Task 12: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is `C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`.

5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC160'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Task 13: Post-upgrade

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
4. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 16 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - If you have a version 16 application, export the **EXCEL EXPORT ACTION** permission set. Then, import it to your application and add it to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate

permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

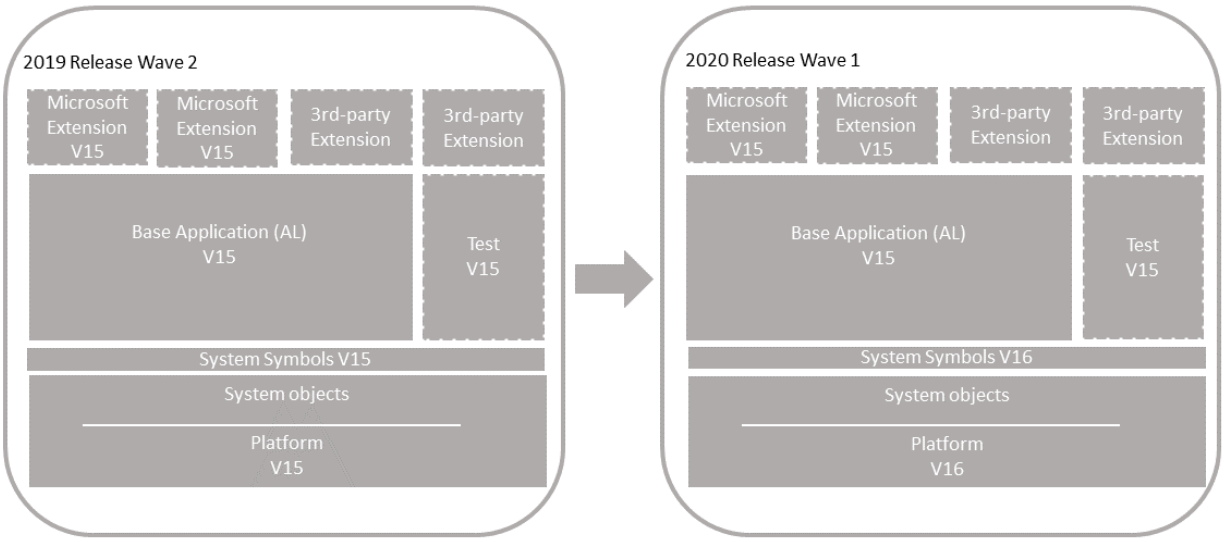
See Also

Upgrading to Business Central
Business Central 14.X to 15.X compatibility matrix

Technical Upgrade from Version 15 to Version 16

2/17/2021 • 9 minutes to read • Edit Online

Use this process to upgrade from Business Central 2019 release wave 2 (version 15) to the Business Central 2020 release wave 1 platform (version 16). This process won't upgrade the application to the latest version.



Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Your version 15 platform is compatible with version 16.

   There are several updates for version 15. The updates have a compatible version 16 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix. For example, if your solution is currently running 15.5, you can't upgrade to 16.0. You must wait until 16.1 is available.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 16

1. Before you install version 16, it can be useful to create desktop shortcuts to the version 15.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 16.0 tools.

2. Install version 16 components.

   You can keep version 15 component installed, but it is not required. If you do, when you install version 16, you must either specify different port numbers for components during installation or stop the version 15.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 2: Rewrite code for obsoleted system tables

In version 16, a number of tables have been deprecated and replaced by new tables. You must rewrite code that uses the deprecated tables to use the new tables. For a list of the deprecated tables and new tables, see Deprecated Tables.

Task 3: Prepare databases

In this task, you prepare the application and tenant databases for the upgrade.

1. Make backup of the database.
2. Make sure that you have the extension packages for all published extensions.

   You'll need these packages later to publish and install the extensions again.

3. (Single-tenant only) Uninstall all extensions from the old tenants.

   Run the Business Central Administration Shell for version 15.0 as an administrator. Use the Uninstall-NAVApp cmdlet to uninstall an extension. For example, together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance> | % { Uninstall-NAVApp -ServerInstance <BC15 server instance> -Name $_.Name -Version $_.Version }
   ```

4. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet. You can unpublish all symbols by using the Get-NAVAppInfo cmdlet with the `-SymbolsOnly` switch as follows:

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance> -SymbolsOnly | % { Unpublish-NAVApp -ServerInstance <BC15 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

5. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <BC15 server instance> -Tenant <tenant ID>
   ```

6. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <BC15 server instance>
   ```

Task 4: Convert the version 15.0 application database

This task runs a technical upgrade on the application database. A technical upgrade converts the database from the version 15.0 platform to the version 16.0 platform. This conversion updates the system tables of the database to the new schema (data structure). It also provides the latest platform features and performance enhancements.

> **IMPORTANT**
> The conversion does not modify the application objects, but it will remove any modifications that you have made to system tables. After the conversion you will no longer be able to use it with Business Central 14.

1. Start Business Central Administration Shell for version 16.0 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion. In a multitenant deployment, run this cmdlet against the application database.

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server>\<database instance> -
   DatabaseName "<BC15 database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (15-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 5: Configure version 16 server

When you installed version 16 in **Task 1**, a version 16 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 15 to version 16.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <BC15 server instance> -KeyName DatabaseName -KeyValue "
   <BC15 database name>"
   ```

   In a single tenant deployment, this command mounts the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <BC16 server instance> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <BC16 server instance>
   ```

Task 6: Publish new system symbols

Use the Publish-NAVApp cmdlet to publish the new symbols extension package. This package is called **System.app**. If you've installed the **AL Development Environment**, you find the file in the installation folder. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment.

```
Publish-NAVApp -ServerInstance <BC16 server instance> -Path "<path to the System.app file>" -PackageType
SymbolsOnly
```

Task 7: Recompile published extensions

Compile all published extensions against the new platform.

1. To compile an extension, use the Repair-NAVApp cmdlet, For example:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   To compile all published extensions at once, you can use this command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Task 8: Synchronize tenant

1. (Multitenant only) Mount the tenant to the new Business Central Server instance.

   You'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

2. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID> -Mode Sync
   ```

   For a single-tenant deployment, you can either set the `<tenant ID>` to `default` or omit the `-Tenant <tenant ID>` parameter. For more information about syncing, see Synchronizing the Tenant Database and Application Database.

Task 9: Reinstall extensions

Skip this task for a multitenant environment. In this task, you reinstall the same extensions that were installed on the tenant before.

To install an extension, you use the Install-NAVApp cmdlet.

1. If your solution uses the System Application, install this first.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version <extension
   version>
   ```

Replace `<extension version>` with the exact version of the published System Application.

2. Install the Base Application.

```
Install-NAVApp -ServerInstance <server instance> -Name "Base Application" -Version <extension
version>
```

   Replace `<extension version>` with the exact version of the published System Application.

3. (Upgrading from 15.3 and later) Install the Application extension.

   This extension was introduced in 15.3. For more information about it see The Microsoft_Application.app File.

```
Install-NAVApp -ServerInstance <server instance> -Name "Application" -Version <extension version>
```

   Replace `<extension version>` with the exact version of the published System Application.

4. Install other extensions, including Microsoft and third-party extensions.

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension
version>
```

At this point, your solution has been updated to the latest platform.

Task 10: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

```
C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-
ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
```

   .

5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC160'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Task 11: Post-upgrade

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
4. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 16 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - If you have a version 16 application, export the **EXCEL EXPORT ACTION** permission set. Then, import it to your application and add it to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate

permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

See Also

Upgrading to Business Central
Business Central 14.X to 15.X compatibility matrix

Installing a Business Central 2020 Release Wave 1 Update

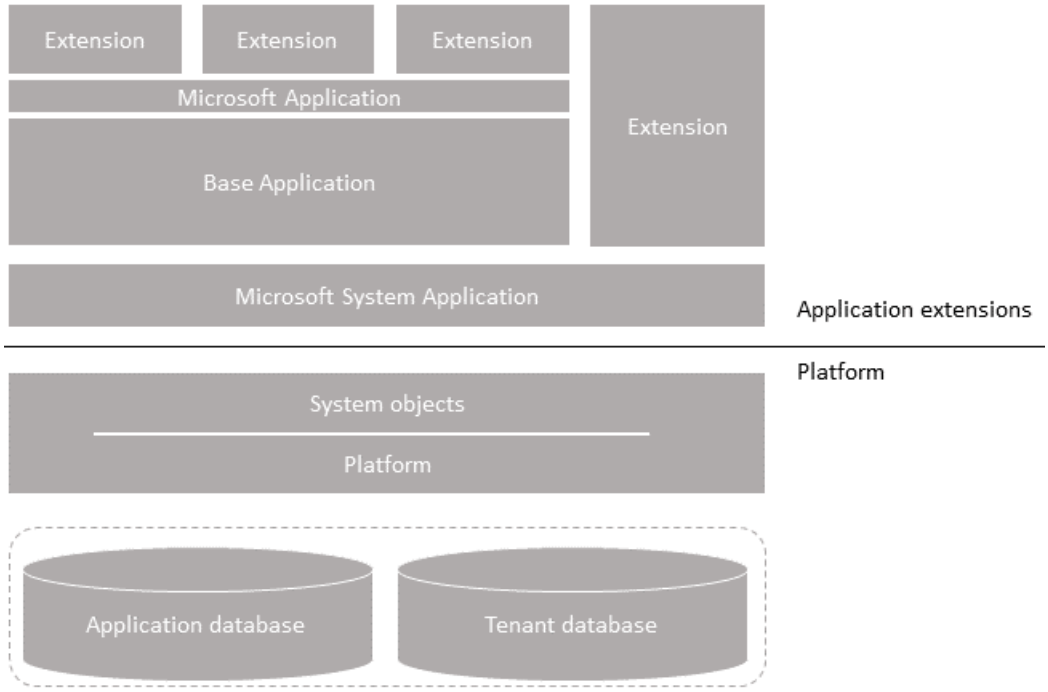
2/17/2021 • 17 minutes to read • Edit Online

This article describes how to install an update for Business Central on-premises. An update is a set of files that includes all hotfixes and regulatory features that have been released for Business Central.

You can choose to update only the platform or both the platform and application code. The installation guidelines are separated into PLATFORM tasks and APPLICATION tasks.

Overview

The following figure provides a high-level representation of a Business Central solution and the components that are involved in the installation of an update.



The databases store the application metadata and business data. If you have a single-tenant deployment, this data is stored in a single database. A multitenant deployment stores the application metadata in the application database and the business data in one or more tenant databases.

Application stack

The application includes AL extensions that define the objects and code that makes up the business logic. For example, objects include tables, report, pages, codeunits, and more. Each extension is compiled and delivered as an .app file, which is published to the Business Central Server instance.

- System Application extension

  The Microsoft System Application extension includes functionality that isn't directly related the business logic. For more information, see Overview of the System Application. When using the Microsoft Base Application, your solution uses the System Application. With a custom Base Application, your solution may or may not use the System Application. If it doesn't, you can skip any steps in this article related to the System Application.

- Base application extension

  As a minimum, the solution always includes the Base Application. The Base Application contains the objects (such as table, pages, codeunits, and reports) that define the business logic and functionality of the solution. The Base Application can be either the Microsoft Base Application or a customized Base Application. The Microsoft Base Application is the standard application that is on the installation media (DVD). A customized Base Application is an application that includes customized code.

- Application extension

  The Application extension logically encapsulates all of the extensions making up a solution, for example, version `16.0.0.0` of the base and system application package files, and it provides a convenient way to define and refer to this solution identity. This extension is required for Microsoft extensions, but is optional for third-party extensions. For more information, see The Microsoft_Application.app File.

- Customization extensions

  Customization extensions add functionality and features to the Base Application or System Application. Extensions can be either Microsoft extensions or third-party extensions. Microsoft extensions are available on the DVD. Third-party extensions are extensions developed by your organization or by another organization, like an ISV.

Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Platform versus application update

A platform update doesn't change the application. It involves converting your databases to the new platform and recompiling the existing extensions to ensure that they're compatible with the new platform.

An application update involves:

- Publishing new versions of extensions that include the latest application modifications
- Synchronizing the databases with any schema change introduced by the new extensions
- Updating affected data.

The installation media (DVD) includes new versions of Microsoft's Base Application, System Application, and extensions. The DVD also includes the AL source code for the Microsoft Base Application. This code useful if you have a custom base application. You can use the code to compare and merge updates into your application. You'll only have to recompile third-party extensions that you don't have a new version to publish.

PREPARATION

Download update package

The first thing to do is to download the update package that matches your Business Central solution.

1. Go to the list of available updates for your on-premises version of Business Central. Then, choose the update that you want.

2. From the update page, under the **Resolution** section, select the link for downloading the update, and follow the instructions.

3. On the computer where you downloaded the update .zip file, extract the all the files to a selected location.

   When extracted, the update includes the DVD folder. This folder contains the full Business Central product. For example, the folder includes the Business Central installation program (setup.exe), tools for upgrading to the platform, and the Microsoft extensions.

When this step is completed, you can continue to update your Business Central solution to the new platform and application.

Prepare existing databases

1. Back up your databases.
2. Run the Business Central Administration Shell as an administrator.
3. (Single-tenant only) Uninstall all extensions from the all tenants.

   In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name>
      ```

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extensions
      name> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the installed extension. For single-tenant deployment, set `<tenant ID>` to `default` or omit the `-Tenant` parameter.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID> | % { Uninstall-
      NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
      $_.Version -Force}
      ```

4. Unpublish the existing system symbols.

   To unpublish the system symbols, use the Unpublish-NAVApp cmdlet as follows:

   ```
   Unpublish-NAVApp -ServerInstance <server instance> -Name System -version <version>
   ```

   What are symbols?.

5. (Multitenant only) Dismount the tenants from the application database.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

```
Dismount-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID>
```

Install Business Central components

From the installation media (DVD), run setup.exe to uninstall the current Business Central components and install the Business Central components included in the update.

1. Stop the Business Central Server instance.

```
Stop-NAVServerInstance -ServerInstance <server instance>
```

2. Run setup.exe to uninstall your current version of Business Central.
3. Run setup.exe again to install components of the update.
   a. Follow setup pages until you get to the **Microsoft Dynamics 365 Business Central Setup** page.
   b. Select **Advanced installation options** > **Choose an installation option** > **Custom**.
   c. On the **Customize the installation** page, select the following components as a minimum:
      - AL Development Environment (optional but recommended)
      - Server
      - Web Server Components.
   d. Select **Next**.
   e. On the **Specify parameters** page, set the fields as needed.

      IMPORTANT
      Clear the **SQL Database** field so that it is blank. At this time, do not set this to the database that you want to update; otherwise, the installation of the Business Central Server will fail. You will connect the database to the Business Central Server later after it is converted to the new platform.

   f. Select **Apply** to complete the installation.

For more information, see Installing Business Central Using Setup.

PLATFORM UPDATE

Follow the next few tasks to convert your database to the new platform of the update. A multitenant deployment includes the application and tenant databases. The conversion updates the system tables of the database to the new schema (data structure) and provides the latest platform features and performance enhancements.

Also, to ensure that the existing published extensions work on the new platform, you'll recompile the extensions.

Convert existing database to new platform

1. Run the Business Central Administration Shell as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the database conversion to the new platform.

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
instance> -DatabaseName "<database name>" [-Force]
```

For example, in a single tenant deployment:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (16-
0)"
```

In a multitenant deployment, run this cmdlet against the application database and use the `-Force` parameter. For example:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer .\BCDEMO -DatabaseName "BC16 Application" -
Force
```

When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (16-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

NOTE

Depending on the update that you're installing, you might get a message similar to the following:

```
Invoke-NAVApplicationDatabaseConversion : A technical upgrade of database <database name> on
server '.\<database instance>' cannot be run, because the database's application version 'NNNNNN'
is greater than or equal to the platform version 'NNNNNN'
```

This is not an error, and you can continue installing the update. This message is recorded as a warning in the event log as well. This message indicates that the application database is already compatible with the new platform, which happens when the update does not make any schema changes to the system tables.

Connect server instance to database

1. (Multitenant only) Enable the server instance as a multitenant instance:

```
Set-NAVServerConfiguration -ServerInstance <server instance> -KeyName Multitenant -KeyValue true
```

2. Connect the server instance to connect to the database.

```
Set-NAVServerConfiguration -ServerInstance <server instance> -KeyName DatabaseName -KeyValue "
<database name>"
```

In a multitenant deployment, the database is the application database. For more information, see Connecting a Server Instance to a Database.

3. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance>
```

Publish the new system symbols

Use the Publish-NAVApp cmdlet to publish the new symbols extension package. This package is called **System.app**. If you've installed the **AL Development Environment**, you find the file in the installation folder. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\160\AL Development Environment. Or, it's also on the installation media (DVD) in the ModernDev\program files\Microsoft Dynamics NAV\160\AL Development Environment folder.

```
Publish-NAVApp -ServerInstance <server instance> -Path "<path to the System.app file>" -PackageType
SymbolsOnly
```

Recompile published extensions

Compile all published extensions against the new platform.

> **NOTE**
> If you plan on updating the application you can skip this step for extensions for which you have new versions built on the new platform. For example, this includes Microsoft extensions that are on the DVD.

1. To compile an extension, use the Repair-NAVApp cmdlet, For example:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   To compile all published extensions at once, you can use this command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Synchronize tenant

1. (Multitenant only) Mount the tenant to the new Business Central Server instance.

   You'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

2. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID> -Mode Sync
   ```

   For a single-tenant deployment, you can either set the `<tenant ID>` to `default` or omit the `-Tenant <tenant ID>` parameter. For more information about syncing, see Synchronizing the Tenant Database and Application Database.

NOTE

At this point, if you want to update the application, you can skip the next step and proceed APPLICATION.

(Single-tenant only) Reinstall extensions

In this task, you reinstall the same extensions that were installed on the tenant before. If you're planning on updating the application, then skip this step.

To install an extension, you use the Install-NAVApp cmdlet.

1. If your solution uses the System Application, install this first.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

2. Install the Base Application.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Base Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

3. Install the Application extension as needed.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Application extension.

4. Install other extensions, including Microsoft and third-party extensions.

   ```
   Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
   ```

At this point, your solution has been updated to the latest platform.

IMPORTANT

If your solution uses any Microsoft control add-ins, you must upgrade the add-ins to the latest version. Go to Upgrade control add-ins under **Post Upgrade** section.

APPLICATION UPDATE

Follow the next tasks to update the application code to the new features and hotfixes. The tasks include publishing new versions of the System Application, Base Application, and add-on extensions.

You publish the System Application extension only if it was used in old solution. Add-on extensions include Microsoft and third- party extensions that were used in the old solution.

> **NOTE**
> If a license update is required for a regulatory feature, customers can download an updated license from CustomerSource (see How to Download a Microsoft Dynamics 365 Business Central License from CustomerSource), and partners can download their customers' updated license from VOICE (see How to Download a Microsoft Dynamics 365 Business Central Customer License from VOICE).

Upgrade System Application

Follow these steps if your existing solution uses the Microsoft System Application. Otherwise, you can skip this procedure.

1. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
   Application.app>"
   ```

2. Synchronize the tenant(s) with the **System Application** extension (Microsoft_System Application):

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

   > **TIP**
   > To get a list of all published extensions, along with their names and versions, use the Get-NAVAppInfo cmdlet.

3. Run the data upgrade on the System Application.

   To run the data upgrade, use the Start-NavAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System
   Application" -Version <extension version>
   ```

   Upgrading data updates the data in the tables of the tenant database to the schema changes made to tables of the System Application.

Upgrade Base Application

Microsoft Base Application

Follow these steps if your existing solution uses the Microsoft Base Application.

1. Publish the Business Central Base Application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
Application.app>"
```

2. Synchronize the tenant with the Business Central Base Application extension (Microsoft_BaseApp):

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

   With this step, the base app takes ownership of the database tables. When completed, in SQL Server, the table names will be suffixed with the base app extension ID. This process can take several minutes.

3. Run the data upgrade on the Base Application.

   To run the data upgrade, use the Start-NavAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base
   Application" -Version <extension version>
   ```

   Upgrading data updates the data in the tables of the tenant database to the schema changes made to tables of the Base Application.

Upgrade custom Base Application

With a custom Base Application, you may want the new application features and hotfixes in the Microsoft Base Application. If so, you'll have to merge the modifications made in the Microsoft Base Application into your custom Base Application. Then, create a new version of your custom Base Application.

The source code for the new Microsoft Base Application version is in the **Base Application.Source.zip** file. This file is on the installation media (DVD), in the **Applications\BaseApp\Source** folder. You can compare this source code with the source code of the previous Microsoft Base Application and your custom application source. Then merge the code into a new custom application version.

After you've created the new version of your custom application, you publish it to the application server instance. Then, you synchronize and run the data upgrade on the tenants.

Upgrade Microsoft extensions

If your old solution used Microsoft extensions, then you upgrade these extensions to the new versions that are available on the Business Central installation media (DVD). The new versions are in the **Applications** folder, which contains a subfolder for each extension. The extension package (.app file) that you need for publishing the extension is in the **Source** folder, for example, **Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app**.

The general steps for this task are listed below. For detailed steps, see Publishing, Upgrading, and Installing Extensions During Upgrade.

Publish and install Microsoft_Application extension

The Microsoft_Application extension was introduced in 15.3. In short, it's used to contain the dependency declarations on the system and base application extensions. For more information about this extension, see The Microsoft_Application.app File.

1. Publish the Microsoft_Application.app package.

   The installation media path to the extension package (.app file) is

Applications\Application\Source\Microsoft_Application.app"

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<folder path>\Microsoft_Application.app"
```

2. Synchronize the tenant database with the schema changes of the Microsoft_Application extension.

```
Sync-NavApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension name> -Tenant
<tenant ID>
```

3. Upgrade the Microsoft_Application extension on your tenants.

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name Application
```

Publish and install Microsoft extensions

Complete these steps for each Microsoft extension that you want to upgrade.

1. Publish the new extension versions from the installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path <path to extension package file>
```

2. Synchronize the tenant database with the schema changes of the extensions.

```
Sync-NavApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension name> -
Version <extension version>
```

3. Upgrade the data associated with the Microsoft extensions. This step will automatically install the new version on the tenant

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension
name> -Version <extension version>
```

Upgrade Third-Party extensions

If the old solution used third-party extensions, and you still want to use them, they must be compiled to work on the new platform. For more information, see Recompile published extensions. You then reinstall the extensions on tenants using the Install-NAVApp cmdlet.

As an alternative, if you have the source for these extensions, you can build and compile a new version of the extension in the AL development environment. Then, you upgrade to the new version as described in the previous task.

Post Upgrade

Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins, do the following steps:

1. Open the Business Central client.

2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC160'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Update application version

This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version. The **Help and Support** page in the client displays an application version, such as 16.1.2345.6. This version number isn't updated automatically when you install an update.

However, the Business Central Server includes a configuration setting called **Solution Version Extension** (SolutionVersionExtension). This setting lets you specify an extension whose version number will show as the Application Version on the client's **Help and Support** page. Typically, you'd use the extension considered to be

your solution's base application. You set **Solution Version Extension** to ID of the extension. For example, if your solution uses the Microsoft Base Application, set the value to `437dbf0e-84ff-417a-965d-ed2bb9650972`.

You can set **Solution Version Extension** by using the Business Central Server Administration tool or the Set-NAVServerConfiguration cmdlet of the Business Central Administration Shell.

The following example uses the Set-NAVServerConfiguration cmdlet to set the **Solution Version Extension** to the Microsoft Base Application:

```
Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName SolutionVersionExtension -
KeyValue 437dbf0e-84ff-417a-965d-ed2bb9650972 -ApplyTo All
```

For more information about how to configure a server instance, see Configuring Business Central Server.

See Also

Dynamics 365 Business Central On-Premises Release Wave 2 Updates
Upgrading to Dynamics 365 Business Central 2019 Release Wave 2
Synchronizing the Tenant Database and Application Database
Version numbers in Business Central
Publish and Install an Extension
Getting Started in AL
Version numbers in Business Central

Upgrading to Dynamics 365 Business Central 2020 Release Wave 2

2/17/2021 • 2 minutes to read • Edit Online

Business Central 2020 release wave 2 (version 17) is the third major release that is fully AL-based. Business Central 2019 release wave 2 (version 15) marked the release in which C/AL was replaced by AL. The classic development environment, known as C/SIDE, was deprecated. From an application perspective, Business Central is now extension-based only. The Business Central base application is delivered as an AL extension. Also, application functionality that isn't related to the business logic has been moved into separate modules. These modules are combined into an extension known as the System Application. This change will influence how you do the upgrade compared to earlier releases.

Upgrade path

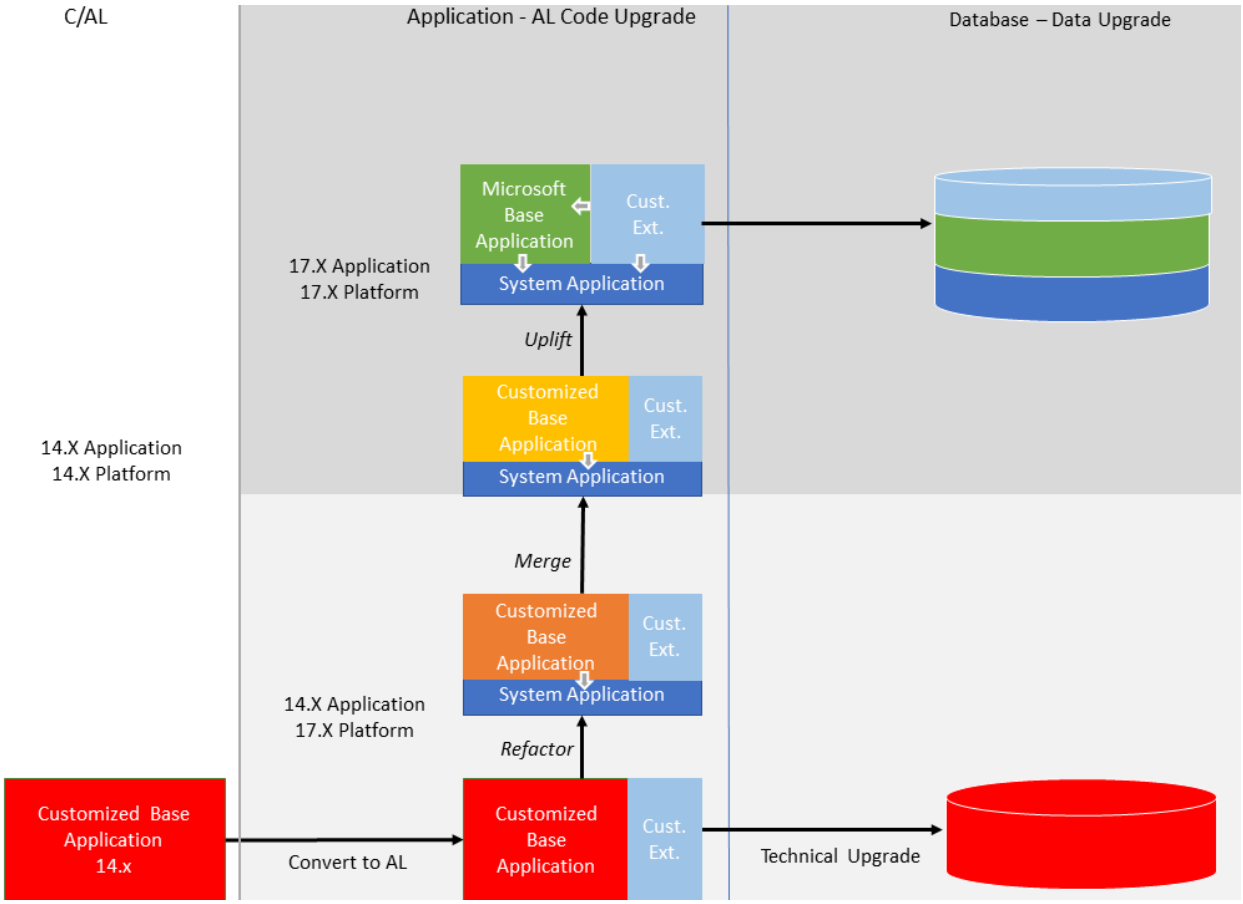
Depending on your current version, a direct upgrade to version 17 isn't always possible. You may have to first upgrade to an intermediate version. The following table describes the upgrade paths for supported versions:

SOURCE VERSION	PATH	COMMENT
• Microsoft Dynamics NAV 2015 (version 8) • Microsoft Dynamics NAV 2016 (version 9) • Microsoft Dynamics NAV 2017 (version 10) • Microsoft Dynamics NAV 2018 (version 11) • Business Central October 2018 (version 13)	1. Business Central Spring 2019 (version 14) 2. Business Central 2020 Release Wave 2 (version 17)	This path requires you convert your application from C/AL to AL.
• Business Central Spring 2019 (version 14) • Business Central 2019 Release Wave 2 (version 15) • Business Central 2020 release wave 1 (version 16)	Direct to version 17	

Your current version doesn't have to be the latest update for the version. However, for intermediate versions, use to the latest available update.

Upgrade overview

When upgrading your Business Central Spring 2019 (version 14) solution to version 17, the goal is to move towards a full uptake of the Business Central base and system applications, as they are, and migrating code customizations to add-on extensions. There are different upgrade levels that you follow to get to this state, as illustrated in the following figure. We recommend that you refactor to the system application as a minimum.



Technical Upgrade

Although it's recommend to refactor to the system application as a minimum, you can do a technical upgrade only. A technical upgrade changes the database so that it works on the latest Business Central platform. The conversion updates the system tables of the old database to the new schema (data structure). It provides you with the latest platform features and performance enhancements.

When upgrading from version 14, part of the technical upgrade process includes converting your customized base application from C/AL to AL.

New and changed features

There are several new and changed platform and application features available in Business Central 2020 release wave 2. These changes affect users, administrators, and developers. For an overview of these features, see Overview of Dynamics 365 Business Central 2020 release wave 2.

To take advantage of these features, you'll have to do an application code upgrade, not just a technical (platform) upgrade.

Deprecated features

Before you upgrade, review the following articles to get an overview of features deprecated in this release:

- Deprecated Tables
- Deprecated Features in W1

  From this article, use the links in the table of content to view deprecated features specific to local versions

Migrating from on-premises to online

For information about migrating an on-premises solution to online, see Migrate to Business Central Online from

Business Central On-premises.

See Also

Upgrading to Business Central
Upgrading Extensions
Dynamics 365 Business Central Upgrade Compatibility Matrix

Dynamics 365 Business Central Upgrade Compatibility Matrix

2/17/2021 • 2 minutes to read • Edit Online

You can upgrade Business Central Spring 2019 (version 14) directly to 2019 release wave 2 (version 15) or to 2020 release wave 1 (version 16). And, of course, you can upgrade 2019 release wave 2 directly to 2020 release wave 1 (version 16). For an overview of the supported paths, see Supported Upgrade Paths.

However, minor updates are regularly made available for each major release, like 14.1 or 15.2. When upgrading, it's important to target an update version that's compatible with your current version.

Before choosing the target version

Before you choose the target version for your upgrade, read the Some Known Issues article. This article will describe issues in Business Central versions that affect upgrade.

Version 14 compatibility

The following table lists the Business Central 14 versions and the minimum 15, 16, and 17 version that's compatible for upgrade.

VERSION 14	VERSION 15	VERSION 16	VERSION 17
14.0 (GA and cumulative update 01)	15.0	16.0	17.0
14.3 (cumulative update 02)	15.0	16.0	17.0
14.4 (cumulative update 03)	15.0	16.0	17.0
14.5 (cumulative update 04)	15.0	16.0	17.0
14.6 (cumulative update 05)	15.1	16.0	17.0
14.7 (cumulative update 06)	15.2	16.0	17.0
14.8 (cumulative update 07)	15.3	16.0	17.0
14.9 (cumulative update 08)	15.3	16.0	17.0
14.10 (cumulative update 09)	15.4	16.0	17.0
14.11 (cumulative update 10)	15.5	16.0	17.0
14.12 (cumulative update 11)	15.6	16.1	17.0

VERSION 14	VERSION 15	VERSION 16	VERSION 17
14.13 (cumulative update 12)	15.7	16.2	17.0
14.14 (cumulative update 13)	15.8	16.3	17.0
14.15 (cumulative update 14)	15.9	16.4	17.0
14.16 (cumulative update 15)	15.10	16.5	17.0
14.17 (cumulative update 16)	15.11	16.6	17.0
14.18 (cumulative update 17)	15.12	16.7	17.1
14.19 (cumulative update 18)	15.13	16.8	17.2
14.20 (cumulative update 19)	15.14	16.9	17.3
14.21 (cumulative update 20)	15.15	16.10	17.4
14.22 (cumulative update 21)[1]	15.16	16.11	17.5

For example, you can upgrade version 14.0 to any 15 or 16 version. You can only upgrade version 14.11 to version 15.5 (or later) or version 16.0 (or later).

To see the available updates for Business Central 2019 Release Wave 2, see Released Updates for Microsoft Dynamics 365 Business Central 2019 Release Wave 2 on-premises.

[1]The compatible upgrade versions for this version aren't available yet. If you are currently operating on this version, you'll have to wait until the next round of updates before you upgrade.

Version 15 compatibility

The following table lists the Business Central 15 versions and the minimum 16 and 17 version that's compatible for upgrade.

VERSION 15	VERSION 16	VERSION 17
15.0 to 15.4	16.0	17.0
15.5	16.1	17.0
15.6	16.2	17.0
15.7	16.3	17.0

VERSION 15	VERSION 16	VERSION 17
15.8	16.4	17.0
15.9	16.5	17.0
15.10	16.6	17.0
15.11	16.7	17.1
15.12	16.8	17.2
15.13	16.9	17.3
15.14	16.10	17.4
15.15[1]	16.11	17.5

To see the available updates for Business Central 2020 Release Wave 1, see Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 1 on-premises.

Version 16 compatibility

The following table lists the Business Central 16 versions and the minimum 17 version that's compatible for upgrade.

VERSION 16	VERSION 17
16.0 to 16.5	17.0
16.6	17.1
16.7	17.2
16.8	17.3
16.9	17.4
16.10[1]	17.5

To see the available updates for Business Central 2020 Release Wave 2, see Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

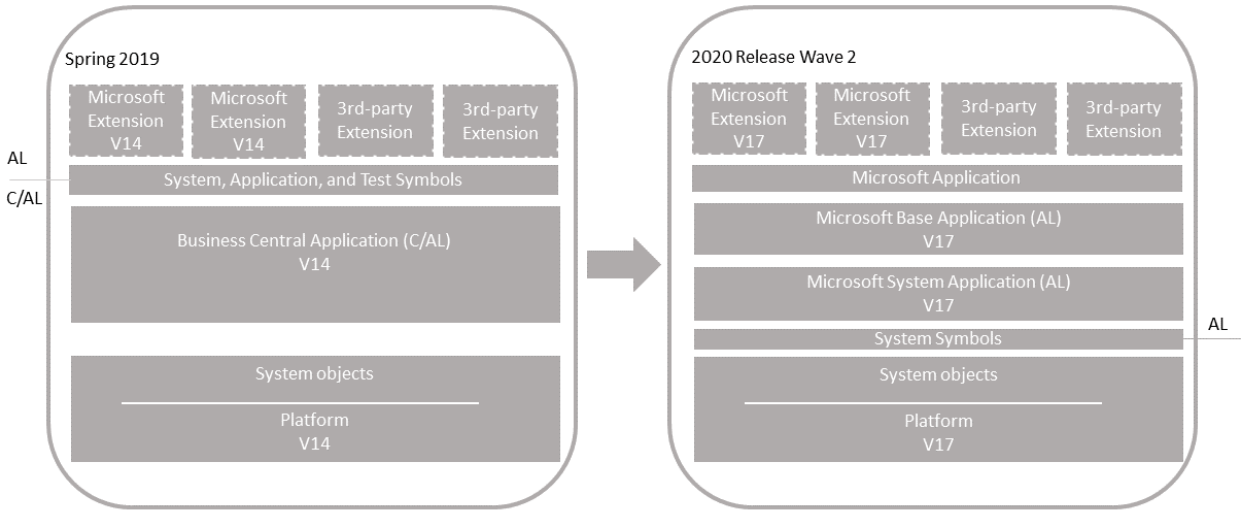
See Also

Upgrading to Business Central

Upgrading Unmodified C/AL Application to Version 17

2/17/2021 • 15 minutes to read • Edit Online

Use this scenario if you have a Business Central Spring 2019 (version 14) application or earlier that doesn't include any code customization. Your solution might include Microsoft (first party) extensions and customization extensions (3rd-party). With this upgrade, you'll replace the C/AL base application with the new Microsoft System and Base Application extensions. The result will be a fully upgraded Business Central 2020 release wave 1 (version 17) application and platform.



Single-tenant and multitenant deployments

The process for upgrading the similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application code and business data are in the same database. In a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisite

1. Upgrade to Business Central Spring 2019 (version 14).

   - If your solution is already on version 14, then no action on this step is required.
   - If you're upgrading from Business Central Fall 2018 (version 13) or Dynamics NAV, we recommend you upgrade to the latest update for version 14 that has a compatible update for version 17. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   To download the latest update, go to Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

   For information about how to do the upgrade, see Upgrading to Dynamics 365 Business Central On-Premises.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

For more information, see Managing Encryption and Encryption Keys.

Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 17

1. Download the latest available update for version 17 that is compatible with your version 14.

   To download the latest update, go to Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

   The guidelines in this article assume that you're running the latest available update.

2. Before you install version 17, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 17 tools.

3. Install Business Central version 17 components.

   You'll have to keep version 14 installed to complete some steps in the upgrade process. When you install version 17, you must either specify different port numbers for components (like the Business Central Server instance and web services) or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 2: Prepare version 14 databases

1. Make backup of the databases.

2. Start Business Central Administration Shell for version 14 as an administrator.

3. Uninstall all extensions from the old tenants.

   In this step, you uninstall any extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>
      ```

      For a single-tenant deployment, set the `<tenant ID>` to default.

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Name <extensions name> -Tenant
      <tenant ID> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the published System Application.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single

command:

```
Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-
NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
$_.Version -Force}
```

4. Unpublish all extensions from the application server instance.

   To unpublish an extension, use the Unpublish-NAVApp cmdlet:

   ```
   Unpublish-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension
   version>
   ```

   Together with the Get-NAVAppInfo cmdlet, you can unpublish all extensions by using a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
   server instance> -Name $_.Name -Version $_.Version }
   ```

5. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

6. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

7. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 3: Convert the version 14 database

This task runs a technical upgrade on the application database to convert it from the version 14 platform to the version 17 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 17 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

   When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (14-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 4: Configure version 17 server for DestinationAppsForMigration

When you installed version 17 in **Task 1**, a version 17 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 14 to version 17.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Configure the server instance for migrate extensions to the use the new base application and system application extensions.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"63ca2fa4-4f03-4f2b-a480-172fef340d3f",
   "name":"System Application", "publisher": "Microsoft"},{"appId":"437dbf0e-84ff-417a-965d-
   ed2bb9650972", "name":"Base Application", "publisher": "Microsoft"}]'
   ```

   This setting serves the following purposes:

   - When you run the data upgrade on a tenant, the server will run the data upgrade for the base and system application extensions. The base and system applications will be automatically installed on the tenant also.
   - Lets you republish extensions that haven't been built on version 17. The extensions typically include the third-party extensions that were used in your version 14. When you publish the extensions, the extension manifests are automatically modified with a dependency on the base and system applications.

   For more information about this setting, see DestinationAppsForMigration.

3. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

4. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 5: Import version 17 license

If you have a new Business Central partner license, make sure that it has been uploaded to the database.

1. To upload the license, use the Import-NAVServerLicense cmdlet:

```
Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path and file name>
```

2. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

For more information, see Uploading a License File for a Specific Database.

Task 6: Publish symbols and extensions

In this task, you'll publish the platform symbols and extensions. As minimum, you publish the new base application and system application extensions from the installation media (DVD). You also publish new versions of any Microsoft extensions and third-party extensions that were used on your old deployment.

Publishing an extension adds the extension to the application database that is mounted on the server instance. Once published, it's available for installing on tenants. This task updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 14 that you started in the previous task.

1. Publish version 17 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

```
Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
SymbolsOnly
```

   What are symbols?

2. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
Application.app>"
```

   What is the System Application?

3. Publish the Business Central base application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the (Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
Application.app>"
```

4. Publish the Microsoft_Application extension

   The Microsoft_Application extension is a new extension introduced in 15.3. For more information about this extension, see The Microsoft_Application.app File.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<folder path>\Microsoft_Application.app"
   ```

5. Publish the new versions of Microsoft extensions.

   In this step, you publish new versions of Microsoft extensions that were used on your version 14 deployment. You find the extensions in the **Applications** folder of the installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft extension>"
   ```

   For example:

   ```
   Publish-NAVApp -ServerInstance BC170 -Path
   "C:\W1DVD\Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app"
   ```

6. Publish 3rd-party extensions.

   Publish 3rd-party extensions that were used on your version 14 solution. If you have new versions of these extensions, built on the Business Central version 17, then publish the new versions. Otherwise, republish the same versions that were previously published in the old deployment.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to extension>"
   ```

Task 7: Restart server instance

Restart the Business Central Server to free up resources for completing the upgrade.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

This step is important, otherwise you might experience issues when you run the data upgrade.

Task 8: Synchronize tenant

In this task, you'll synchronize the tenant's database schema with any schema changes in the application database and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 17 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
   <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

> **IMPORTANT**
>
> You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

> **NOTE**
>
> For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the **System Application** extension.

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application. To get the version, you can use the Get-NAVAppInfo cmdlet.

4. Synchronize the tenant with the **Base Application** extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

5. Synchronize the tenant with the **Application** extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
   ```

6. Synchronize the tenant with Microsoft and 3rd-party extensions.

   For each extension, run the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant default -Name "<extension name>" -Version
   <extension version>
   ```

TIP

When you synchronize an extension, the extension takes ownership of any tables that it includes. In SQL Server, you'll notice that the table names will be suffixed with the extension ID. For example, Base Application tables will have `437dbf0e-84ff-417a-965d-ed2bb9650972` in the name. In addition, the systemId column is added to application tables that are not already part of an extension.

Task 9: Upgrade data

In this task, you run a data upgrade on tables to handle data changes made by platform and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. Upgrade the data to the platform, system application, and base application.

   a. To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -
   FunctionExecutionMode Serial -SkipAppVersionCheck
   ```

   b. To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

   This step will automatically install the base application and system application on the tenant.

2. Upgrade the new versions of Microsoft extensions and third-party extensions.

   Complete this task to upgrade any Microsoft extension and third-party extension. Microsoft extensions used in the old deployment to new versions on the installation media. The new versions are in the **Application** folder of the DVD. There's a folder for each extension. The extension package (.app file) is in the **Source** folder.

   a. Install **Application** extension.

   You'll have to install the **Application** extension first, otherwise you can't upgrade Microsoft extensions.

   ```
   Install-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
   ```

   b. For each extension, run Start-NAVAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Name "<extension name>" -
   Version <extension version>
   ```

   This step will also automatically install the new extension version on the tenant.

3. (Multitenant only) Repeat steps 1 through 3 for each tenant.

Task 10: Install 3rd-party extensions

Complete this task to install third-party extensions for which a new version wasn't published. For each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
```

Task 11: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

At this point, the upgrade is complete, and you can open the client.

Post-upgrade tasks

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
4. Change application version.

   (Optional) This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version.

   On the **Help and Support** page in the client, you'll see an application version, such as 14.0.2345.6. For an explanation of the number, see Version numbers in Business Central. This version isn't updated automatically when you install an update. If you want the version to reflect the version of the update or your own version, you change it manually.

   We recommend setting the value to application build number for the version 17 update. You get the number from the Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

   a. Run the Set-NAVApplication cmdlet:

   ```
   Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
   version> -Force
   ```

   For example:

   ```
   Set-NAVApplication -ServerInstance BC170 -ApplicationVersion 17.0.38071.0 -Force
   ```

   b. Run the Sync-NAVTenant cmdlet to synchronize the tenant with the application database.

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   c. Run the Start-NavDataUpgrade cmdlet to change the version number:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -FunctionExecutionMode Serial -Tenant
   <tenant ID>
   ```
5. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 17 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

- Assign the **EXCEL EXPORT ACTION** permission set to appropriate users.
- Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

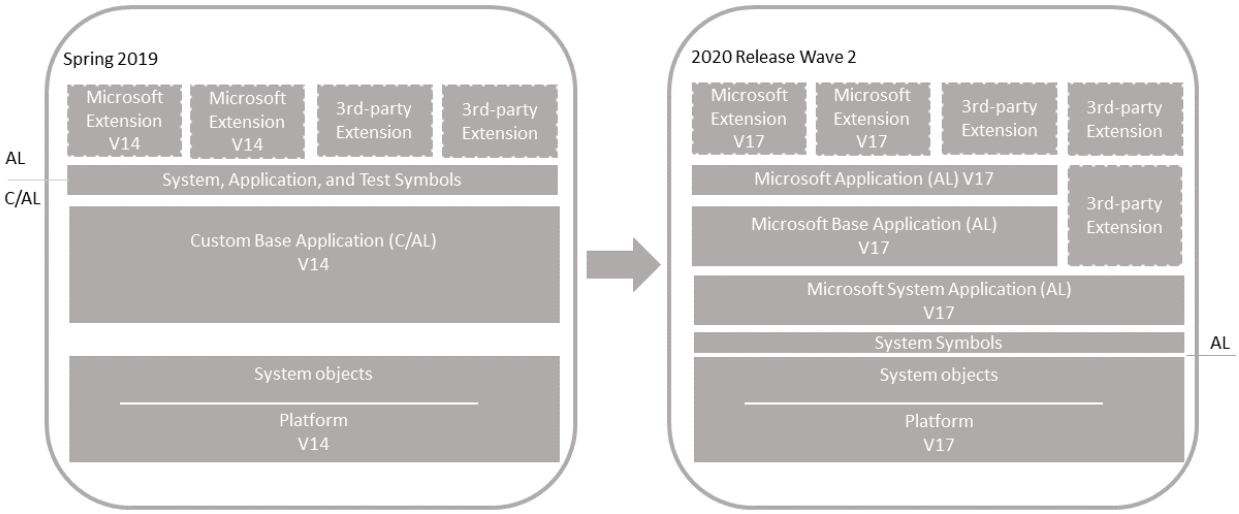
See Also

Publishing and Installing an Extension
Upgrading to Business Central
Upgrading Extensions

Upgrading Customized C/AL Application to Microsoft Base Application Version 17

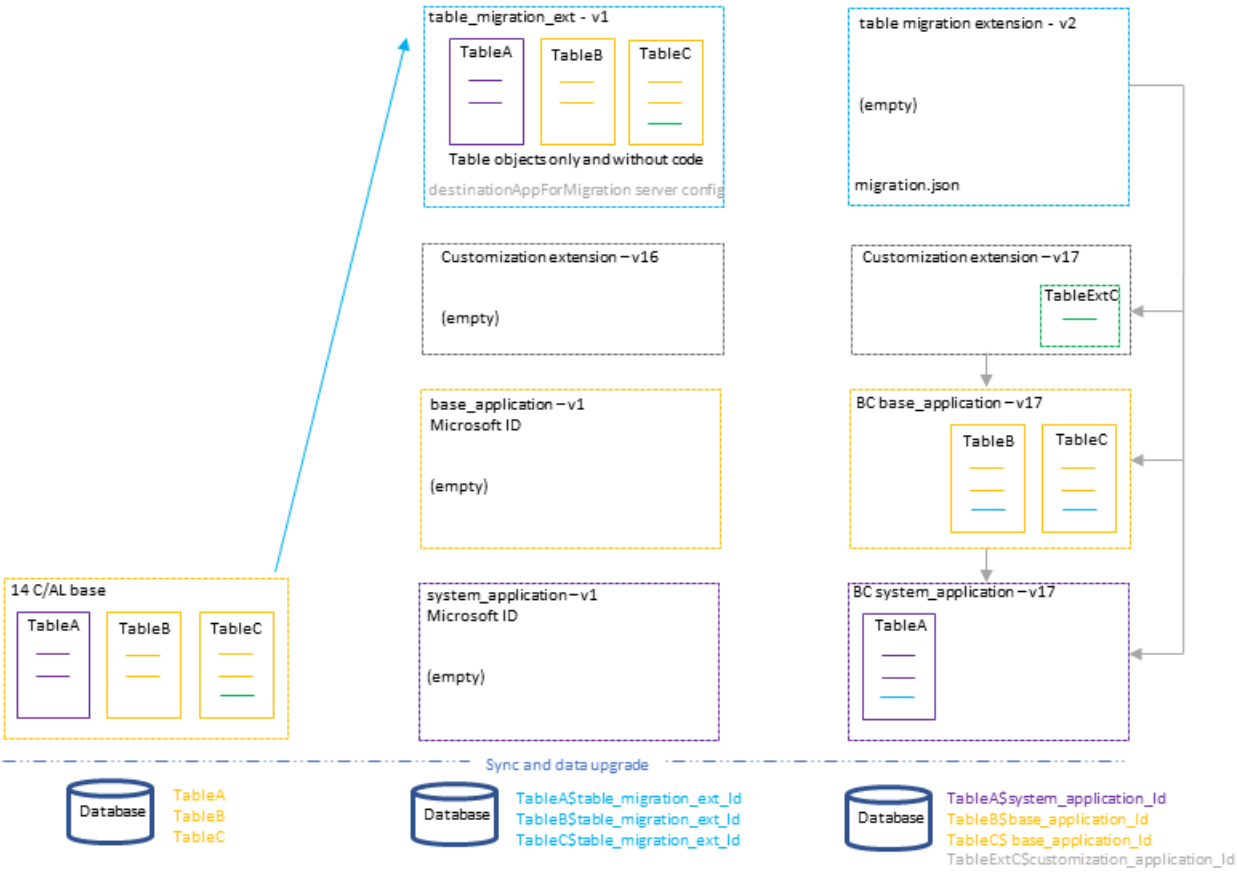
2/17/2021 • 20 minutes to read • Edit Online

This article describes how to upgrade a customized version 14 application to a version 17 solution that uses the Microsoft Base Application.



Overview

The upgrade is divided into two sections: Application Upgrade and Data Upgrade. The Application Upgrade section deals with upgrading the application code. For the application upgrade, you'll have to create several extensions. Some of these extensions are only used for upgrade purposes. The Data Upgrade section deals with upgrading the data on tenants - publishing, syncing, and installing extensions. For this scenario, the data upgrade consists of two phases for migrating data from the current tables to extension-based tables. The following figure illustrates the upgrade process.



The process uses two special features for migrating tables and data to extensions:

- destinationappsformigration server setting

  The *destinationappsformigration* setting is a configuration setting on the Business Central Server. In short, it's used to transfer ownership of the existing tables to the table migration extension. For more information, see DestinationAppsForMigration.

- migration.json file

  The *migration.json* file is used to migrate tables and fields from one extension to another. In this case, migration is from the table migration extension to system and base application tables. For more information about the migration.json, see The Migration.json File.

Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Your version 14 is compatible with version 17.

   There are several updates for version 14. The updates have a compatible version 17 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

2. The version 14 Dynamics NAV Development Shell and Business Central Administration Shell are installed.

3. Get the required version of the txt2al conversion tool.

   During the upgrade, you'll use the txt2al conversion tool to convert existing tables to the AL syntax. You'll

need to use a version of txt2al conversion tool that supports the `--tableDataOnly` parameter. This parameter was first introduced in version 14.12 (cumulative update 11, platform 14.0.41862). So if you're upgrading from version 14.11 (cumulative update 10) or earlier, you'll have to download the txt2al conversion tool from a later version 14 update. See Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

Install version 17

1. Download the latest available update for Dynamics 365 Business Central (version 17) that is compatible with your version 14.

   For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

2. Before you install version 17, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 17 tools.

3. Install Business Central version 17 components.

   You'll have to keep version 14 installed to complete some steps in the upgrade process. When you install version 17, you must either specify different port numbers for components (like the Business Central Server instance and web services) or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

APPLICATION UPGRADE

This section describes how to upgrade the application code. This work involves creating various extensions.

Task 1: Move code customizations to extensions

The first step, and the largest step, is to create extensions for the customizations compared to the Microsoft base application.

- Create extensions for the target platform **6.0 Business Central 2020 release wave 2**.
- Include dependencies for the Microsoft System, Base, and Application extensions for version 17.0.0.0.

Task 2: Create table migration extension

In this step, you create an extension that consists only of the non-system table objects from your custom base application. The table objects will only include the properties and field definitions. They won't include AL code on triggers or methods. This extension is an interim extension used only during the upgrade.

You'll create two versions of this extension. The first version contains the table objects. The second version, is an empty extension that contains a migrate.json file.

Create the first version

1. Create a folder where you'll store exported txt files for tables (for example, C:\export2al\bc14tablesonly).

2. Start the Dynamics NAV Development Shell for version 14.

3. Run the Export-NAVApplicationObject cmdlet to export only tables from the database.

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "C:\export2al\bc14tablesonly\exportedbc14-tables.txt" -Filter
'Type=Table;Id=1..1999999999'
```

4. Use the txt2al conversion tool to convert the exported tables to the AL syntax. Use the `--tableDataOnly` parameter to include table and field definitions only.

   a. Create a folder for storing the AL files for base application objects (for example, C:\export2al\bc14tablesonly\al).

   b. Start a command prompt as an administrator, and navigate to the folder that contains txt2al.exe file.

      By default, the location is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client.

   c. Run the txt2al command:

```
txt2al --source=C:\export2al\bc14tablesonly --target=C:\export2al\bc14tablesonly\al --tableDataOnly
```

   For more information about this tool, see The Txt2Al Conversion Tool.

   > **NOTE**
   > If the `--tableDataOnly` parameter isn't available, you'll need a later version ot the txt2al conversion tool. For more information, see Prerequisites.

5. Make sure you've installed the latest AL Extension for Visual Studio Code from the version 17 DVD.

   For more information, see Getting Started with AL.

6. In Visual Studio Code, create an AL project for table migration extension using the **AL: Go!** command.

   Set the target platform to **6.0 Business Central 2020 release wave 2**.

7. Configure the project's app.json file:

   - Set the `"name"`, `"publisher"`, and `"version"`. You can use any valid values.
   - Delete the `"application"` parameter.
   - Clear the `"dependencies"`.
   - Set the `"idRanges"` to cover the table object IDs or clear all values.
   - Add the `"target"` parameter and set it to `"Onprem"`.

   Make a note of the `"id"` setting value, which is the ID assigned to the table migration extension. You'll use this ID later in the process. **Don't** use the ID in the following example. It's for illustration purposes only.

```
{
  "id": "11111111-aaaa-2222-bbbb-333333333333",
  "name": "bc14tablesonly",
  "publisher": "My publisher",
  "version": "1.0.0.0",
  "brief": "",
  "description": "",
  "privacyStatement": "",
  "EULA": "",
  "help": "",
  "url": "",
  "logo": "",
  "dependencies": [],
  "screenshots": [],
  "platform": "17.0.0.0",
  "idRanges": [  ],
  "contextSensitiveHelpUrl": "https://bc14tablesonly.com/help/",
  "showMyCode": true,
  "runtime": "6.0",
  "target": "OnPrem"
}
```

8. Create an `.alpackages` folder in the root folder of the project and then copy the version 17 system symbols extension (System.app file) to the folder.

   The System.app file is located where you installed the AL Development Environment. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment. This package contains the symbols for all the system tables and codeunits.

9. Add the AL files for tha tables that you converted earlier to the root folder for the project.

   The folder used in this article is C:\export2al\bc14tablesonly\al.

   > **IMPORTANT**
   > If you're upgrading a **CH** or **NA** local version (14.18 or later), you'll have to rename the primary keys in some tables. For more information, see Known Issues.

10. Build the extension package for the first version.

    To build the extension package, press Ctrl+Shift+B. This step creates an .app file for your extension. The file name has the format <publisher>_<name>_<version>.app.

Create the second version

1. In Visual Studio Code, create a new file called migration.json file and add it to the project's root folder.
2. In the migration.json, include rules for the Microsoft base application, system application, and your customization extensions.

```
{
    "apprules": [
        {
            "id": "63ca2fa4-4f03-4f2b-a480-172fef340d3f"
        },
        {
            "id": "437dbf0e-84ff-417a-965d-ed2bb9650972"
        },
        {
            "id": "<NNNNNNNN-NNNN-NNNN-NNNN-NNNNNNNNNNNN>"
        },
        {
            "id": "<NNNNNNNN-NNNN-NNNN-NNNN-NNNNNNNNNNNN>"
        }
    ]
}
```

In the example code:

- `63ca2fa4-4f03-4f2b-a480-172fef340d3f` identifies the system application extension
- `437dbf0e-84ff-417a-965d-ed2bb9650972` identifies the base application extension
- The two other IDs are examples that identify other new customization extensions you might have. Replace or remove these entries as needed.

For more information about the migration.json, see The Migration.json File.

3. Delete the AL object files.
4. Increase the `version` in the app.json file.
5. Build the extension package for the second version.

   To build the extension package, press Ctrl+Shift+B.

Task 3: Create empty extensions System, Base, and customization extensions

Create two empty extensions: one for the Microsoft base application and another for the System Application. Also, create an empty extension for each new customization extension. The only file in the extension project that is required is an app.json.

You can create the empty extension like any other extension by adding an AL project in Visual Studio Code:

1. In Visual Studio Code, select **View** > **Command Palette** > **AL: Go!** and follow instructions.
2. Delete the HelloWorld.al sample file from the project.
3. Modify the app.json file.

   The important settings in the app.json file are: `"id"`, `"name"`, `"version"`, `"publisher"`, `"dependencies"`, and `"runtime"`.

   - The `id` and `name` must match the value used by Microsoft's extensions.
   - Set the `version` to any version lower than 17.0.0.0, like 14.0.0.0.
   - You'll also have to include the `"publisher"`. You can use your own publisher name or `"Microsoft"`.
   - Remove all other settings. It's important that there are no `"dependencies"` set.
   - Set the `runtime` to `"6.0"`.

   The app.json files for the **System Application** and **Base Application** extensions, should look similar to following examples:

System Application

```
"id": "63ca2fa4-4f03-4f2b-a480-172fef340d3f",
"name": "System Application",
"publisher": "Microsoft",
"version": "14.0.0.0",
"runtime": "6.0"
```

Base Application

```
"id": "437dbf0e-84ff-417a-965d-ed2bb9650972",
"name": "Base Application",
"publisher": "Microsoft",
"version": "14.0.0.0",
"runtime": "6.0"
```

4. Build and compile the extension package. To build the extension package, press Ctrl+Shift+B.

> **TIP**
> This step is only required if you need to trigger a data upgrade on these extensions, which you'll do by running Start-NavAppDataUpgrade on these extensions in Task 14. For the scenario in this article, at a minimum this step is required for the System and Base Applications. You can skip this step for any customization extensions that do not not include upgrade code.

DATA UPGRADE

Task 4: Prepare databases

1. Make backup of the databases.
2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.
3. Start Business Central Administration Shell for version 14 as an administrator.
4. (Single-tenant only) Uninstall all extensions from the tenants.

   To uninstall an extension, you use the Uninstall-NAVApp cmdlet. For example, you can uninstall all extensions with a single command:

```
Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-NAVApp -
ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version $_.Version -Force}
```

5. (Single-tenant only) Unpublish all extensions from the application server instance.

   To unpublish an extension, use the Unpublish-NAVApp cmdlet. For example, you can unpublish all extensions by using a single command:

```
Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
server instance> -Name $_.Name -Version $_.Version }
```

6. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

7. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

8. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 5: Convert version 14 database

This task runs a technical upgrade on the application database. The task converts the database from the version 14 platform to the version 17 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 17 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (14-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 6: Configure version 17 server for DestinationAppsForMigration

In this step, you configure the version 17 server instance. In particular, you configure it to migrate the table migration extension that you created earlier. The migration is controlled by the `DestinationAppsForMigration` setting for the server instance. For more information about the `DestinationAppsForMigration` setting, see DestinationAppsForMigration.

1. Set the server instance to connect to the application database.

```
Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
<database name>"
```

In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Configure the `DestinationAppsForMigration` setting of the server instance to table migration extension.

   You'll need the ID, name, and publisher for the table migration extension that you created in **Task 2**.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName
   "DestinationAppsForMigration" -KeyValue '[{"appId":"<table migration extension ID>", "name":"<table
   migration extension>", "publisher": "<publisher>"}]'
   ```

   For example:

   ```
   Set-NAVServerConfiguration -ServerInstance BC -KeyName "DestinationAppsForMigration" -KeyValue
   '[{"appId":"11111111-aaaa-2222-bbbb-333333333333", "name":"bc14tablesonly", "publisher": "My
   publisher"}]'
   ```

   > **NOTE**
   > You can add multiple extensions to this setting.

3. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

4. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 7: Import License

Import the version 17 partner license. To import the license, use the Import-NAVServerLicense cmdlet:

```
Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path>
```

Restart the server instance.

Task 8: Publish symbols and DestinationAppsForMigrations

In this task, you'll publish the platform symbols and the extensions configured as DestinationAppsForMigration.

1. Publish version 17 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business

Central\170\AL Development Environment.

```
Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
SymbolsOnly
```

What are symbols?

2. Publish the first version of the table migration extension, which is the version that contains the table objects.

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to extension .app file>"
```

3. Publish the empty versions of the following extensions:

   - **System Application** extension
   - **Base Application** extension
   - Customization extensions (if any).

   This step publishes the extensions you created in Task 3. Publish the extensions using the Publish-NAVApp, like in the previous steps. Except if the extensions aren't signed, use the `-SkipVerification` switch parameter.

4. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 9: Synchronize tenant

In this task, you'll synchronize the tenant's database schema with any schema changes in the application database and extensions.

If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 15 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

```
Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
<database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
```

> **IMPORTANT**
> You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

> **NOTE**
> For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the table migration extension. This extension is the tables-only extension you created in task 2.

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "<table migration
   extension>" -Version <extension version>
   ```

   This step creates empty tables in the database for the table objects defined in the table migration extension. When completed, the table migration extension takes ownership of the table. In SQL Server, you'll notice that the table names will be suffixed with the extension ID. At this point, the tenant state is OperationalDataUpgradePending.

   TIP

   To verify the tenant state, run Get-NAVTenant cmdlet with the `-ForceRefresh` switch:

   `Get-NAVTenant <server instance> -Tenant <default> -ForceRefresh`

4. Synchronize the empty versions of system application, base application, and customization extensions that you published in Task 8.

Task 10: Install DestinationAppsForMigration and move tables

In this task, you run a data upgrade on tables to handle data changes made by platform and extensions. The step installs table migration extension. It moves data from the old tables to the new tables owned by the table migration extension.

1. To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -
   FunctionExecutionMode Serial -SkipAppVersionCheck
   ```

2. To view the progress of the data upgrade, you can run Get-NavDataUpgrade cmdlet with the `-Progress` switch.

   When completed, the table migration extension will be installed.

3. Install the empty versions of the system, base, and custom extensions that you published in Task 8.

   To install the extension, you use the Install-NAVApp cmdlet.

   ```
   Install-NAVApp -ServerInstance <server instance name> -Name "<name>" -Version <extension version>
   ```

4. (Multitenant only) Repeat steps 1 and 2 for each tenant.

Task 11: Publish final extensions

This step starts the second phase of the data upgrade. You'll publish the second version of the table migration extension and the production versions of extensions. The production extensions include the new versions of Microsoft System Application, Base Application extension, and customization extensions. The extension packages for Microsoft extensions are on the installation media (DVD). Customization extensions include the extension versions that you created in Task 1, not the empty versions.

Publish extensions using the Publish-NAVApp cmdlet like you did in previous steps.

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to extension .app file>"
```

Publish the extensions in the following order:

1. Second version of the table migration extension, which is the empty version with the migration.json file.
2. Microsoft System Application

   Publish the Microsoft_System Application.app extension package file that is in the **Applications\System Application\Source** folder of installation media (DVD).
3. Microsoft Base Application

   Publish the Microsoft_Base Application.app extension package file that is in the **Applications\BaseApp\Source** folder of installation media (DVD).

   > **NOTE**
   > The other .app files in this folder, like Microsoft_Danish language (Denmark).app, are extensions that add translations for a specific language. By publishing and installing these extensions, you add the capability of showing the base application in another language. These extensions aren't required to complete the upgrade and can be published and installed later.
4. Customization extensions.
5. Application extension.
6. Microsoft and third-party extensions.

   The Microsoft extensions are in the **Applications** folder of installation media (DVD).

Task 12: Synchronize final extensions

Synchronize the newly published extensions using the Sync-NAVApp cmdlet like you did in previous steps.

```
Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "<extension name>" -Version
<extension version>
```

Synchronize the extensions in the following order:

1. Microsoft System Application
2. Microsoft Base Application
3. Microsoft Application
4. Microsoft and third-party extensions
5. Customization extensions
6. Second version of the table migration extension (empty version)

IMPORTANT

Synchronize extensions in the order of dependencies. The migration extension must be synchronized last. This step will change table ownership to the system and base application.

Task 13: Upgrade empty table migration extension

Run data upgrade on the table migration extension (empty version) by using the Start-NAVAppDataUpgrade cmdlet. For example:

```
Start-NAVAppDataUpgrade -ServerInstance <server instance> -Name "<table migration extension>" -version
<version 2>
```

Task 14: Upgrade and install final extensions

The final step is to upgrade to the new extension versions in the following order. Use the Start-NAVAppDataUpgrade or Install-NAVApp cmdlets like you did in the previous task.

Run the data upgrade on the extensions in the following order:

1. Upgrade the Microsoft System Application extension.
2. Upgrade the Microsoft Base Application extension.
3. Install the Microsoft Application extension
4. Upgrade customization extensions, Microsoft, and third-party extensions.

   For customization extensions, only do this task for those extensions that have an empty version currently installed on the tenant (see **Task 11**). If you have a customization extension for which you didn't create and publish an empty version, complete the next task to install these extensions.

Task 15: Install remaining customization extensions

Complete this task for customizations extension that you created in Task 1, but create and publish an empty version first.

To install each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name "<name>" -Version <extension version>
```

Task 16: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins that must be upgraded.

- Microsoft.Dynamics.Nav.Client.BusinessChart
- Microsoft.Dynamics.Nav.Client.FlowIntegration
- Microsoft.Dynamics.Nav.Client.OAuthIntegration
- Microsoft.Dynamics.Nav.Client.PageReady
- Microsoft.Dynamics.Nav.Client.PowerBIManagement
- Microsoft.Dynamics.Nav.Client.RoleCenterSelector
- Microsoft.Dynamics.Nav.Client.SocialListening

- Microsoft.Dynamics.Nav.Client.SatisficationSurvey
- Microsoft.Dynamics.Nav.Client.TimelineVisualization
- Microsoft.Dynamics.Nav.Client.VideoPlayer
- Microsoft.Dynamics.Nav.Client.WebPageViewer
- Microsoft.Dynamics.Nav.Client.WelcomeWizard

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is
   `C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`
   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Instead, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Post-upgrade tasks

1. Uninstall the table migration extension.
2. Enable task scheduler on the server instance.
3. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
4. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
5. Change application version.

   (Optional) This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version.

   On the **Help and Support** page in the client, you'll see an application version, such as 14.0.2345.6. For an explanation of the number, see Version numbers in Business Central. This version isn't updated automatically when you install an update. If you want the version to reflect the version of the update or your own version, you change it manually.

   We recommend setting the value to application build number for the version 17 update. You get the number from the Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

   a. Run the Set-NAVApplication cmdlet:

   ```
   Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
   version> -Force
   ```

   For example:

   ```
   Set-NAVApplication -ServerInstance BC170 -ApplicationVersion 17.0.38071.0 -Force
   ```

   b. Run the Sync-NAVTenant cmdlet to synchronize the tenant with the application database.

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   c. Run the Start-NavDataUpgrade cmdlet to change the version number:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -FunctionExecutionMode Serial -Tenant
   <tenant ID>
   ```
6. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 17 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of

the following steps:

- Assign the **EXCEL EXPORT ACTION** permission set to appropriate users.
- Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

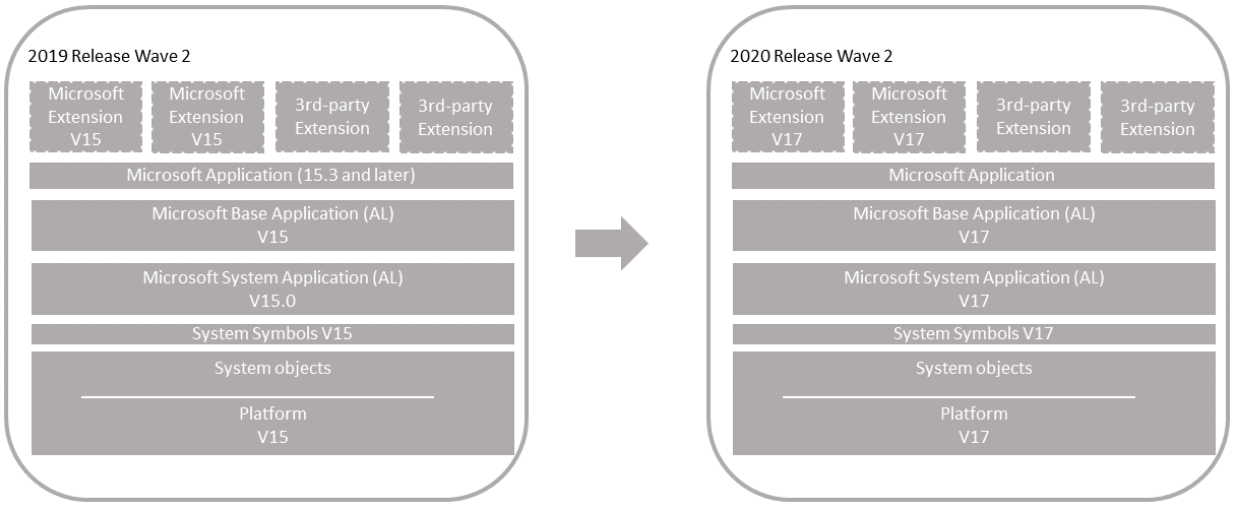
See Also

Publishing and Installing an Extension
Upgrading to Business Central
Signing an APP Package File

Upgrading Version 15 Base Application to Version 17

2/17/2021 • 13 minutes to read • Edit Online

Use this scenario if you have a Business Central 2019 release wave 2 solution that uses the Microsoft System and Base applications.



Single-tenant and multitenant deployments

The process for upgrading the similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application code and business data are in the same database. In a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Your version 15 is compatible with version 17.

   There are several updates for version 15. The updates have a compatible version 17 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix. For example, if your solution is currently running 15.5, you can't upgrade to 17.0. You must wait until 17.1 is available.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Prepare version 15 databases

1. Make backup of the databases.
2. Start Business Central Administration Shell for version 15 as an administrator.
3. (Single-tenant only) Uninstall all extensions from the old tenants.

In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

a. Get a list of installed extensions.

   This step is optional, but it can be useful to the names and versions of the extensions.

   To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

   ```
   Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

   For a single-tenant deployment, set the `<tenant ID>` to default.

b. Uninstall the extensions.

   To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

   ```
   Uninstall-NAVApp -ServerInstance <server instance name> -Name <extensions name> -Tenant
   <tenant ID> -Version <extension version> -Force
   ```

   Replace `<extension name>` and `<extension version>` with the exact name and version the published System Application.

   For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-
   NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
   $_.Version -Force}
   ```

4. Unpublish all system symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC15 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

5. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

6. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 2: Install version 17

1. Download the latest available update for Business Central 2020 release wave 2 (version 17) that is compatible with your version 15.

For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

2. Before you install version 17, it can be useful to create desktop shortcuts to the version 15.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 17 tools.

3. Install Business Central version 17 components.

   You'll keep version 15 installed to complete some steps in the upgrade process. When you install version 17, you must either specify different port numbers for components (like the Business Central Server instance and web services) or you must stop the version 15.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 3: Convert version 15 database

This task runs a technical upgrade on the application database to convert it from the version 15 platform to the version 17 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 17 as an administrator.

2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (15-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 4: Configure version 17 server

When you installed version 17 in **Task 2**, a version 17 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 15 to version 17 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task Scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 5: Import version 17 license

1. Use the Import-NAVServerLicense to upload the version 17 license to the database.

```
Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path and file name>
```

2. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 6: Publish symbols and extensions

In this task, you'll publish the platform symbols and extensions. As minimum, you publish the new base application and system application extensions from the installation media (DVD). You also publish new versions of any Microsoft extensions and third-party extensions that were used on your old deployment.

Publishing an extension adds the extension to the application database that is mounted on the server instance. Once published, it's available for installing on tenants. This task updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 17 that you started in the previous task.

1. Publish version 17 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

```
Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
SymbolsOnly
```

   What are symbols?

2. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
Application.app>"
```

   What is the System Application?

3. Publish the Business Central base application extension (Microsoft_Base Application.app).

The **Base Application** extension contains the application business objects. You find the (Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
Application.app>"
```

4. Publish the Microsoft_Application extension.

   The Microsoft_Application extension is a new extension introduced in 15.3. For more information about this extension, see The Microsoft_Application.app File.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Application.app>"
   ```

5. Publish the new versions of Microsoft extensions.

   In this step, you publish new versions of Microsoft extensions that were used on your version 15 deployment. You find the extensions in the **Applications** folder of the installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft extension>"
   ```

   For example:

   ```
   Publish-NAVApp -ServerInstance BC170 -Path
   "C:\W1DVD\Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app"
   ```

6. Publish new versions of 3rd-party extensions.

   If you have new versions of these extensions, built on the Business Central version 17, then publish the new versions.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to extension>"
   ```

7. Recompile extensions not build on version 17.

   This step pertains to any published extension versions that aren't built on version 17, which you want to reinstall on tenants. These extensions must be recompiled to work with version 17. To recompile the extensions, use the Repair-NAVApp cmdlet:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   For example, to recompile all extensions that are not published by Microsoft, you could run the following command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Where-Object {$_.Publisher -notlike 'Microsoft'} |
   Repair-NAVApp
   ```

   Restart the Business Central Server when completed.

Task 7: Synchronize tenant

You'll synchronize the tenant's database schema with any schema changes in the application database and extensions. If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 15 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
   <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

   IMPORTANT

   You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   NOTE

   For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

   At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the **System Application** extension.

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application. To get the version, you can use the Get-NAVAppInfo cmdlet.

4. Synchronize the tenant with the Business Central Base Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

5. Synchronize the tenant with the Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
   ```

6. Synchronize the tenant with Microsoft and 3rd-party extensions.

   For each extension, run the Sync-NAVApp cmdlet:

```
Sync-NAVApp -ServerInstance <server instance name> -Tenant default -Name "<extension name>" -Version
<extension version>
```

Task 8: Upgrade data

In this task, you run a data upgrade for extensions.

Single tenant

Run the data upgrade on extensions in order of dependency.

1. Run the data upgrade for the system application, followed by the base application.

   To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Name "<extension name>" -Version
   <extension version>
   ```

   This step will automatically install the new system application and base application versions on the tenant.

2. Upgrade the new versions of Microsoft extensions and third-party extensions.

   Complete this task to upgrade any Microsoft and third-party extension used in the old deployment to new versions on the installation media. The new versions are in the **Application** folder of the DVD. There's a folder for each extension. The extension package (.app file) is in the **Source** folder.

   a. Install **Application** extension (when comping from 15.2 and earlier only).

      You'll have to install the **Application** extension first, otherwise you can't upgrade Microsoft extensions.

      ```
      Install-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
      ```

   b. For each extension, run Start-NAVAppDataUpgrade cmdlet.

      This step will also automatically install the new extension version on the tenant.

Multitenant

On each tenant, run the Start-NavDataUpgrade cmdlet as follows:

```
Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -FunctionExecutionMode
Serial -SkipAppVersionCheck
```

This command will upgrade and install the extensions on the tenant.

Task 9: Install 3rd-party extensions

Complete this task to install third-party extensions for which a new version wasn't published. For each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
```

Task 10: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

At this point, the upgrade is complete, and you can open the client.

Post-upgrade tasks

1. Enable task scheduler on the server instance.

2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.

3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

4. Change application version.

   (Optional) This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version.

   On the **Help and Support** page in the client, you'll see an application version, such as 15.0.2345.6. For an explanation of the number, see Version numbers in Business Central. This version isn't updated automatically when you install an update. If you want the version to reflect the version of the update or your own version, you change it manually.

   We recommend setting the value to application build number for the version 17 update. You get the number from the Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

   a. Run the Set-NAVApplication cmdlet:

   ```
   Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
   version> -Force
   ```

   For example:

   ```
   Set-NAVApplication -ServerInstance BC170 -ApplicationVersion 17.0.38071.0 -Force
   ```

   b. Run the Sync-NAVTenant cmdlet to synchronize the tenant with the application database.

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   c. Run the Start-NavDataUpgrade cmdlet to change the version number:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -FunctionExecutionMode Serial -Tenant
   <tenant ID>
   ```

5. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 17 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

- Assign the **EXCEL EXPORT ACTION** permission set to appropriate users.
- Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

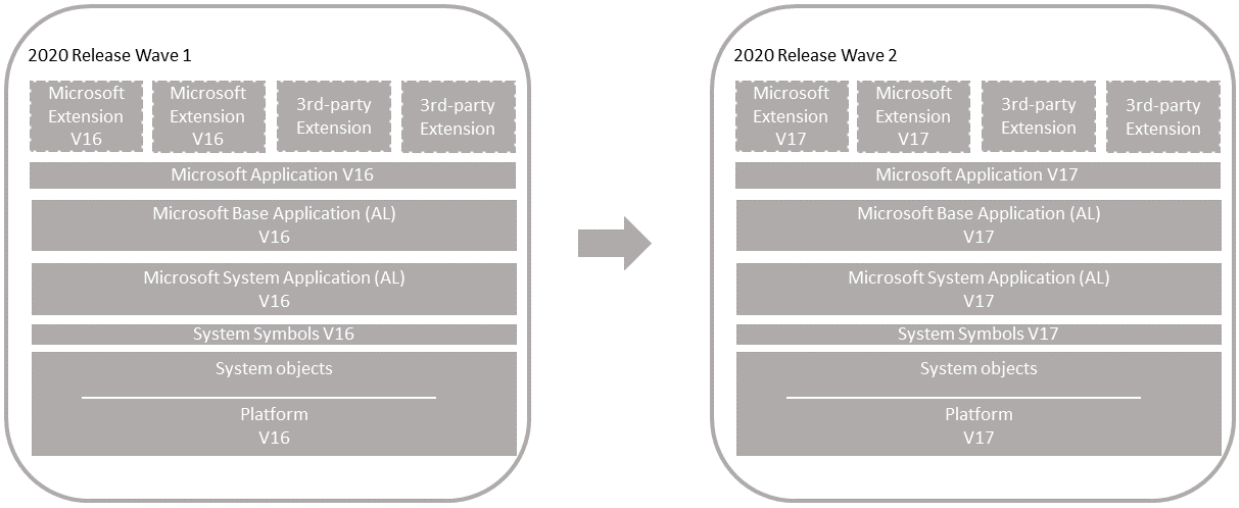
See Also

Upgrading to Business Central
Upgrading Extensions

Upgrading Version 16 Microsoft System and Base Application to Version 17

2/17/2021 • 12 minutes to read • Edit Online

Use this scenario if you have a Business Central 2020 release wave 1 solution that uses the Microsoft System and Base applications.



Single-tenant and multitenant deployments

The process for upgrading the similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application code and business data are in the same database. In a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Your version 16 is compatible with version 17.

   There are several updates for version 16. The updates have a compatible version 17 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix. For example, if your solution is currently running 16.6, you can't upgrade to 17.0. You must wait until 17.1 is available.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Prepare version 16 databases

1. Make backup of the databases.

2. Start Business Central Administration Shell for version 16 as an administrator.

3. (Single-tenant only) Uninstall all extensions from the old tenants.

   In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>
      ```

      For a single-tenant deployment, set the `<tenant ID>` to default.

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Name <extensions name> -Tenant
      <tenant ID> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the published System Application.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID>| % { Uninstall-
      NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
      $_.Version -Force}
      ```

4. Unpublish all system symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet with the `-SymbolsOnly` switch.

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC15 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

5. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <server instance name> -Tenant <tenant ID>
   ```

6. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <server instance name>
   ```

Task 2: Install version 17

1. Download the latest available update for Business Central 2020 (version 17) that is compatible with your

version 16.

For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

2. Before you install version 17, it can be useful to create desktop shortcuts to the version 16.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 16 tools.

3. Install Business Central version 17 components.

   You can keep version 16 installed to complete some steps in the upgrade process. When you install version 17, you must either specify different port numbers for components (like the Business Central Server instance and web services) or you must stop the version 16.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 3: Convert version 16 database

This task runs a technical upgrade on the application database to convert it from the version 16 platform to the version 17 platform. The conversion updates the system tables of the database to the new schema (data structure). It provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 17 as an administrator.

2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
   instance> -DatabaseName "<database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (16-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 4: Configure version 17 server

When you installed version 17 in **Task 2**, a version 17 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 16 to version 17 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName DatabaseName -KeyValue "
   <database name>"
   ```

   In a single tenant deployment, this command will mount the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task Scheduler on the server instance for purposes of upgrade.

```
Set-NavServerConfiguration -ServerInstance <server instance name> -KeyName "EnableTaskScheduler" -
KeyValue false
```

Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 5: Import version 17 license

1. Use the Import-NAVServerLicense to upload the version 17 license to the database.

```
Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path and file name>
```

2. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 6: Publish symbols and extensions

In this task, you'll publish the platform symbols and extensions. As minimum, you publish the new base application and system application extensions from the installation media (DVD). You also publish new versions of any Microsoft extensions and third-party extensions that were used on your old deployment.

Publishing an extension adds the extension to the application database that is mounted on the server instance. Once published, it's available for installing on tenants. This task updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 16 that you started in the previous task.

1. Publish version 17 system symbols extension.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

```
Publish-NAVApp -ServerInstance  <server instance name> -Path "<path to system.app>" -PackageType
SymbolsOnly
```

   What are symbols?

2. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
Application.app>"
```

What is the System Application?

3. Publish the Business Central base application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the (Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
   Application.app>"
   ```

4. Publish the Microsoft_Application extension.

   For more information about this extension, see The Microsoft_Application.app File.

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Application.app>"
   ```

5. Publish the new versions of Microsoft extensions.

   In this step, you publish new versions of Microsoft extensions that were used on your version 16 deployment. You find the extensions in the **Applications** folder of the installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft extension>"
   ```

   For example:

   ```
   Publish-NAVApp -ServerInstance BC170 -Path
   "C:\W1DVD\Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app"
   ```

6. Publish new versions of 3rd-party extensions.

   If you have new versions of these extensions, built on the Business Central version 17, then publish the new versions.

   ```
   Publish-NAVApp -ServerInstance BC170 -Path "<path to extension>"
   ```

7. Recompile extensions not build on version 17.

   This step pertains to any published extension versions that aren't built on version 16, which you want to reinstall on tenants. These extensions must be recompiled to work with version 16. To recompile the extensions, use the Repair-NAVApp cmdlet:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   For example, to recompile all extensions that are not published by Microsoft, you could run the following command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Where-Object {$_.Publisher -notlike 'Microsoft'} |
   Repair-NAVApp
   ```

   Restart the Business Central Server when completed.

Task 7: Synchronize tenant

You'll synchronize the tenant's database schema with any schema changes in the application database and extensions. If you have a multitenant deployment, do these steps for each tenant.

1. (Multitenant only) Mount the tenant to the version 17 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

   ```
   Mount-NAVTenant -ServerInstance <server instance name> -DatabaseName <database name> -DatabaseServer
   <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
   ```

   IMPORTANT

   You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you'll get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

   NOTE

   For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

   At this stage, the tenant state is OperationalWithSyncPending.

2. Synchronize the tenant with the application database.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

3. Synchronize the tenant with the **System Application** extension.

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application. To get the version, you can use the Get-NAVAppInfo cmdlet.

4. Synchronize the tenant with the Business Central Base Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

5. Synchronize the tenant with the Application extension.

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Application"
   ```

6. Synchronize the tenant with Microsoft and 3rd-party extensions.

   For each extension, run the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance BC150 -Tenant default -Name "<extension name>" -Version <extension
   version>
   ```

Task 8: Upgrade data

In this task, you run a data upgrade for extensions.

Single tenant

Run the data upgrade on extensions in order of dependency.

1. Run the data upgrade for the System Application, followed by the Base Application.

   To run the data upgrade, use the Start-NavDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Name "<extension name>" -Version
   <extension version>
   ```

   This step will automatically install the new system application and base application versions on the tenant.

2. Upgrade the new versions of Microsoft extensions and third-party extensions.

   Complete this task to upgrade any Microsoft and third-party extension used in the old deployment to new versions on the installation media. The new versions are in the **Application** folder of the DVD. There's a folder for each extension. The extension package (.app file) is in the **Source** folder.

   For each extension, run Start-NAVAppDataUpgrade cmdlet. First, run the data upgrade for the Application extension, then run it for other Microsoft extensions and third-party extensions.

   This step will also automatically install the new extension version on the tenant.

Multitenant

On each tenant, run the Start-NavDataUpgrade cmdlet as follows:

```
Start-NAVDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -FunctionExecutionMode
Serial -SkipAppVersionCheck
```

This command will upgrade and install the extensions on the tenant.

Task 9: Install 3rd-party extensions

Complete this task to install third-party extensions for which a new version wasn't published. For each extension, run the Install-NAVApp cmdlet:

```
Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
```

Task 10: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

At this point, the upgrade is complete, and you can open the client.

Post-upgrade tasks

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you

mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.

3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

4. Change application version.

   (Optional) This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version.

   On the **Help and Support** page in the client, you'll see an application version, such as 16.0.2345.6. For an explanation of the number, see Version numbers in Business Central. This version isn't updated automatically when you install an update. If you want the version to reflect the version of the update or your own version, you change it manually.

   We recommend setting the value to application build number for the version 17 update. You get the number from the Released Updates for Microsoft Dynamics 365 Business Central 2020 Release Wave 2 on-premises.

   a. Run the Set-NAVApplication cmdlet:

   ```
   Set-NAVApplication -ServerInstance <server instance name> -ApplicationVersion <new application
   version> -Force
   ```

   For example:

   ```
   Set-NAVApplication -ServerInstance BC160 -ApplicationVersion 16.0.38071.0 -Force
   ```

   b. Run the Sync-NAVTenant cmdlet to synchronize the tenant with the application database.

   ```
   Sync-NAVTenant -ServerInstance <server instance name> -Mode Sync -Tenant <tenant ID>
   ```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   c. Run the Start-NavDataUpgrade cmdlet to change the version number:

   ```
   Start-NAVDataUpgrade -ServerInstance <server instance name> -FunctionExecutionMode Serial -Tenant
   <tenant ID>
   ```

See Also

Upgrading to Business Central
Upgrading Extensions

Code Conversion from C/AL to AL

2/17/2021 • 11 minutes to read • Edit Online

This article explains how to convert a Business Central (version 14) C/AL code-customized on-premises solution to AL code.

You'll use this procedure as part of the upgrade process when going from version 14 to a later version, like 15, 16, or 17.

Before you start

Get familiar with the basics of setting up and developing in Visual Studio Code and AL, see Developing Extensions in AL.

> **NOTE**
> Moving on-premise C/AL code customizations to Dynamics 365 Business Central online, requires converting these to AL extensions. This could include converting the C/AL deltas to AL extension code as a starting point, as outlined in The Txt2Al Conversion Tool.

Also, review the Known Issues for information about issues that may affect the upgrade.

Breaking changes

When converting from C/AL to AL, it's important that you don't introduce any breaking schema changes to the database. Otherwise, you won't be able to synchronize the new extension with the database.

Task 1: Import the test library into your C/AL solution

If your solution uses Microsoft (1st-party) extensions, you'll have to convert the test library from C/AL to AL. The reason for this is that the Microsoft extensions rely on the test symbols. The easiest way is to import the **CALTestLibraries.W1.fob** file into the old database. This file is available on the version 14 installation media (DVD) in the **TestToolKit** folder.

You can do this using the (Dynamics NAV Development Environment). For more information, see Exporting and Importing Objects.

Task 2: Compile all the objects in your C/AL solution

Compiling all the objects is a prerequisite for a successful and complete export. To compile objects, you can use either of the following tools:

- C/SIDE (Dynamics NAV Development Environment). See Compiling Objects.
- Compile-NAVApplicationObject cmdlet of the Dynamics NAV Development Shell. Make sure to run this cmdlet as an administrator.

Task 3: Export the application objects to the new TXT syntax

Once the application compiles, you must export all C/AL application objects, except system tables and codeunits (IDs in the 2000000000 range), to the new TXT format. The exported objects will be used as input to the Txt2AL conversion tool. To export objects, use the Export-NAVApplicationObject cmdlet of the Dynamics NAV Development Shell. it's important to:

- Omit omit all system objects, which have IDs in the 2000000000 range.
- Use the `ExportToNewSyntax` switch to export the objects in a syntax that is compatible with the Txt2Al conversion tool.

The Export-NAVApplicationObject cmdlet will export all objects to a single .txt file. If you imported the test library objects into the database, then you'll export the base application objects and the test library separately. Later, you'll create a separate AL project for each set of files.

For example, do the following steps:

1. Export the custom base application objects.
   a. Create a folder for storing the exported base application objects to TXT files (for example, c:\export2al\baseapplication).
   b. Run the following commands to export tha application objects, but omitting the system objects and test library objects.

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\baseapplication\exportedbc14app-part1.txt" -Filter
'Id=1..129999'
```

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\baseapplication\exportedbc14app-part2.txt" -Filter
'Id=140000..1999999999'
```

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\baseapplication\exportedbc14testobjects.txt" -Filter
'Id=130400..130416'
```

2. Export the test library objects.
   a. Create a folder for storing the exported test library objects to TXT files (for example, c:\export2al\testlibrary).
   b. Run the following commands to export the test library objects only

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\testlibrary\bc14testlibrary-part1.txt" -Filter
'Id=130000..130399'
```

```
Export-NAVApplicationObject -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (14-0)" -
ExportToNewSyntax -Path "c:\export2al\testlibrary\bc14testlibrary-part2.txt" -Filter
'Id=130440..139999'
```

Task 4: Create a declaration file for custom .NET assemblies (optional)

If your solution contains .NET interoperability code and control add-ins, you can create a file that contains the declarations to the assemblies. This file will be used when you convert the C/AL TXT files to AL in the next step. Alternatively, after the conversion, you'll have to manually add the declarations to objects that use the assemblies.

To create the file, use a text editor or Visual Studio code to create a file that contains the assembly declarations as follows:

```
dotnet

{
    assembly("Microsoft.Dynamics.Nav.Client.BusinessChart")
    {

type("Microsoft.Dynamics.Nav.Client.BusinessChart.BusinessChartAddIn";"Microsoft.Dynamics.Nav.Client.Busines
sChart")
        {
            IsControlAddIn = true;
        }

        // Other control add-ins in this assembly
    }

    // Other assemblies containing control add ins
}
```

Save the file with any name and the extension **.al**, for example **mydotnet.al**. Make a note of the path because you'll use it in the next step.

Task 5: Convert the C/AL TXT files to AL

With C/AL exported to the new TXT format, you now convert the code to AL using the The Txt2Al Conversion Tool. The Txt2Al creates .al files for each object in the TXT files. Similar to **Task 3**, if you imported the test library objects into the database, then you'll convert the base application objects and the test library separately.

Get the Txt2AL conversion tool

The Txt2Al conversion tool (txt2al.exe) is only available with version 14, which is the last version to support C/AL. Use this version of the tool no matter what later version you may eventually be upgrading to. The AL objects created by the tool will be compatible with later versions.

You find the txt2al.exe on the installation media (DVD) in the "DVD\RoleTailoredClient\program files\Microsoft Dynamics NAV\140\RoleTailored Client" folder. Or, it's installed locally with Dynamics NAV Development Environment, for example, in the "C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client" folder.

Run the Txt2AL conversion tool

1. Convert the base application TXT files to AL.

   a. Create a folder for storing the AL files for base application objects (for example, c:\export2al\baseapplication\al).

   b. Start a command prompt as administrator, and navigate to the folder that contain txt2al.exe file.

      By default, the location is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client.

   c. Run the txt2al command:

      ```
      txt2al --source=C:\export2al\baseapplication --target=C:\export2al\baseapplication\al --
      injectDotNetAddIns --dotNetAddInsPackage=C:\export2al\dotnet\mydotnet.al --dotNetTypePrefix=BC
      --rename
      ```

      If your solution contains .NET interoperability code, the following Txt2Al command line parameters are used to achieve a conversion that requires less manual intervention:

      - `--injectDotNetAddIns` injects the definition of standard .NET add-ins in the resulting .NET

package. The standard .NET add-ins are a set of add-ins that are embedded into the platform.

- `--dotNetAddInsPackage` should be used to point the conversion tool to an AL file containing declarations for the .NET types that represent .NET control addins. Use this to inject a custom set of .NET control add-in declarations. This parameter is only required if you completed **Task 4**, and you set it to point to the location of the dotnet.al file.

  > **NOTE**
  > If you are interested in migrating your localization resources, you should use the `--addLegacyTranslationInfo` switch to instruct Txt2Al to generate information about the legacy IDs of the translation code.

- `--dotNetTypePrefix` specifies a prefix to be used for all .NET type aliases created during the conversion. This will ensure that no naming conflicts occur with existing types. In the example, `BC` is the prefix.

- `--rename` renames the output files to prevent clashes with the source .txt files.

When completed, there will be an .al file for each object.

2. Convert the test library TXT files to AL.

   This is similar to the previous step.

   a. Create a folder for storing the AL files for base application objects (for example, c:\export2al\baseapplication\al).

   b. Run the txt2al command:

   ```
   txt2al --source=C:\export2al\testlibrary --target=C:\export2al\testlibrary\al --
   injectDotNetAddIns --dotNetTypePrefix=BCTest --rename
   ```

   Use a different value for the `--dotNetTypePrefix` than you did for the base application.

Task 6: Create a new application database for development

To build your base application, you'll create a new application database on the Business Central version 15, 16, or 17 platform. This will only be used during development.

1. Start the Business Central Administration Shell for version 16 as an administrator.
2. Run the New-NAVApplicationDatabase cmdlet to create the database. For example:

   ```
   New-NAVApplicationDatabase -DatabaseServer .\BCDEMO -DatabaseName MyDBforupgrade
   ```

3. Connect your Business Central Server instance to the database. See Connecting a Business Central Server Instance to a Database.

   ```
   Set-NAVServerConfiguration -ServerInstance BC -KeyName DatabaseName -KeyValue "MyDBforupgrade"
   ```

4. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance BC
```

Task 7: Create and build an AL project for custom base application

In this task, you'll create a AL project in Visual Studio code that you'll use for building your custom base application extension based on your converted C/AL application.

1. If you haven't already, install Visual Studio Code and the latest AL Language extension for version 15, 16, or 17 as outlined in Getting Started with AL.

2. Configure Visual Studio Code for optimal performance with AL projects.

   This step is optional, but recommended. For more information, see Optimize Visual Studio Code for Editing and Building.

3. In Visual Studio Code, from the **Command Palette**, select the **AL Go!** command to create a new project.

   Specify the path for the project, and set the **Target Platform** to **4.0 Business Central 2019 release wave 2** or **5.0 Business Central 2020 release wave 1**. When prompted to select your server, choose **Your own server**.

4. Create a **.alpackages** folder in the root folder of the project and then copy the system (platform) symbols extension (System.app file) to the folder.

   The System.app file is located where you installed the AL Development Environment, which by default is the C:\Program Files (x86)\Microsoft Dynamics 365 Business Central<150, 160, or 170>\AL Development Environment folder. This package contains the symbols for all the system tables and codeunits.

5. Delete the **HelloWorld.al** sample file from the project.

6. Modify the `settings.json` file of Visual Studio Code to configure the assembly probing path.

   Change `"al.assemblyProbingPaths": ["./.netpackages"]` to point to all the folders that contain .NET assemblies that are used by your project. Here is an example that contains the most typical paths:

   ```
   "al.assemblyProbingPaths": [
   "C:\\Program Files\\Microsoft Dynamics 365 Business Central\\150",
   "C:\\Program Files (x86)\\Microsoft Dynamics 365 Business Central\\150\\RoleTailored Client",
   "C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\Framework\\.NETFramework\\v4.7.2",
   "C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\WindowsPowerShell\\3.0"
   ]
   ```

   For more information about the settings.json, see User and Workspace Settings.

   > **NOTE**
   > Adding assemblies to the folders in your assembly probing paths is not automatically detected by the compiler. You must restart Visual Studio Code for the changes to be detected.

7. Modify the `app.json` for the project as follows:

   - **Important** The ID, name, and publisher, and version of the custom base application must match the Business Central base application. Set the parameters to the following values`:

```
"appId": "437dbf0e-84ff-417a-965d-ed2bb9650972",
"name": "Base Application",
"publisher": "Microsoft",
"version": "14.5.0.0"
```

We recommend that you set the "version" to the same version as the C/AL application.

- Add the setting `"target": "OnPrem"` somewhere in the file.
- Change the `idRange` to include all the IDs used by your base application (or leave blank).
- Delete the values in the `dependencies` parameter.

8. Copy all of the base application AL files generated in the previous task (**Task 5**) to the root folder of your project.
9. Open the **dotnet.al** file for the project, and make the following changes:
   - Delete all instances of `Version = '14.0.0.0';` for **Microsoft.Dynamics.Nav** assembly declarations.
   - For the `DocumentFormat.OpenXml` assembly declaration, remove the `version` and `culture` keys and set `PublicKeyToken = '8fb06cb64d019a17'`.

   ```
   assembly("DocumentFormat.OpenXml")
   {
       PublicKeyToken = '8fb06cb64d019a17';
       ...
   ```
10. Delete objects that are related to the client debugger client.

    Debugging from the client has been discontinued, and replaced by AL Debugger. The version 14 debugger objects are not supported on version 15, 16, or 17. To avoid compilation errors, delete the following objects:

    - Debugger.Page.al
    - DebuggerBreakpointCondition.Page.al
    - DebuggerBreakpointList.Page.al
    - DebuggerCallstackFactBox.Page.al
    - DebuggerCodeViewer.Page.al
    - DebuggerManagement.Codeunit.al
    - DebuggerVariableList.Page.al
    - DebuggerWatchValueFactBox.Page.al
    - SessionList.Page.al

    You might also have to remove references to `SessionList` in ChangeGlobalDimensions.Codeunit.al.
11. Build and compile your project (press Ctrl+Shift+B).

    The AL compiler will issue errors for constructs that are not valid. Fix any errors that occur, and build again.

> **TIP**
> If you are maintaining your C/AL solution going forward, we recommend that you fix errors in C/AL objects and convert to AL again. This makes it future changes easier to foward push changes because code bases will be similar.

When all errors are fixed, the custom base application package (.app) will be created.

Task 8: Create and build an AL project for the test library

If you converted the test library form C/AL to AL, you'll now create and build a project for test library, similar to what you did for the base application.

a. Follow steps 1 through 5 in **Task 7** to create an AL project for the test library.

b. As with base application project, you have to modify the `app.json` file, but in this case, you have to change the version and add a dependency on the base application that you created.

- Set the `"version"` to the old application version, such as `14.5.0.0`.
- Set the `"dependencies"` to include information about your custom the base application.

```
"dependencies": [
{
    "appId": "437dbf0e-84ff-417a-965d-ed2bb9650972",
    "publisher": "Microsoft",
    "name": "Base Application",
    "version": "14.5.0.0"
}
],
```

- Set the `target` to `OnPrem`.
- Change the `idRange` to include all the IDs used by your test application (or leave blank).

c. Copy all of the AL files that you generated for the test library in **Task 5** to the root folder of your project.

d. Open the **dotnet.al** file for the project, and make the following changes:

- Delete all instances of `Version = '14.0.0.0';` for **Microsoft.Dynamics.Nav** assembly declarations.

e. Build the project.

Next Steps

If you are performing a technical upgrade from version 14.0 to version 15, 16, or 17, return to the technical upgrade step where you left off.

- Technical Upgrade to version 15.0
- Technical Upgrade to to version 16.0
- Technical Upgrade to to version 17.0

See Also

The Txt2Al Conversion Tool
Developing Extensions

Migrating Tables and Fields Between Extensions

2/17/2021 • 4 minutes to read • Edit Online

INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

Data migration allows you to move table and field data between extensions. The concepts and processes in this article apply to large-scale and small-scale data migrations. A large-scale migration is typically upgrade scenario, like upgrading from Business Central version 14 to version 16. Small-scale migrations are scenarios where you want to move a limited number of objects from one extension to another.

Overview

The move is divided into two phases: development and deployment. However, before you begin, you have to determine the direction of the migration within the dependency graph.

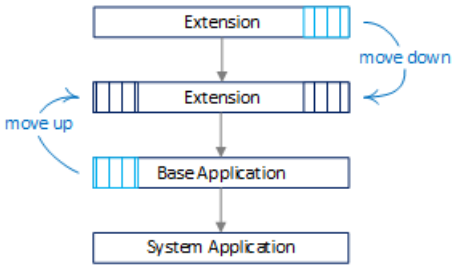
Limitations

- The concepts discussed in this article pertain only to AL. They're not supported in C/AL.
- You can only move fields from a table object to a table extension or move entire tables from one extension to another.
- You can't:
  - Move table extensions to other extensions.
  - Merge multiple table extensions into one extension.
  - Split table extensions into multiple extensions.

  Although you can't use this feature for these scenarios, you can achieve the scenarios by other means, which require more manual work. Examples include obsoleting, copying/re-IDing/renaming, moving data, and more.

Determining the migration direction in the dependency graph

The process to migrate tables and fields to another extension depends on the migration's direction in the dependency graph. The following figure illustrates a simplified extension dependency graph. From top to bottom, an extension is dependent on any extension below it in the graph.



When to move down

Typical move-down scenarios include:

- Moving to a shared extension.

  You want to move common tables to a separate extension that other extensions can have a dependency on.

- Transitioning from a customized base application extension with its own ID to an extension on top of the

Microsoft Base Application.

You have a customized base application extension with its own ID. You want to transition to the Microsoft Base Application. In this case, customizations remain in the current extension. Base objects are removed and ownership transferred to Microsoft Base Application.

This scenario is typical for embed ISV apps and on premises solutions moving to the cloud. It's also relevant for solutions that will remain on-premises for the foreseeable future. Use it to refactor code customizations into cleaner, standard base with extensions as part of upgrading.

For more information, see Moving Tables and Fields to Extension Down the Dependency Graph.

When to move up

Typical move-up scenarios include:

- Splitting an extension in two, with one dependent on the other.
- Extracting the system application from the base application.
- Transitioning from a customized base application extension with Microsoft ID to the Microsoft Base Application.

  You have a customized base application extension that reuses the Microsoft application ID. You want to transition to the standard Microsoft Base Application. In this case, customizations are moved out of the base application up into new extensions. The new extensions have new application IDs and dependencies to the standard Microsoft base application. The customization objects are removed from the custom base application and ownership transferred to the new extensions.

For more information, see Moving Tables and Fields to Extension Up the Dependency Graph.

Development

Development involves making application code changes required for the move. In short, the work involves:

- Creating the *releasing extension* version

  You create a new version of the original extension. This new version contains the tables or fields you want to keep in the extension. The tables and fields that you want to move are deleted from this extension. They're moved to the *receiving extension*.

- Creating the *receiving extension*

  You create new extension that includes the table and fields that you want moved. It essentially includes those tables and fields deleted from the releasing extension.

The key to the move is the *migration.json* file. You add the file to the project for the releasing extension. This file provides a pointer to the ID of new receiving extension where tables and fields are to be moved to. The *migration.json* is used in the deployment phase. Its purpose is to transfer ownership of tables and fields in the database from one extension to another. For more information, see Migration.json File.

Deployment

The deployment phase is when the data is migrated to new tables in the database. In this phase, ownership of tables and fields is switched from one extension to another. Deployment involves publishing, syncing, upgrading, and installing extensions.

The order that you synchronize extensions is important:

- The receiving extensions must be synchronized first.

- The releasing extensions, which include those extensions that include the migration.json file, must be synchronized last.

  These extensions are synchronized last because the tables are moved during the synchronization of the releasing extensions. At the end of the synchronization process, the system checks for breaking changes introduced by the extension. If the extension isn't synchronized last, breaking changes will be detected.

See Also

Publishing and Installing an Extension
JSON Files
Upgrading Extensions

Moving Tables and Fields to Extensions Down the Dependency Graph

2/17/2021 • 3 minutes to read • Edit Online

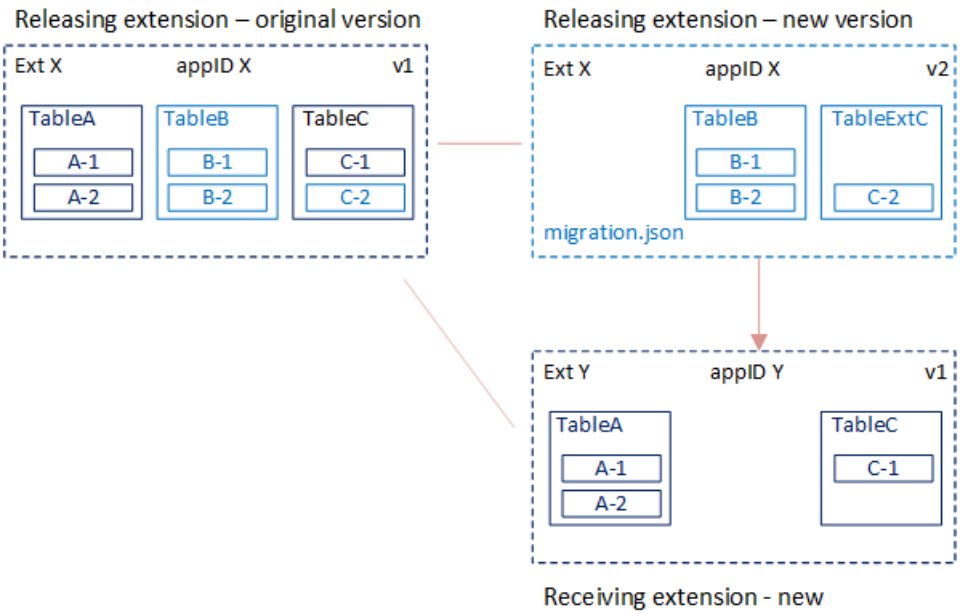
INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

This article explains how to move tables and fields from an extension to another extension that is down the dependency graph.

> **TIP**
> For more information about the general concepts for moving tables and fields between extensions, see Migrating Tables and Fields Between Extensions. This article explains the the difference between moving up and down the dependency graph.

Overview

The steps in this article are based on the example illustrated in the following figure. Although your scenario is different, the concept and process are much the same.



In the example, **TableB** and **Field C-2** are customizations. You'll keep these elements in the original extension, but create a new version without **TableA** and **TableC**. You'll move **TableA** and **TableC** down the dependency chain to a new, separate extension.

Prerequisite

If you're moving enum type fields, then your solution must be running on Business Central 2020 release wave 1, version 16.5 or later. For more information, see Known Issues.

Create receiving extension (Ext Y)

The receiving extension will contain the table and fields that you want to move. In this example, these objects

include **TableA** and **TableC**.

1. Create an AL project for the receiving extension.
2. Add a table object definition for **TableA**.

   The table definition (schema) must include the full schema of the releasing extension **Ext X**, with the same field definitions. You can add new fields.
3. Add a table object definition for **TableC**.

   The table definition (schema) must include the full schema of the releasing extension **Ext X**, with the same field definitions, except don't include field **C-2**. You can add new fields.
4. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `11111111-aaaa-2222-bbbb-333333333333`.
5. Compile the extension package.

Create new version of releasing extension (Ext X v2)

1. In the releasing extension AL project, add a migration.json file that points to the ID of the target extension.

   ```
   {
   "apprules": [
       {
           "id": "11111111-aaaa-2222-bbbb-333333333333"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.
2. Modify the app.json file as follows:

   - Ensure that `"target": "OnPrem"`.
   - Increase the `"version"` value.
   - In the `"dependencies"` parameter, set up a dependency on the new receiving extension **Ext Y**.

   For more information, see App.json file.
3. Complete the following steps for **TableC**.

   a. Add a table extension object **TableExtC**.
   b. In table extension object **TableExtC**, add a field definition for field **C-2** that matches its definition in the original **TableC** object.
   c. Delete the original **TableC** object.
4. Delete the entire table object for **TableA**.
5. Compile a new version of the extension package.

Deploy extensions

1. Uninstall the old version of the releasing extension **Ext X**.
2. Publish the new receiving extension **Ext Y** and releasing extension version **Ext X v2**.
3. Synchronize the receiving extension **Ext Y**.

This step creates empty database tables for **TableA** and **TableC** that are owned by the receiving extension **Ext Y**.

> **IMPORTANT**
>
> The receiving extension must always be synchronized first.

4. Synchronize the new version of the releasing extension **Ext X v2**.

   This step first reads rules in the migration.json file of the extension, then does the following operations in the database:

   - Creates a companion table for field **C-2** of the table extension object **TableExtC**.
   - Copies data from column **C-2** in the original **TableC** to new companion table **TableExtC**.
   - Temporarily renames the new empty tables **TableA** and **TableC** made by receiving extension **Ext Y**.
   - Renames the original tables **TableA** and **TableC** that include the data. Instead of including the ID of the releasing extension **Ext X**, the names are changed to include the ID of the receiving extension **Ext Y**. This step essentially transfers ownership from **Ext X** to **Ext Y**.
   - Deletes the unused column for **C-2** in the original table **TableC**.
   - Deletes the empty, renamed tables of **Ext Y**.
5. Install the receiving extension **Ext Y**.
6. Run Start-NAVAppDataUpgrade cmdlet on the new releasing extension version **Ext X v2**.

   This step basically installs the new extension version. You run a data upgrade because an earlier version has been installed and is still published.

See Also

Migrating Tables and Fields Between Extensions
Moving Tables and Fields to Extension Up the Dependency Graph
Upgrading Extensions
Publishing and Installing an Extension
JSON Files

Moving Tables and Fields to Extensions Up the Dependency Graph

2/17/2021 • 5 minutes to read • Edit Online

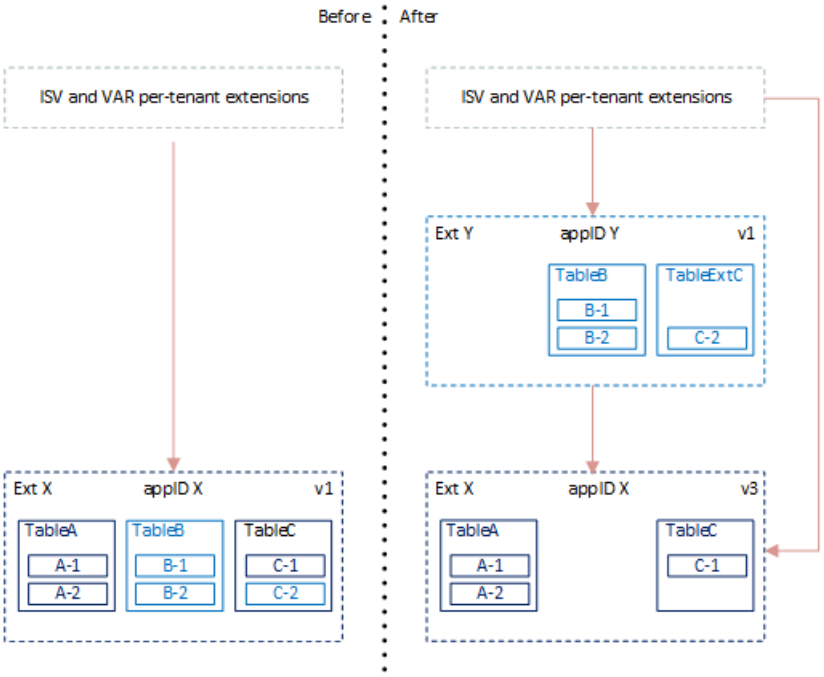
INTRODUCED IN: Business Central 2019 Release Wave 2, update 15.3, for on-premises only

This article explains how to move tables and fields from an extension to another extension that is up the dependency graph.

> **TIP**
> For more information about the general concepts for moving tables and fields between extensions, see Migrating Tables and Fields Between Extensions. This article explains the the difference between moving up and down the dependency graph.

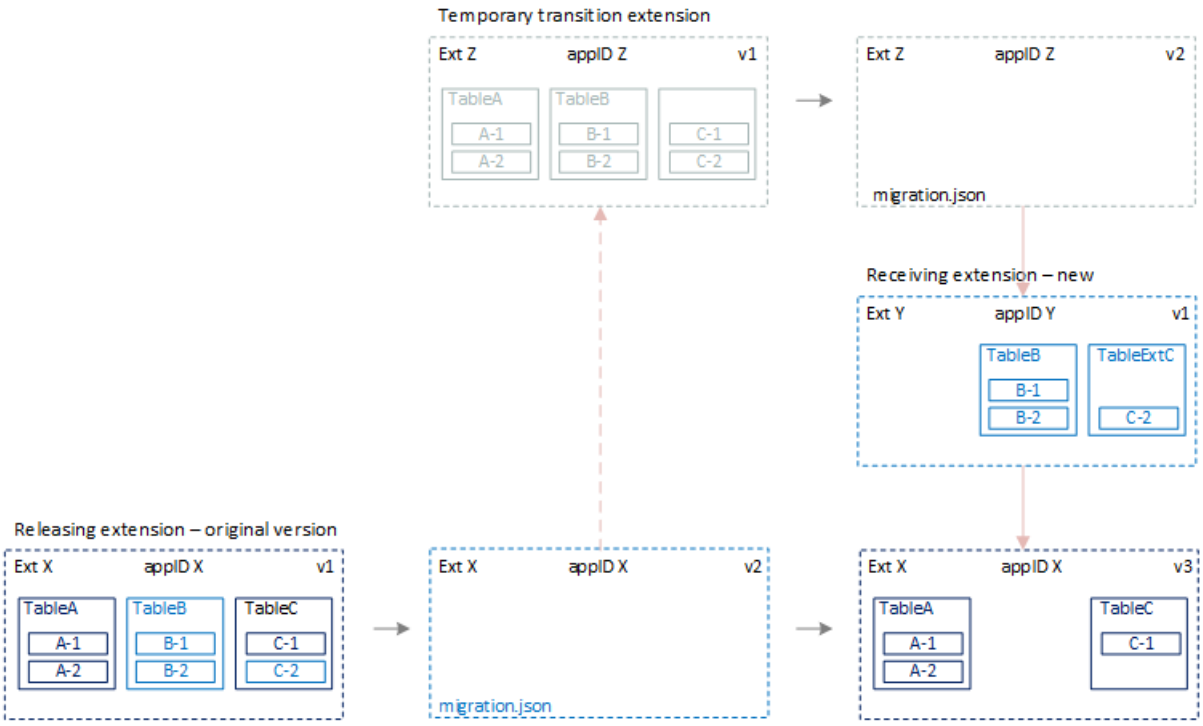
Overview

The steps are based on the example illustrated in the following figure. Although your scenario is different, the concept and process are much the same.



In the example, **TableB** and **Field C-2** are customizations. You'll move these elements from the original extension up to a new extension. This new extension will have a dependency on the original extension. You'll keep **TableA** and **TableC** in the original extension.

To accommodate data migration, you'll have to create an extension that is only used for deployment. This extension is **Ext Z** in the figure. There are two stages of deployment:



- In the first stage, **Ext Z** temporarily takes ownership of tables and fields from **Ext X**.
- In the second stage, **Ext Z** releases ownership to extensions **Ext X** and **Ext Y**. You uninstall and unpublish transition extension **Ext Z** when you finish deployment.

This process is a two-step process because we only support moving down the dependency graph. So instead, the concept is to first move the tables' ownership to an extension above the receiving extensions in the dependency graph. Then, the extensions are moved down. This concept essentially turns the process into a two-step, move-down process.

Ext Z is used just as a temporary extension for moving ownership. So, it only includes schema objects. As a result, customers can't run on the tenant until both steps have been done.

> **NOTE**
> You can only use this process for on-premise solutions.

Prerequisite

If you're moving enum type fields, then your solution must be running on Business Central 2020 release wave 1, version 16.5 or later. For more information, see Known Issues.

Create transition extension (Ext Z v1)

The transition extension will contain replicas of all table object definitions in the releasing extension, except logic code. In the illustration, these objects include **TableA**, **TableB**, and **TableC** and current their field definitions. The transition extension is **Ext Z**.

1. Create an AL project for the transition extension.
2. Add a table object that exactly matches the table object definitions for **TableA**, **TableB**, and **TableC** in the releasing extension.
3. Compile the extension package.
4. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

For purposes of the example, the ID is `11111111-aaaa-2222-bbbb-333333333333`. The value for your extension will be different.

Create empty version of releasing extension (Ext X v2)

In this step, you create a new version of the releasing extension that doesn't contain any objects. It only contains a `migration.json` file that points to **Ext Z**.

1. In the releasing extension AL project, add a migration.json file that points to the ID of the transition extension **Ext Z**.

   ```
   {
   "apprules": [
       {
           "id": "11111111-aaaa-2222-bbbb-333333333333"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.

2. Delete all objects from the extension. The objects include **TableA**, **TableB**, and **TableC**.

3. In the app.json file, increase the `version` value. Ensure that `"target": "OnPrem"`.

4. Compile a new version of the extension package.

Create receiving extension (Ext Y v1)

You now create a new extension that contains the customization you want to move from the releasing. In this example, the customizations include **TableB** and a **TableExtC**.

1. Create an AL project for **Ext Y**.

2. In the app.json file, set up a dependency on the releasing extension **Ext X**.

3. Add a table definition and code for **TableB** that exactly matches the definition in the original releasing extension.

4. Add a table extension object called **TableExtC**. Then, add a field definition for field **C-2** that matches its definition in the original **TableC** object of the releasing extension.

5. Compile the extension package.

6. Make a note of the `ID` of the new extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `44444444-cccc-5555-dddd-666666666666`. The value for your extension will be different.

Create final version of releasing extension (Ext X v3)

In this step, you create another version of the releasing extension **Ext X**. This version will contain the objects and code that you want to finally publish.

1. Create an AL project for **Ext X**.

2. Add a table definition and code for **TableA** that exactly matches the definition in the original releasing extension.

3. Add a table object for **TableC** and field definition for **C-1** that matches the definitions in the original

TableC object of the releasing extension.

4. In the app.json file, increase the `version` value.
5. Compile the extension package.
6. Make a note of the `ID` of the extension. You'll use this ID in the next task.

   For purposes of the example, the ID is `77777777-eeee-8888-ffff-999999999999`. The value for your extension will be different.

Create new empty version of transition extension (Ext Z v2)

In this step, you create a new version of **Ext Z** that only contains a `migration.json` file. This `migration.json` file points the IDS of **Ext X** and **Ext Y**. The file is used to release ownership.

1. In the extension AL project, add a migration.json file that points to the IDs of the releasing extension **Ext X** and receiving extension **Ext Y**.

   ```
   {
   "apprules": [
       {
           "id": "77777777-eeee-8888-ffff-999999999999"
       }
       {
           "id": "44444444-cccc-5555-dddd-666666666666"
       }
   ]
   }
   ```

   For more information, see The Migration.json File.

2. Delete the object definitions for **TableA**, **TableB**, and **TableC**.
3. In the app.json file, increase the `version` value.
4. Compile the extension package.

Deploy the extensions

1. Uninstall the current version of the releasing extension **Ext X**.
2. Complete the following steps for the first stage of deployment:

   a. Publish the transition extension **Ext Z** and empty version of **Ext X v2**.

   b. Synchronize the transition extension **Ext Z**.

      This step creates empty database tables **TableA**, **TableB**, and **TableC** that are owned by **Ext Z**.

      > **IMPORTANT**
      > Extensions receiving table objects must be synced first. Extension releasing/giving away table objects must be synced last.

   c. Synchronize the releasing extension **Ext X v2**.

      This step will read the migration.json of the extension. Then transfer ownership of the original tables **TableA**, **TableB**, and **TableC** to **Ext Z**.

3. Complete the following steps for the second stage of deployment:

    a. Publish the next version for **Ext Z v2** and **Ext X v3**, and the first version of **Ext Y**.

    b. Synchronize the extensions in the following order: **Ext X v3**, **Ext Y v1**, and **Ext Z v2**.

    Synchronize **Ext Z v2** last. When you synchronize **Ext Z v2**, ownership of the tables is transferred from **Ext Z** to **Ext X** and **Ext Y**.

4. Run Start-NAVAppDataUpgrade cmdlet on the new releasing extension version**Ext X v3**.

    This step basically installs the new extension version. You run a data upgrade because an earlier version has been installed and is still published.

5. Install the new receiving extension **Ext Y v1**.

6. Unpublish both versions of **Ext Z**.

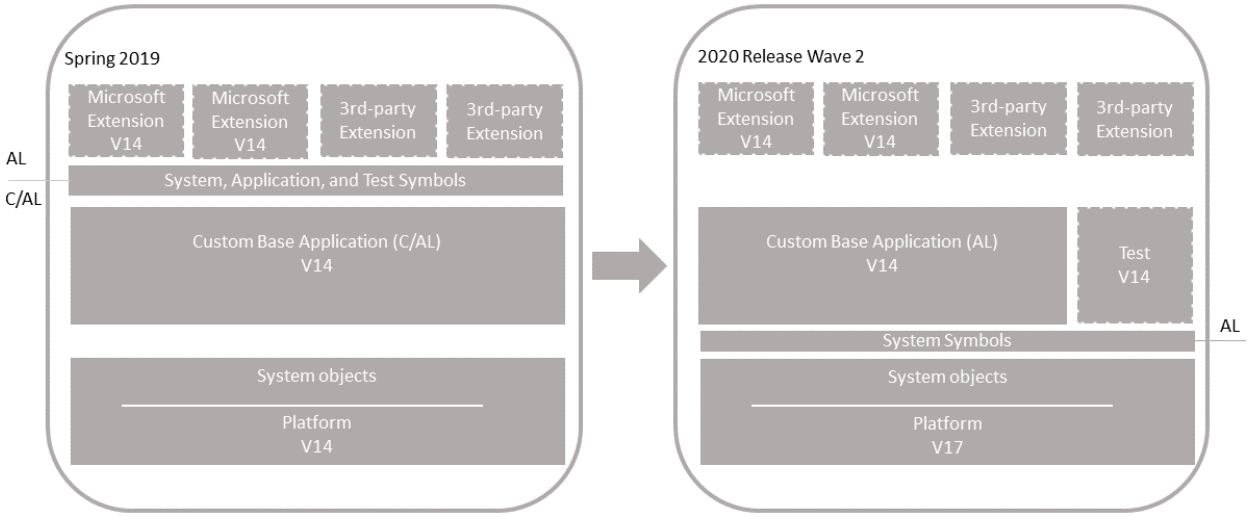
See Also

Migrating Tables and Fields Between Extensions
Moving Tables and Fields to Extension Down the Dependency Graph
Upgrading Extensions
Publishing and Installing an Extension
JSON Files

Technical Upgrade From Version 14 to Version 17

2/17/2021 • 11 minutes to read • Edit Online

Use this process when you have a code customized Business Central application (version 14) that you want to upgrade to the Business Central 2020 release wave 2 platform (version 17). This process won't upgrade the application to the latest version. You'll convert the entire application from C/AL to an AL base application extension.



Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Upgrade to Business Central Spring 2019 (version 14).

   There are several updates for version 14. When upgrading from Business Central Fall 2018 (version 13) or Dynamics NAV, upgrade to the latest version 14 update that has a compatible version 17 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix.

   If your solution is already on version 14, then you don't have to upgrade to the latest version 17 update.

   To download the latest update, go to Released Cumulative Updates for Microsoft Dynamics 365 Business Central Spring 2019 Update on-premises.

   For information, see Upgrading to Dynamics 365 Business Central On-Premises.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 17

1. Before you install version 17, it can be useful to create desktop shortcuts to the version 14.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 17.0 tools.

2. Install all components of version 17.

   You'll have to keep version 14.0 installed, because you'll need the it to complete the C/AL to AL conversion in this process. Therefore, when you install version 17, you must either specify different port numbers for components during installation or stop the version 14.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

3. Copy the version 14 **CodeViewer** add-in to the version 17.0 server installation

   a. Find the **CodeViewer** folder in the **Add-ins** folder of the version 14 RoleTailored client installation. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\140\RoleTailored Client\Add-ins.

   b. Copy the folder to the **Add-ins** folder of the version 17 server installation. By default, the folder path is C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins. Replace the existing folder and files, if any.

   CodeViewer is no longer used in version 17. But it's required because of references that exist in the converted application. If you omit this step, you might get compilation errors later.

Task 2: Convert your v14 C/AL application to AL

For more information, see Code Conversion from C/AL to AL.

Task 3: Prepare databases

In this task, you prepare the application and tenant databases for the upgrade.

1. Make backup of the database.

2. Make sure that you have the extension packages for all published extensions.

   You'll need these packages later to publish and install the extensions again.

3. Uninstall all extensions from the old tenants.

   Run the Business Central Administration Shell for version 14.0 as an administrator. Use the Uninstall-NAVApp cmdlet to uninstall an extension. For example, together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -Tenant <tenant ID> | % { Uninstall-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version -Tenant <tenant ID>}
   ```

   If you have a single tenant deployment, you can omit the `-Tenant` parameter and value.

4. Unpublish all extensions from the application server instance.

   To unpublish extensions, use the Unpublish-NAVAPP cmdlet. Together with the Get-NAVAppInfo cmdlet,

you can uninstall all extensions from the tenant using a single command:

```
Get-NAVAppInfo -ServerInstance <BC14 server instance> | % { Unpublish-NAVApp -ServerInstance <BC14
server instance> -Name $_.Name -Version $_.Version }
```

5. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet. You can unpublish all symbols by using the Get-NAVAppInfo cmdlet with the `-SymbolsOnly` switch as follows:

   ```
   Get-NAVAppInfo -ServerInstance <BC14 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC14 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

6. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <BC14 server instance> -Tenant <tenant ID>
   ```

7. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <BC14 server instance>
   ```

Task 4: Convert version 14.0 application database

This task runs a technical upgrade on the application database. A technical upgrade converts the database from the version 14.0 platform to the version 17.0 platform. This conversion updates the system tables of the database to the new schema (data structure). It also provides the latest platform features and performance enhancements.

1. Start Business Central Administration Shell for version 17.0 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion:

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server>\<database instance> -
   DatabaseName "<BC14 database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (14-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 5: Configure version 17 server

When you installed version 17 in **Task 1**, a version 17 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 14 to version 17.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <BC17 server instance> -KeyName DatabaseName -KeyValue "
   <BC14 database name>"
   ```

   In a single tenant deployment, this command mounts the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <BC17 server instance> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <BC17 server instance>
   ```

Task 6: Upload Business Central partner license

If you have a new Business Central partner license, make sure that it has been uploaded to the database. To upload the license, use the Import-NAVServerLicense cmdlet:

```
Import-NAVServerLicense -ServerInstance <BC17 server instance> -LicenseFile "<path to the license>"
```

For more information, see Uploading a License File for a Specific Database.

Task 7: Publish system symbols, base application, and test library extensions

In this task, you'll publish extensions to the version 17.0 server instance. Publishing adds the extension to the application database that is mounted on the server instance. The extension is then available for installing on tenants later. It updates internal tables, compiles the components of the extension behind-the-scenes, and builds the necessary metadata objects that are used at runtime.

The steps in this task continue to use the Business Central Administration Shell for version 17.0 that you started in the previous task.

1. Publish the system symbols extension for version 17.

   The symbols extension contains the required platform symbols that the base application depends on. The symbols extension package is called **System.app**. You find it where the **AL Development Environment** is installed. The default installation path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

   ```
   Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to the System.app file>" -
   PackageType SymbolsOnly
   ```

2. Publish the custom base application extension that you created in **Task 2**.

```
Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to the base application extension
package file>"
```

3. Publish the test library extension if you created one in **Task 2**.

```
Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to the test library extension
package file>"
```

Task 8: Synchronize tenant

This task updates the tenant database schema with schema changes in system objects and application objects.

If you have a multitenant deployment, run these steps for each tenant (replacing `<tenant ID>` with the appropriate tenant ID).

1. (Multitenant only) Mount the tenant to the version 17 server instance.

   To mount the tenant, use the Mount-NAVTenant cmdlet:

```
Mount-NAVTenant -ServerInstance <BC17 server instance> -DatabaseName "<BC14 database name>" -
DatabaseServer <database server>\<database instance> -Tenant <tenant ID> -AllowAppDatabaseWrite
```

> **IMPORTANT**
> You must use the same tenant ID for the tenant that was used in the old deployment; otherwise you will get an error when mounting or syncing the tenant. If you want to use a different ID for the tenant, you can either use the `-AlternateId` parameter now or after upgrading, dismount the tenant, then mount it again using the new ID and the `OverwriteTenantIdInDatabase` parameter.

> **NOTE**
> For upgrade, we recommend that you use the `-AllowAppDatabaseWrite` parameter. After upgrade, you can dismount and mount the tenant again without the parameter if needed.

2. Synchronize the tenant to the system objects.

   Use the Sync-NAVTenant cmdlet:

```
Sync-NAVTenant -ServerInstance <BC17 server instance> -Tenant <tenant ID> -Mode Sync
```

   With a single-tenant deployment, you can omit the `-Tenant` parameter and value.

   At this stage, the tenant state is **Operational**.

Task 9: Install base and test applications

You'll synchronize the tenant to the custom base application and test library extension (if any). When you synchronize the tenant, extensions take ownership of the tables in the SQL Database.

If you have a multitenant deployment, run these steps for each tenant (replacing `<tenant ID>` with the appropriate tenant ID).

1. Synchronize the tenant to the base application extension (Base Application).

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <BC17 server instance> -Name "Base Application" -Version <extension
   version> -tenant <tenant ID>
   ```

   With this step, the base app takes ownership of the database tables. When completed, in SQL Server, the table names will be suffixed with the base app extension ID.

2. Install custom base application extension on the tenant.

   To install the extension, you use the Install-NAVApp cmdlet.

   ```
   Install-NAVApp -ServerInstance <BC17 server instance> -Name "Base Application" -Version <extension
   version>
   ```

   At this point, the base application is upgraded to the version 17 platform and is operational. You can open the application in the client.

3. Synchronize and install the test library extension.

   This step is like what you did for the custom base application in steps 1 and 2.

Task 10: Publish and install extensions

Now, you can install the Microsoft and 3rd-party extensions that were installed on the tenant before the upgrade.

1. Publish the old extension versions.

   ```
   Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to extension package file>"
   ```

   You only need to do this step once.

2. Synchronize the extension.

   ```
   Sync-NAVApp -ServerInstance <BC17 server instance> -Name "<extension name>" -Version <extension
   version> -tenant <tenant ID>
   ```

3. Install the extension:

   ```
   Install-NAVApp -ServerInstance <BC17 server instance> -Name "<extension name>" -Version <extension
   version> -tenant <tenant ID>
   ```

4. Repeat steps 2 and 3 for each extension and on each tenant.

Now, your application is fully upgraded to the version 17 platform.

Task 11: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins from the client, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is `C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`.
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Task 12: Post-upgrade

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount

them again without using the `-AllowAppDatabaseWrite` parameter.

3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

4. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 17 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - Export the **EXCEL EXPORT ACTION** permission set. Then, import it to your application and add it to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate permission sets.

   For more information about working with permission sets and permissions, see Export and Import Permission Sets.

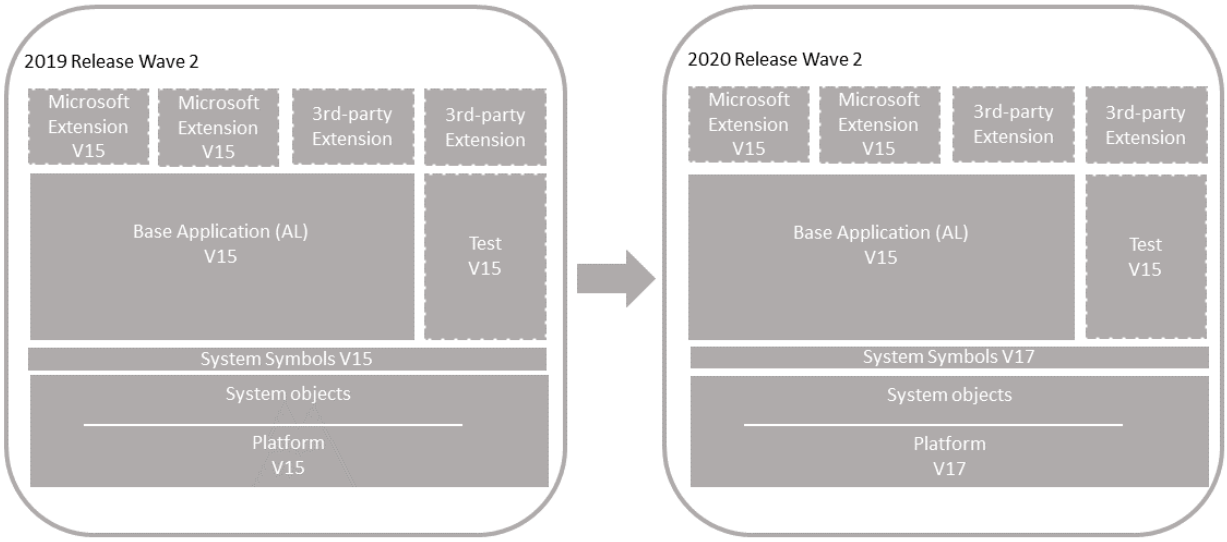
See Also

Upgrading to Business Central
Business Central 14.X to 15.X compatibility matrix

Technical Upgrade from Version 15 to Version 17

2/17/2021 • 10 minutes to read • Edit Online

Use this process to upgrade from Business Central 2019 release wave 2 (version 15) to the Business Central 2020 release wave 1 platform (version 17). This process won't upgrade the application to the latest version.



Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Considerations

In version 17, a number of tables have been deprecated and replaced by new tables compared to version 15. For a list of these tables and the corresponding new tables, see Deprecated Tables. Code that uses the deprecated table or tables, must be rewritten to use the tables. In this article, this work is done under Task 2.

Prerequisites

1. Your version 15 platform is compatible with version 17.

   There are several updates for version 15. The updates have a compatible version 17 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix. For example, if your solution is currently running 15.11, you can't upgrade to 17.0. You must wait until 17.1 is available.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Install version 17

1. Before you install version 17, it can be useful to create desktop shortcuts to the version 15.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 17.0 tools.

2. Install version 17 components.

   You'll keep version 15 component installed for now. When you install version 17, you must either specify different port numbers for components during installation or stop the version 15.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

APPLICATION

Task 2: Rewrite code to handle obsoleted system tables

In version 17, several tables have been deprecated and replaced by new tables, compared to version 15. For a list of these tables and the corresponding new tables, see Deprecated Tables. Code that uses the deprecated tables, must be rewritten to use the tables. This change will typically affect your base application or the Microsoft System Application, if you're using it.

For the base application or system application extensions, you'll have to create a new version that uses the new tables. The basic steps are as follows:

1. Create AL project in Visual Studio Code for the system and/or base application.

   If you're using the Microsoft System Application, start with this project first.

2. Include the source files of the current version in the project.

3. Copy the version 17 System symbols (System.app) file to the **.alpackages** folder of the project.

   You'll find the System.app file on the installation media (DVD) for version 17 or in the **AL Development Environment** installation folder. By default, the folder is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

   For the base application, also include the extension package (.app) for the new version of the Microsoft System Application, if you're using it.

4. Modify the app.json:

   - Increase `"version"` number. You have to increase the version so you can run a data upgrade later in this process.
   - Set `"runtime"` to `6.0`
   - Set `"platform"` to `17.0.0.0`
   - Set `"target"` to `OnPrem`

   For more information about the app.json file, see App.json file.

5. In the **dotnet.al** files in the project, find and delete all instances of `Version = '15.0.0.0';` in **Microsoft.Dynamics.Nav** and **Microsoft.Dynamics.Framework** assembly declarations.

6. Rewrite code that references the deprecated table to reference the new tables.

   Try to build the project first to see what errors you get. Then, resolve the errors.

7. Compile and build the new version of the extension.

DATA

Task 3: Prepare databases

In this task, you prepare the application and tenant databases for the upgrade.

1. Make backup of the database.
2. Make sure that you have the extension packages for all published extensions.

   You'll need these packages later to publish and install the extensions again.
3. (Single-tenant only) Uninstall all extensions from the old tenants.

   Run the Business Central Administration Shell for version 15.0 as an administrator. Use the Uninstall-NAVApp cmdlet to uninstall an extension. For example, together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance> | % { Uninstall-NAVApp -ServerInstance <BC15
   server instance> -Name $_.Name -Version $_.Version }
   ```
4. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet. You can unpublish all symbols by using the Get-NAVAppInfo cmdlet with the `-SymbolsOnly` switch as follows:

   ```
   Get-NAVAppInfo -ServerInstance <BC15 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC15 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?
5. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <BC15 server instance> -Tenant <tenant ID>
   ```
6. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <BC15 server instance>
   ```

Task 4: Convert the version 15.0 application database

This task runs a technical upgrade on the application database. A technical upgrade converts the database from the version 15.0 platform to the version 17.0 platform. This conversion updates the system tables of the database to the new schema (data structure). It also provides the latest platform features and performance enhancements.

> **IMPORTANT**
> The conversion does not modify the application objects, but it will remove any modifications that you have made to system tables. After the conversion you will no longer be able to use it with Business Central 14.

1. Start Business Central Administration Shell for version 17.0 as an administrator.

2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion. In a multitenant deployment, run this cmdlet against the application database.

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server>\<database instance> -
DatabaseName "<BC15 database name>"
```

When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (15-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

Task 5: Configure version 17 server

When you installed version 17 in **Task 1**, a version 17 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 15 to version 17.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

```
Set-NAVServerConfiguration -ServerInstance <BC17 server instance> -KeyName DatabaseName -KeyValue "
<BC15 database name>"
```

In a single tenant deployment, this command mounts the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task scheduler on the server instance for purposes of upgrade.

```
Set-NavServerConfiguration -ServerInstance <BC17 server instance> -KeyName "EnableTaskScheduler" -
KeyValue false
```

Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <BC17 server instance>
```

Task 6: Import version 17 license

1. Use the Import-NAVServerLicense to upload the version 17 license to the database.

```
Import-NAVServerLicense -ServerInstance <server instance name> -LicenseFile <path and file name>
```

2. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance name>
```

Task 7: Publish new system symbols

Use the Publish-NAVApp cmdlet to publish the new symbols extension package. This package is called **System.app**. If you've installed the **AL Development Environment**, you find the file in the installation folder. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

```
Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to the System.app file>" -PackageType
SymbolsOnly
```

Task 8: Publish new extension versions

Publish the new versions of the system and base application extensions.

```
Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to .app file for system or base
application>"
```

Task 9: Recompile published extensions

Compile all other published extensions, except the Microsoft System Application and you're base application, against the new platform.

1. To compile an extension, use the Repair-NAVApp cmdlet, For example:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   To compile all published extensions at once, you can use this command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Task 10: Synchronize tenant

1. (Multitenant only) Mount the tenant to the new Business Central Server instance.

   You'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

2. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID> -Mode Sync
   ```

   For a single-tenant deployment, you can either set the `<tenant ID>` to `default` or omit the `-Tenant <tenant ID>` parameter. For more information about syncing, see Synchronizing the Tenant Database and Application Database.

3. Synchronize the system application extension, then the base application.

```
Sync-NAVApp -ServerInstance <server instance> -Name <extension name> version <extension version> -
Tenant <tenant ID>
```

Task 11: Run upgrade to install system and base applications

Instead of directly installing the system amd base applications, you'll have to run a data upgrade because previous versions of these extensions were installed.

To do this, run the Start-NAVAppDataUpgrade cmdlet on each extension. Start with the system application, then the base application.

```
Start-NAVAppDataUpgrade -ServerInstance <server instance> -Name <extension name> version <extension version>
-Tenant <tenant ID>
```

This will run the data upgrade and install the extensions on the tenant.

Task 12: Reinstall other extensions

In this step, you install the remaining extensions that were previously installed on your tenant before upgrading. To install an extension, you use the Install-NAVApp cmdlet.

```
Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version <extension version>
```

If your previous tenant including the **Application** extension, install this extension first.

Task 13: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is

   ```
   C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-
   ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip
   ```

   .
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Task 14: Post-upgrade

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.
4. Grant users permission to the *Open in Excel* and *Edit in Excel* actions.

   Version 17 introduces a system permission that protects these two actions. The permission is granted by the system object **6110 Allow Action Export To Excel**. Because of this change, users who had permission to these actions before upgrading, will lose permission. To grant permission again, do one of the following steps:

   - If you have a version 17 application, export the **EXCEL EXPORT ACTION** permission set. Then, import it to your application and add it to appropriate users.
   - Add the system object **6110 Allow Action Export To Excel** permission directly to appropriate

permission sets.

For more information about working with permission sets and permissions, see Export and Import Permission Sets.

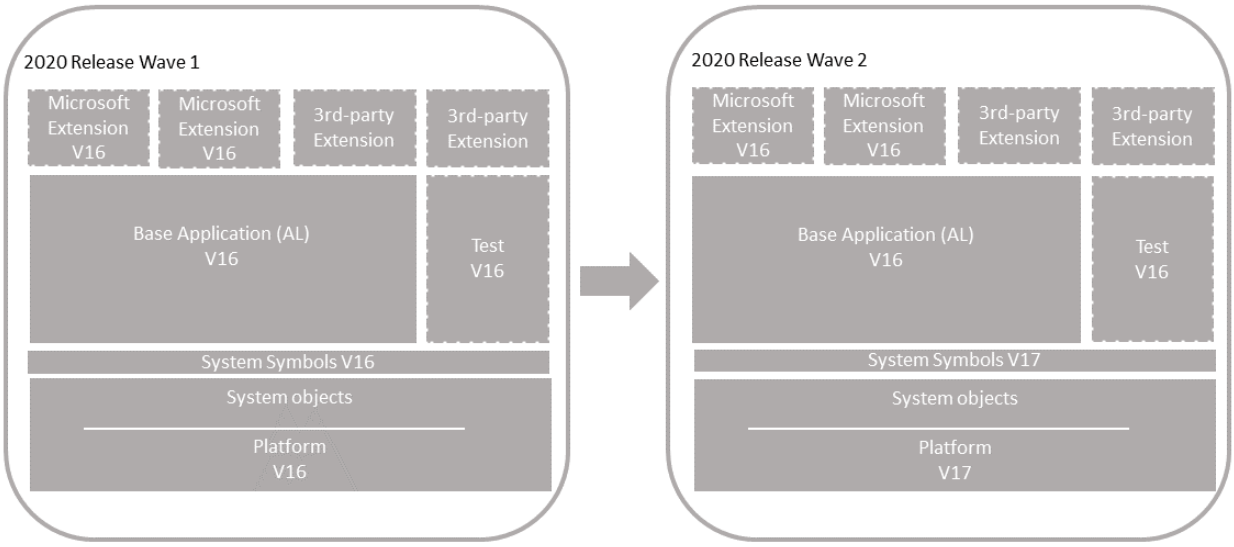
See Also

Upgrading to Business Central
Business Central Compatibility Matrix

Technical Upgrade from Version 16 to Version 17

2/17/2021 • 8 minutes to read • Edit Online

Use this process to upgrade from Business Central 2020 release wave 1 (version 16) to the Business Central 2020 release wave 2 platform (version 17). This process won't upgrade the application to the latest version.



Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Prerequisites

1. Your version 16 platform is compatible with version 17.

   There are several updates for version 16. The updates have a compatible version 17 update. For more information, see Dynamics 365 Business Central Upgrade Compatibility Matrix. For example, if your solution is currently running 16.6, you can't upgrade to 17.0. You must wait until 17.1 is available.

2. Disable data encryption.

   If the current server instance uses data encryption, disable it. You can enable it again after upgrading.

   For more information, see Managing Encryption and Encryption Keys.

   Instead of disabling encryption, you can export the current encryption key, which you'll then import after upgrade. However, we recommend disabling encryption before upgrading.

Task 1: Prepare databases

In this task, you prepare the application and tenant databases for the upgrade.

1. Make backup of the database.

3. (Single-tenant only) Uninstall all extensions from the old tenants.

   Run the Business Central Administration Shell for version 16.0 as an administrator. Use the Uninstall-NAVApp cmdlet to uninstall an extension. For example, together with the Get-NAVAppInfo cmdlet, you can uninstall all extensions with a single command:

   ```
   Get-NAVAppInfo -ServerInstance <BC16 server instance> | % { Uninstall-NAVApp -ServerInstance <BC16
   server instance> -Name $_.Name -Version $_.Version }
   ```

4. Unpublish all system, test, and application symbols.

   To unpublish symbols, use the Unpublish-NAVAPP cmdlet. You can unpublish all symbols by using the Get-NAVAppInfo cmdlet with the `-SymbolsOnly` switch as follows:

   ```
   Get-NAVAppInfo -ServerInstance <BC16 server instance> -SymbolsOnly | % { Unpublish-NAVApp -
   ServerInstance <BC16 server instance> -Name $_.Name -Version $_.Version }
   ```

   What are symbols?

5. (Multitenant only) Dismount the tenants from the application server instance.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

   ```
   Dismount-NAVTenant -ServerInstance <BC16 server instance> -Tenant <tenant ID>
   ```

6. Stop the server instance.

   ```
   Stop-NAVServerInstance -ServerInstance <BC16 server instance>
   ```

Task 2: Install version 17

1. Before you install version 17, it can be useful to create desktop shortcuts to the version 16.0 tools, such as the Business Central Server Administration tool, Business Central Administration Shell, and Dynamics NAV Development Shell because the Start menu items for these tools will be replaced with the version 17.0 tools.

2. Install version 17 components.

   You'll keep version 16 installed for now. When you install version 17, you must either specify different port numbers for components (like the Business Central Server instance and web services) or stop the version 16.0 Business Central Server instance before you run the installation. Otherwise, you'll get an error that the Business Central Server failed to install.

   For more information, see Installing Business Central Using Setup.

Task 3: Copy Dynamics Online Connect add-in

The Dynamics Online Connect add-in has been deprecated in version 17. As a result, it has been removed from the DVD and is no longer installed as part of the Business Central Server. However, when doing a technical upgrade, it is still required for the old System Application. To meet this requirement, copy the **Add-ins\Connect** folder of the version 16 server installation to the **Add-ins** folder of the version 17 server installation.

Task 4: Convert the version 16.0 application database

This task runs a technical upgrade on the application database. A technical upgrade converts the database from the version 16.0 platform to the version 17.0 platform. This conversion updates the system tables of the database to the new schema (data structure). It also provides the latest platform features and performance enhancements.

> **IMPORTANT**
> The conversion does not modify the application objects, but it will remove any modifications that you have made to system tables. After the conversion you will no longer be able to use it with Business Central 14.

1. Start Business Central Administration Shell for version 17.0 as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the conversion. In a multitenant deployment, run this cmdlet against the application database.

   ```
   Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server>\<database instance> -
   DatabaseName "<BC16 database name>"
   ```

   When completed, a message like the following displays in the console:

   ```
   DatabaseServer      : .\BCDEMO
   DatabaseName        : Demo Database BC (16-0)
   DatabaseCredentials :
   DatabaseLocation    :
   Collation           :
   ```

Task 5: Configure version 17 server

When you installed version 17 in **Task 1**, a version 17 Business Central Server instance was created. In this task, you change server configuration settings that are required to complete the upgrade. Some of the changes are only required for version 16 to version 17.0 upgrade and can be reverted after you complete the upgrade.

1. Set the server instance to connect to the application database.

   ```
   Set-NAVServerConfiguration -ServerInstance <BC17 server instance> -KeyName DatabaseName -KeyValue "
   <BC17 database name>"
   ```

   In a single tenant deployment, this command mounts the tenant automatically. For more information, see Connecting a Server Instance to a Database.

2. Disable task scheduler on the server instance for purposes of upgrade.

   ```
   Set-NavServerConfiguration -ServerInstance <BC17 server instance> -KeyName "EnableTaskScheduler" -
   KeyValue false
   ```

   Be sure to re-enable task scheduler after upgrade if needed.

3. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <BC17 server instance>
   ```

Task 6: Upload Business Central partner license

If you have a new Business Central partner license, make sure that it has been uploaded to the database. To upload the license, use the Import-NAVServerLicense cmdlet:

```
Import-NAVServerLicense -ServerInstance <BC17 server instance> -LicenseFile "<path to the license>"
```

For more information, see Uploading a License File for a Specific Database.

Task 7: Publish new system symbols

Use the Publish-NAVApp cmdlet to publish the new symbols extension package. This package is called **System.app**. If you've installed the **AL Development Environment**, you find the file in the installation folder. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment.

```
Publish-NAVApp -ServerInstance <BC17 server instance> -Path "<path to the System.app file>" -PackageType
SymbolsOnly
```

Task 8: Recompile published extensions

Compile all published extensions against the new platform.

1. To compile an extension, use the Repair-NAVApp cmdlet, For example:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   To compile all published extensions at once, you can use this command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Task 9: Synchronize tenant

1. (Multitenant only) Mount the tenant to the new Business Central Server instance.

   You'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

2. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID> -Mode Sync
   ```

   For a single-tenant deployment, you can either set the `<tenant ID>` to `default` or omit the `-Tenant <tenant ID>` parameter. For more information about syncing, see Synchronizing the Tenant Database and Application Database.

Task 10: Reinstall extensions

Skip this task for a multitenant environment. In this task, you reinstall the same extensions that were installed on the tenant before.

To install an extension, you use the Install-NAVApp cmdlet.

1. If your solution uses the System Application, install this first.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

2. Install the Base Application.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Base Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

3. Install the Application extension.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Application extension.

   For more information about the Application extension, see The Microsoft_Application.app File.

4. Install other extensions, including Microsoft and third-party extensions.

   ```
   Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
   ```

At this point, your solution has been updated to the latest platform.

Task 11: Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins, do the following steps:

1. Open the Business Central client.
2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is `C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`.

5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Task 12: Post-upgrade

1. Enable task scheduler on the server instance.
2. (Multitenant only) For tenants other than the tenant that you use for administration purposes, if you mounted the tenants using the `-AllowAppDatabaseWrite` parameter, dismount the tenants, then mount them again without using the `-AllowAppDatabaseWrite` parameter.
3. If you want to use data encryption as before, enable it.

   For more information, see Managing Encryption and Encryption Keys.

   Optionally, if you exported the encryption key instead of disabling encryption earlier, import the encryption key file to enable encryption.

See Also

Upgrading to Business Central
Business Central Compatibility matrix

Installing a Business Central 2020 Release Wave 2 Update

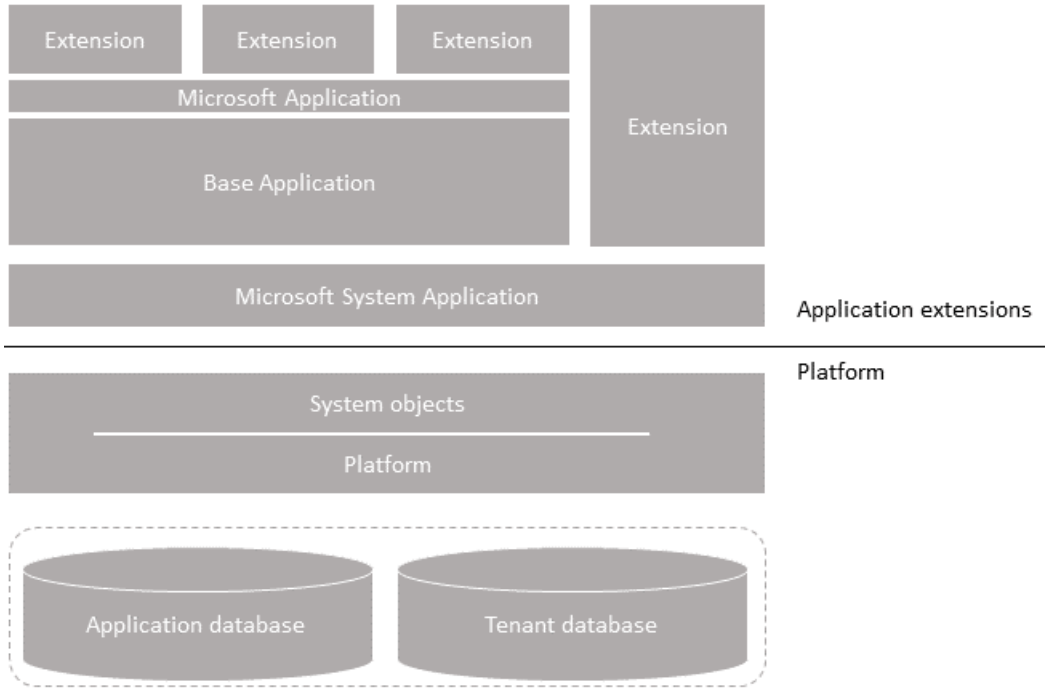
2/17/2021 • 17 minutes to read • Edit Online

This article describes how to install an update for Business Central on-premises. An update is a set of files that includes all hotfixes and regulatory features that have been released for Business Central.

You can choose to update only the platform or both the platform and application code. The installation guidelines are separated into PLATFORM tasks and APPLICATION tasks.

Overview

The following figure provides a high-level representation of a Business Central solution and the components that are involved in the installation of an update.



The databases store the application metadata and business data. If you have a single-tenant deployment, this data is stored in a single database. A multitenant deployment stores the application metadata in the application database and the business data in one or more tenant databases.

Application stack

The application includes AL extensions that define the objects and code that makes up the business logic. For example, objects include tables, report, pages, codeunits, and more. Each extension is compiled and delivered as an .app file, which is published to the Business Central Server instance.

- System Application extension

  The Microsoft System Application extension includes functionality that isn't directly related the business logic. For more information, see Overview of the System Application. When using the Microsoft Base Application, your solution uses the System Application. With a custom Base Application, your solution may or may not use the System Application. If it doesn't, you can skip any steps in this article related to the System Application.

- Base application extension

  As a minimum, the solution always includes the Base Application. The Base Application contains the objects (such as table, pages, codeunits, and reports) that define the business logic and functionality of the solution. The Base Application can be either the Microsoft Base Application or a customized Base Application. The Microsoft Base Application is the standard application that is on the installation media (DVD). A customized Base Application is an application that includes customized code.

- Application extension

  The Application extension logically encapsulates all of the extensions making up a solution, for example, version `16.0.0.0` of the base and system application package files, and it provides a convenient way to define and refer to this solution identity. This extension is required for Microsoft extensions, but is optional for third-party extensions. For more information, see The Microsoft_Application.app File.

- Customization extensions

  Customization extensions add functionality and features to the Base Application or System Application. Extensions can be either Microsoft extensions or third-party extensions. Microsoft extensions are available on the DVD. Third-party extensions are extensions developed by your organization or by another organization, like an ISV.

Single-tenant and multitenant deployments

The process for upgrading is similar for a single-tenant and multitenant deployment. However, there are some inherent differences. With a single-tenant deployment, the application and business data are included in the same database. While with a multitenant deployment, application code is in a separate database (the application database) than the business data (tenant). In the procedures that follow, for a single-tenant deployment, consider references to the *application database* and *tenant database* as the same database. Steps are marked as *Single-tenant only* or *Multitenant only* where applicable.

Platform versus application update

A platform update doesn't change the application. It involves converting your databases to the new platform and recompiling the existing extensions to ensure that they're compatible with the new platform.

An application update involves:

- Publishing new versions of extensions that include the latest application modifications
- Synchronizing the databases with any schema change introduced by the new extensions
- Updating affected data.

The installation media (DVD) includes new versions of Microsoft's Base Application, System Application, and extensions. The DVD also includes the AL source code for the Microsoft Base Application. This code useful if you have a custom base application. You can use the code to compare and merge updates into your application. You'll only have to recompile third-party extensions that you don't have a new version to publish.

PREPARATION

Download update package

The first thing to do is to download the update package that matches your Business Central solution.

1. Go to the list of available updates for your on-premises version of Business Central. Then, choose the update that you want.

2. From the update page, under the **Resolution** section, select the link for downloading the update, and follow the instructions.

3. On the computer where you downloaded the update .zip file, extract the all the files to a selected location.

   When extracted, the update includes the DVD folder. This folder contains the full Business Central product. For example, the folder includes the Business Central installation program (setup.exe), tools for upgrading to the platform, and the Microsoft extensions.

When this step is completed, you can continue to update your Business Central solution to the new platform and application.

Prepare existing databases

1. Back up your databases.
2. Run the Business Central Administration Shell as an administrator.
3. (Single-tenant only) Uninstall all extensions from the all tenants.

   In this step, you uninstall the Base Application, System Application (if used), and any other extensions that are currently installed on the database.

   a. Get a list of installed extensions.

      This step is optional, but it can be useful to the names and versions of the extensions.

      To get a list of installed extensions, use the Get-NAVAppInfo cmdlet.

      ```
      Get-NAVAppInfo -ServerInstance <server instance name>
      ```

   b. Uninstall the extensions.

      To uninstall an extension, you use the Uninstall-NAVApp cmdlet.

      ```
      Uninstall-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extensions
      name> -Version <extension version> -Force
      ```

      Replace `<extension name>` and `<extension version>` with the exact name and version the installed extension. For single-tenant deployment, set `<tenant ID>` to `default` or omit the `-Tenant` parameter.

      For example, together with the Get-NAVApp cmdlet, you can uninstall all extensions with a single command:

      ```
      Get-NAVAppInfo -ServerInstance <server instance name> -Tenant <tenant ID> | % { Uninstall-
      NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name $_.Name -Version
      $_.Version -Force}
      ```

4. Unpublish the existing system symbols.

   To unpublish the system symbols, use the Unpublish-NAVApp cmdlet as follows:

   ```
   Unpublish-NAVApp -ServerInstance <server instance> -Name System -version <version>
   ```

   What are symbols?.

5. (Multitenant only) Dismount the tenants from the application database.

   To dismount a tenant, use the Dismount-NAVTenant cmdlet:

```
Dismount-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID>
```

Install Business Central components

From the installation media (DVD), run setup.exe to uninstall the current Business Central components and install the Business Central components included in the update.

1. Stop the Business Central Server instance.

```
Stop-NAVServerInstance -ServerInstance <server instance>
```

2. Run setup.exe to uninstall your current version of Business Central.
3. Run setup.exe again to install components of the update.
   a. Follow setup pages until you get to the **Microsoft Dynamics 365 Business Central Setup** page.
   b. Select **Advanced installation options** > **Choose an installation option** > **Custom**.
   c. On the **Customize the installation** page, select the following components as a minimum:
      - AL Development Environment (optional but recommended)
      - Server
      - Web Server Components.
   d. Select **Next**.
   e. On the **Specify parameters** page, set the fields as needed.

      IMPORTANT
      Clear the **SQL Database** field so that it is blank. At this time, do not set this to the database that you want to update; otherwise, the installation of the Business Central Server will fail. You will connect the database to the Business Central Server later after it is converted to the new platform.

   f. Select **Apply** to complete the installation.

For more information, see Installing Business Central Using Setup.

PLATFORM UPDATE

Follow the next few tasks to convert your database to the new platform of the update. A multitenant deployment includes the application and tenant databases. The conversion updates the system tables of the database to the new schema (data structure) and provides the latest platform features and performance enhancements.

Also, to ensure that the existing published extensions work on the new platform, you'll recompile the extensions.

Convert existing database to new platform

1. Run the Business Central Administration Shell as an administrator.
2. Run the Invoke-NAVApplicationDatabaseConversion cmdlet to start the database conversion to the new platform.

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer <database server name>\<database server
instance> -DatabaseName "<database name>" [-Force]
```

For example, in a single tenant deployment:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer .\BCDEMO -DatabaseName "Demo Database BC (17-
0)"
```

In a multitenant deployment, run this cmdlet against the application database and use the `-Force` parameter. For example:

```
Invoke-NAVApplicationDatabaseConversion -DatabaseServer .\BCDEMO -DatabaseName "BC17 Application" -
Force
```

When completed, a message like the following displays in the console:

```
DatabaseServer      : .\BCDEMO
DatabaseName        : Demo Database BC (17-0)
DatabaseCredentials :
DatabaseLocation    :
Collation           :
```

NOTE

Depending on the update that you're installing, you might get a message similar to the following:

```
Invoke-NAVApplicationDatabaseConversion : A technical upgrade of database <database name> on
server '.\<database instance>' cannot be run, because the database's application version 'NNNNNN'
is greater than or equal to the platform version 'NNNNNN'
```

This is not an error, and you can continue installing the update. This message is recorded as a warning in the event log as well. This message indicates that the application database is already compatible with the new platform, which happens when the update does not make any schema changes to the system tables.

Connect server instance to database

1. (Multitenant only) Enable the server instance as a multitenant instance:

```
Set-NAVServerConfiguration -ServerInstance <server instance> -KeyName Multitenant -KeyValue true
```

2. Connect the server instance to connect to the database.

```
Set-NAVServerConfiguration -ServerInstance <server instance> -KeyName DatabaseName -KeyValue "
<database name>"
```

In a multitenant deployment, the database is the application database. For more information, see Connecting a Server Instance to a Database.

3. Restart the server instance.

```
Restart-NAVServerInstance -ServerInstance <server instance>
```

Publish the new system symbols

Use the Publish-NAVApp cmdlet to publish the new symbols extension package. This package is called **System.app**. If you've installed the **AL Development Environment**, you find the file in the installation folder. By default, the folder path is C:\Program Files (x86)\Microsoft Dynamics 365 Business Central\170\AL Development Environment. Or, it's also on the installation media (DVD) in the ModernDev\program files\Microsoft Dynamics NAV\170\AL Development Environment folder.

```
Publish-NAVApp -ServerInstance <server instance> -Path "<path to the System.app file>" -PackageType SymbolsOnly
```

Recompile published extensions

Compile all published extensions against the new platform.

> **NOTE**
> If you plan on updating the application you can skip this step for extensions for which you have new versions built on the new platform. For example, this includes Microsoft extensions that are on the DVD.

1. To compile an extension, use the Repair-NAVApp cmdlet, For example:

   ```
   Repair-NAVApp -ServerInstance <server instance> -Name <extension name> -Version <extension name>
   ```

   To compile all published extensions at once, you can use this command:

   ```
   Get-NAVAppInfo -ServerInstance <server instance> | Repair-NAVApp
   ```

2. Restart the server instance.

   ```
   Restart-NAVServerInstance -ServerInstance <server instance>
   ```

Synchronize tenant

1. (Multitenant only) Mount the tenant to the new Business Central Server instance.

   You'll have to do this step and the next for each tenant. For more information, see Mount or Dismount a Tenant.

2. Synchronize the tenant.

   Use the Sync-NAVTenant cmdlet:

   ```
   Sync-NAVTenant -ServerInstance <server instance> -Tenant <tenant ID> -Mode Sync
   ```

   For a single-tenant deployment, you can either set the `<tenant ID>` to `default` or omit the `-Tenant <tenant ID>` parameter. For more information about syncing, see Synchronizing the Tenant Database and Application Database.

NOTE

At this point, if you want to update the application, you can skip the next step and proceed APPLICATION.

(Single-tenant only) Reinstall extensions

In this task, you reinstall the same extensions that were installed on the tenant before. If you're planning on updating the application, then skip this step.

To install an extension, you use the Install-NAVApp cmdlet.

1. If your solution uses the System Application, install this first.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "System Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

2. Install the Base Application.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Base Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

3. Install the Application extension as needed.

   ```
   Install-NAVApp -ServerInstance <server instance> -Name "Application" -Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Application extension.

4. Install other extensions, including Microsoft and third-party extensions.

   ```
   Install-NAVApp -ServerInstance <server instance name> -Name <extension name> -Version <extension version>
   ```

At this point, your solution has been updated to the latest platform.

IMPORTANT

If your solution uses any Microsoft control add-ins, you must upgrade the add-ins to the latest version. Go to Upgrade control add-ins under **Post Upgrade** section.

APPLICATION UPDATE

Follow the next tasks to update the application code to the new features and hotfixes. The tasks include publishing new versions of the System Application, Base Application, and add-on extensions.

You publish the System Application extension only if it was used in old solution. Add-on extensions include Microsoft and third- party extensions that were used in the old solution.

> **NOTE**
> If a license update is required for a regulatory feature, customers can download an updated license from CustomerSource (see How to Download a Microsoft Dynamics 365 Business Central License from CustomerSource), and partners can download their customers' updated license from VOICE (see How to Download a Microsoft Dynamics 365 Business Central Customer License from VOICE).

Upgrade System Application

Follow these steps if your existing solution uses the Microsoft System Application. Otherwise, you can skip this procedure.

1. Publish the **System Application** extension (Microsoft_System Application.app).

   You find the (Microsoft_System Application.app in the **Applications\System Application\Source** folder of installation media (DVD).

   ```
   Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_System
   Application.app>"
   ```

2. Synchronize the tenant(s) with the **System Application** extension (Microsoft_System Application):

   Use the Sync-NAVApp cmdlet:

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published System Application.

   > **TIP**
   > To get a list of all published extensions, along with their names and versions, use the Get-NAVAppInfo cmdlet.

3. Run the data upgrade on the System Application.

   To run the data upgrade, use the Start-NavAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name "System
   Application" -Version <extension version>
   ```

   Upgrading data updates the data in the tables of the tenant database to the schema changes made to tables of the System Application.

Upgrade Base Application

Microsoft Base Application

Follow these steps if your existing solution uses the Microsoft Base Application.

1. Publish the Business Central Base Application extension (Microsoft_Base Application.app).

   The **Base Application** extension contains the application business objects. You find the Microsoft_Base Application.app in the **Applications\BaseApp\Source** folder of installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<path to Microsoft_Base
Application.app>"
```

2. Synchronize the tenant with the Business Central Base Application extension (Microsoft_BaseApp):

   ```
   Sync-NAVApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base Application" -
   Version <extension version>
   ```

   Replace `<extension version>` with the exact version of the published Base Application.

   With this step, the base app takes ownership of the database tables. When completed, in SQL Server, the table names will be suffixed with the base app extension ID. This process can take several minutes.

3. Run the data upgrade on the Base Application.

   To run the data upgrade, use the Start-NavAppDataUpgrade cmdlet:

   ```
   Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name "Base
   Application" -Version <extension version>
   ```

   Upgrading data updates the data in the tables of the tenant database to the schema changes made to tables of the Base Application.

Upgrade custom Base Application

With a custom Base Application, you may want the new application features and hotfixes in the Microsoft Base Application. If so, you'll have to merge the modifications made in the Microsoft Base Application into your custom Base Application. Then, create a new version of your custom Base Application.

The source code for the new Microsoft Base Application version is in the **Base Application.Source.zip** file. This file is on the installation media (DVD), in the **Applications\BaseApp\Source** folder. You can compare this source code with the source code of the previous Microsoft Base Application and your custom application source. Then merge the code into a new custom application version.

After you've created the new version of your custom application, you publish it to the application server instance. Then, you synchronize and run the data upgrade on the tenants.

Upgrade Microsoft extensions

If your old solution used Microsoft extensions, then you upgrade these extensions to the new versions that are available on the Business Central installation media (DVD). The new versions are in the **Applications** folder, which contains a subfolder for each extension. The extension package (.app file) that you need for publishing the extension is in the **Source** folder, for example, **Applications\SalesAndInventoryForecast\Source\SalesAndInventoryForecast.app**.

The general steps for this task are listed below. For detailed steps, see Publishing, Upgrading, and Installing Extensions During Upgrade.

Publish and install Microsoft_Application extension

The Microsoft_Application extension was introduced in 15.3. In short, it's used to contain the dependency declarations on the system and base application extensions. For more information about this extension, see The Microsoft_Application.app File.

1. Publish the Microsoft_Application.app package.

   The installation media path to the extension package (.app file) is

Applications\Application\Source\Microsoft_Application.app"

```
Publish-NAVApp -ServerInstance <server instance name> -Path "<folder path>\Microsoft_Application.app"
```

2. Synchronize the tenant database with the schema changes of the Microsoft_Application extension.

```
Sync-NavApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension name> -Tenant
<tenant ID>
```

3. Upgrade the Microsoft_Application extension on your tenants.

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name Application
```

Publish and install Microsoft extensions

Complete these steps for each Microsoft extension that you want to upgrade.

1. Publish the new extension versions from the installation media (DVD).

```
Publish-NAVApp -ServerInstance <server instance name> -Path <path to extension package file>
```

2. Synchronize the tenant database with the schema changes of the extensions.

```
Sync-NavApp -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension name> -
Version <extension version>
```

3. Upgrade the data associated with the Microsoft extensions. This step will automatically install the new version on the tenant

```
Start-NAVAppDataUpgrade -ServerInstance <server instance name> -Tenant <tenant ID> -Name <extension
name> -Version <extension version>
```

Upgrade Third-Party extensions

If the old solution used third-party extensions, and you still want to use them, they must be compiled to work on the new platform. For more information, see Recompile published extensions. You then reinstall the extensions on tenants using the Install-NAVApp cmdlet.

As an alternative, if you have the source for these extensions, you can build and compile a new version of the extension in the AL development environment. Then, you upgrade to the new version as described in the previous task.

Post Upgrade

Upgrade control add-ins

The Business Central Server installation includes new versions of the Microsoft-provided Javascript-based control add-ins, like Microsoft.Dynamics.Nav.Client.BusinessChart, Microsoft.Dynamics.Nav.Client.VideoPlayer, and more. If your solution uses any of these control add-ins, upgrade them to the latest version.

To upgrade the control add-ins, do the following steps:

1. Open the Business Central client.

2. Search for and open the **Control Add-ins** page.
3. Choose **Actions** > **Control Add-in Resource** > **Import**.
4. Locate and select the .zip file for the control add-in and choose **Open**.

   The .zip files are located in the **Add-ins** folder of the Business Central Server installation. There's a subfolder for each add-in. For example, the path to the Business Chart control add-in is `C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins\BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip`.
5. After you've imported all the new control add-in versions, restart Business Central Server instance.

Alternatively, you can use the Set-NAVAddin cmdlet of the Business Central Administration Shell. For example, the following commands update the control add-ins installed by default. Modify the commands to suit:

```
$InstanceName = 'BC170'
$ServicesAddinsFolder = 'C:\Program Files\Microsoft Dynamics 365 Business Central\170\Service\Add-ins'
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.BusinessChart' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'BusinessChart\Microsoft.Dynamics.Nav.Client.BusinessChart.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.FlowIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'FlowIntegration\Microsoft.Dynamics.Nav.Client.FlowIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.OAuthIntegration' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'OAuthIntegration\Microsoft.Dynamics.Nav.Client.OAuthIntegration.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PageReady' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PageReady\Microsoft.Dynamics.Nav.Client.PageReady.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.PowerBIManagement' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'PowerBIManagement\Microsoft.Dynamics.Nav.Client.PowerBIManagement.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.RoleCenterSelector' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'RoleCenterSelector\Microsoft.Dynamics.Nav.Client.RoleCenterSelector.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SatisfactionSurvey' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SatisfactionSurvey\Microsoft.Dynamics.Nav.Client.SatisfactionSurvey.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.SocialListening' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'SocialListening\Microsoft.Dynamics.Nav.Client.SocialListening.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.VideoPlayer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'VideoPlayer\Microsoft.Dynamics.Nav.Client.VideoPlayer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WebPageViewer' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WebPageViewer\Microsoft.Dynamics.Nav.Client.WebPageViewer.zip')
Set-NAVAddIn -ServerInstance $InstanceName -AddinName 'Microsoft.Dynamics.Nav.Client.WelcomeWizard' -
PublicKeyToken 31bf3856ad364e35 -ResourceFile ($AppName = Join-Path $ServicesAddinsFolder
'WelcomeWizard\Microsoft.Dynamics.Nav.Client.WelcomeWizard.zip')
```

Update application version

This task isn't required for installing the update. However, it might be useful for support purposes and answering a common question about the application version. The **Help and Support** page in the client displays an application version, such as 16.1.2345.6. This version number isn't updated automatically when you install an update.

However, the Business Central Server includes a configuration setting called **Solution Version Extension** (SolutionVersionExtension). This setting lets you specify an extension whose version number will show as the Application Version on the client's **Help and Support** page. Typically, you'd use the extension considered to be

your solution's base application. You set **Solution Version Extension** to ID of the extension. For example, if your solution uses the Microsoft Base Application, set the value to `437dbf0e-84ff-417a-965d-ed2bb9650972`.

You can set **Solution Version Extension** by using the Business Central Server Administration tool or the Set-NAVServerConfiguration cmdlet of the Business Central Administration Shell.

The following example uses the Set-NAVServerConfiguration cmdlet to set the **Solution Version Extension** to the Microsoft Base Application:

```
Set-NAVServerConfiguration -ServerInstance <server instance name> -KeyName SolutionVersionExtension -
KeyValue 437dbf0e-84ff-417a-965d-ed2bb9650972 -ApplyTo All
```

For more information about how to configure a server instance, see Configuring Business Central Server.

See Also

Dynamics 365 Business Central On-Premises Release Wave 2 Updates
Upgrading to Dynamics 365 Business Central 2019 Release Wave 2
Synchronizing the Tenant Database and Application Database
Version numbers in Business Central
Publish and Install an Extension
Getting Started in AL
Version numbers in Business Central

Some Known Issues in Dynamics 365 Business Central On-premises

2/17/2021 • 6 minutes to read • Edit Online

This article describes some known issues in Business Central versions. These issues can impact installation, upgrade, and various operations of Business Central on-premises.

> **NOTE**
> The article doesn't include a complete list of known issues. Instead, it addresses some common issues that you might experience or might consider when upgrading to a version. If you're aware of issues that aren't in this article, or you'd like more help, see Resources for Help and Support.

Primary key mismatch between converted tables in local 14 versions and Microsoft Base Application

> Applies to: 14.18-14.x, 15.12-15.x, 16.7-16.x, 17.1-17.x

Problem

In some local versions, new table objects were added as part of a minor update. In version 14, which is still C/AL based, the primary keys of these tables are unnamed. When you convert the tables from CAL to AL using the txt2AL tool, primary keys are given the name `Key1`. In version 15, 16, and 17, which is fully AL-based, these tables have been moved to Microsoft base application extension, and the primary keys have the name `PK`. This difference can cause problems when you upgrade to use the Microsoft Base Application because it introduces breaking changes.

The affected local versions and tables include:

LOCAL 14 VERSION	TABLES
CH	• QRBill Buffer • QRBill Billing Detail • QRBill Billing Info • QRBill Layout QRBill • QRBill Setup • VAT Reg. No. Srv. Template • VAT Registration Log Details
NA	• VAT Reg. No. Srv. Template • VAT Registration Log Details

> **NOTE**
> You don't experience this problem with table objects in other Microsoft extension because the primary keys are named `KEY1`.

Impact

If you don't resolve this conflict, you'll experience problems when upgrading a customized version 14 (C/AL) application to the Microsoft Base Application.

For example, consider the following articles:

- Upgrading Customized C/AL Application to Microsoft Base Application Version 16
- Upgrading Customized C/AL Application to Microsoft Base Application Version 17

During this upgrade process, you'll create two versions of an extension referred to as the *table migration* extension. This extension is used for transferring ownership of existing tables in the database to other extensions. The first version contains table objects. The second version contains a migration.json file instead of the table objects. When you try synchronize the second version of the table migration extension (Task 13.5 or Task 12.6), you'll get the following errors for each affected table:

```
Table <table name> :: The previous primary key 'Key1' cannot be located. Changing the primary key is not
allowed.
```

```
Table <table name> :: Introducing a new key 'PK' as the primary key is not allowed. Please make the key
'Key1' the primary key again.
```

Workaround

To avoid these errors, after converting the version 14 tables to AL, rename instances of the primary key `Key1` to `PK` in the affected table objects before building your extensions.

- If you're following the upgrade articles listed above, do this change as part of Task 2. If you've already published the first table migration extension with the wrong key names, we recommend restore the databases and go through the upgrade again, starting from Task 2.
- You should also make these changes if you're only doing a technical upgrade, so you won't run into problems later if eventually uptake the Microsoft Base Application.

For each affected table object, do the following steps:

1. Open the .al file.
2. Locate the primary key definition. For example:

```
keys
{
    key(Key1; "Primary key")
    {
        Clustered = true;
    }
}
```

3. Change the name from `Key1` to `PK`. For example:

```
keys
{
    key(PK; "Primary key")
    {
        Clustered = true;
    }
}
```

Enum type fields not transferred as part of schema migration between extensions

> Applies to: 15.3-15.11, 16.0-16.4

Problem

The table migration feature fails to transfer data from a table object to a table extension object that introduces an enumeration.

> **NOTE**
> The table migration feature was introduced in version 15.3. For more information, see Migrating Tables and Fields Between Extensions.

Impact

You'll experience this issue in the following scenarios:

- Upgrading a customized version 14 solution to version 15 or 16
- Migrating data from one extension to another.

The issue occurs when synchronizing extensions. You'll get an error similar to one of the following messages:

```
Sync-NAVApp : The metadata object Enum <ID> was not found.
```

```
Message App Sync Exception -- Microsoft.Dynamics.Nav.Types.NavMetadataNotFoundException: The metadata object
Enum <ID> was not found.
```

Workaround

- If you're upgrading from version 14, use 15.12 (or later) or 16.5 (or later) as the target version.
- If you're migrating data between extensions, upgrade your solution to 15.12 (or later) or 16.5 (or later).

$systemModifiedAt fields in tables not populated during upgrade

> Applies to: 14.0-14.x

Problem

Business Central version 17 introduced data audit fields on tables. When you upgrade from version 15 or 16, the $systemModifiedAt field is populated with values from field **5120 ModifiedOn** in table **5151 Integration Record**. However, this mapping doesn't work in a version 14-to-17 upgrade.

Impact

When upgrading from version 14 to version 17, the $systemModifiedAt field will be blank.

Workaround

Upgrade to version 15 or 16 before upgrading to version 17.

Wrong .NET assemblies for external connected services

> Applies to: 16.5

Problem

Version 16.5 includes the wrong .NET 4.5 assemblies for connecting to external services that use OAuth, like Common Data Service.

Impact

In the Business Central client, you can set up external service connections from the **Service Connections** page.

When you try to set up a connection, it will fail because Business Central can't obtain an access token for connecting to the service.

In the client, you'll get an error similar to the following message:

```
Failed to acquire authorization token for <service>.
```

In the event log, you'll see an error similar to the following message:

```
Could not load file or assembly 'Microsoft.Identity.Client, Version=4.14.0, Culture=neutral,
PublicToken=0a613f4dd989e8e' or one of its dependencies. The located assembly's manifest location does not
match the assembly reference.
```

Workaround

There are two workarounds for this issue:

1. Upgrade to 16.6 or later.
2. Install the required assemblies:
   a. Go to https://www.nuget.org/packages/Microsoft.Identity.Client/4.14.0.
   b. Under **Info**, select **Download package**.
   c. Extract the contents of the nupkg file to a folder.

      There are different ways to extract the content of nupkg file. An easy way is to change the file extension from *.nupkg* to *.zip*. Then, extract the files like you would do with any .zip file.

   d. Copy the files in the **\lib\net45** folder to the **Service** folder of the Business Central Server installation. Replace the existing files when prompted.

      The default folder is C:\Program Files\Microsoft Dynamics 365 Business Central\160\Service.

   e. Restart the Business Central Server instance.

Help Server for India preview missing files

> Applies to: 17.x

Problem

The installation media (DVD) is missing required HTML files for the Help Server.

Impact

The Help Server installation will fail.

Workaround

Install Help Server without the HTML files for local functionality, then pick up the content from GitHub instead. For more information, see Get updates from Microsoft.

Business Central Server Administration tool doesn't work for multitenant server instances

> Applies to: 17.0-17.1

If a Business Central Server instance is configured as a multitenant instance, you can't manage the tenants by using the Business Central Server Administration tool.

Impact

When you try to use the **Tenant** node in the Business Central Server Administration tool, you get a message: The MMC has detected an error in the snap-in and will unload it. This message is followed by:

```
FX:{ba484c42-ed9a-0170-925f-23e64e686fd0}
```

```
Value cannot be null.
```

```
Parameter name: objectToValidate at Microsoft.ManagementConsole.BaseCollection.OnValidate(Object
objectToValidate, Boolean testForDuplicate) at
System.Collections.CollectionBase.System.Collections.IList.Add(Object value) at CallSite.Target(Closure ,
CallSite , NodeSubItemDisplayNameCollection , Object ) at
System.Dynamic.UpdateDelegates.UpdateAndExecuteVoid2[T0,T1](CallSite site, T0 arg0, T1 arg1) at
Microsoft.Dynamics.Nav.ManagementUI.NavTenantsListView.RefreshWithNodeStateChangedCheck(IList1 tenants,
ResultNodeCollection currentResultNodes) at
Microsoft.Dynamics.Nav.ManagementUI.NavTenantsListView.RefreshCore() at
Microsoft.Dynamics.Nav.ManagementUI.NavTenantsListView.OnInitialize(AsyncStatus status) at
Microsoft.ManagementConsole.View.ProcessRequest(Request request) at
Microsoft.ManagementConsole.ViewMessageClient.ProcessRequest(Request request) at
Microsoft.ManagementConsole.Internal.IMessageClient.ProcessRequest(Request request) at
Microsoft.ManagementConsole.Executive.RequestStatus.BeginRequest(IMessageClient messageClient, RequestInfo
requestInfo) at Microsoft.ManagementConsole.Executive.SnapInRequestOperation.ProcessRequest() at
Microsoft.ManagementConsole.Executive.Operation.OnThreadTransfer(SimpleOperationCallback callback)
```

Workaround

Use the Business Central Administration Shell. This error is fixed in later updates.

See Also

Upgrading to Business Central

Migrate Legacy Help to the Dynamics 365 Business Central Format

2/17/2021 • 5 minutes to read • Edit Online

Business Central implements a user assistance model where tooltips explain all fields and actions, and articles with conceptual descriptions of functionality are published to a website. If you are building an app, you are expected to comply with this model. However, there are many ways in which you can migrate and reuse your existing Help within this model.

Reusing existing web content

When you move to Business Central, you can reuse your existing product Help solution in most situations, especially if the content is already published to an *online library*, which is an internal or external website. In that case, all you have to do is to add that website to the configuration of Business Central online or on-premises. For more information, see Configuring the Help Experience.

More specifically, if you have content that you created for Dynamics NAV, then you can choose to reuse that content for your Business Central solution.

For example, you have a Dynamics NAV Help Server website with HTML files that describe your solution according to the Microsoft Dynamics NAV 2013 R2 documentation model and format. You can reuse the Help Server website and rebrand the website and the content. You then connect your Business Central solution with the Help Server website. For more information, see Configuring the Help Experience.

Business Central does not support the field-based approach to context-sensitive Help that Dynamics NAV 2017 and earlier versions use. Instead, you must use the Business Central page-based approach to context-sensitive Help. You do not have to convert your existing Help, but you do need to make the content available. For more information, see the Converting existing content section.

> **NOTE**
> For apps for Business Central online, you must apply tooltips to controls and actions in both page objects and page extensions, and you must supply context-sensitive links. For more information, see Configuring the Help Experience.

Converting existing content

If your existing content is in a different format, such as PDF files, Word documents, or printed manuals, you must decide if you want to keep the content as-is, or if you want to convert it to a format that can be accessed from inside Business Central. There are third-party tools available that can help you migrate your content, depending on the current format and the target format.

For example, if you are migrating your solution from Dynamics GP, you might have content in PDF files. You can choose to convert the content to MarkDown as described in the Moving to MarkDown section, and then publish to a new online library on your current website.

Migrating from Dynamics NAV

If you are migrating your solution from Microsoft Dynamics NAV 2013 R2 or later versions of Dynamics NAV, then you most likely have been using the Dynamics NAV Help Server, and your Help content is in HTML format. That means that you can reuse your existing content as-is, you can use the Custom Help Toolkit to get new HTML files to supplement your existing content, or you can use publicly available third-party solutions to

convert some or all of your HTML files to MarkDown if you want to follow similar processes to the ones the Microsoft team follows. For more information, see the Moving to MarkDown section.

If you are migrating from an earlier version of Dynamics NAV, then you can choose to first migrate to the Microsoft Dynamics NAV 2013 R2 format, and then migrate again to MarkDown. For more information, see Upgrading Your Existing Help Content in the legacy docs for Microsoft Dynamics NAV 2013 R2.

Converting legacy Dynamics NAV field Help to tooltips

Tooltips play an important role as part of the Business Central user assistance model, and we encourage you to apply tooltips to your controls and actions as well.

If you would like to reuse text from your Dynamics NAV Help for table fields, the FieldTopicTextExtractor tool can extract the first paragraph from your Dynamics NAV Help topics for table fields. The resulting spreadsheet will help you copy and paste the text from your Dynamics NAV Help topics into the ToolTip property of your controls.

Mapping the table fields to page controls is not always straightforward because the same table is often used by two or more pages. As a result, the page ID can be many numbers away from the table ID. It requires knowledge of the actual solution to determine which tooltips are relevant for which pages. The **FieldTopicTextExtractor** tool generates an Excel file that you can use for this purpose, since you can sort and filter, and then copy content in Excel.

In the base application, tooltips are in the page objects, so, at Microsoft, we edit code to edit tooltips. You do not have to do the same. You can choose to work with tooltips in the translation files or straight in the .AL files. Different solutions require different processes, so pick the process that is more efficient for you. We chose to associate the "What is this field?"-content with the user interface, meaning the controls on page objects. In the Dynamics NAV Help model, we chose a different approach, focusing on the database structure. Both approaches have their advantages and disadvantages, but the Business Central user assistance model currently focuses on the user interface with tooltips on page objects. For more information, see User Assistance Model.

For more information, see Working with Application Objects as Text Files in the docs for Microsoft Dynamics NAV 2016, How to Add Translated Strings By Importing and Exporting Multilanguage Files in the docs for Microsoft Dynamics NAV 2018, and Working with Translation Files for Business Central.

Moving to MarkDown

Converting your existing content to MarkDown can be done using third-party tools, including but not limited to PanDoc or the Writage plugin for Word.

Once you have converted your content to MarkDown, we recommend storing your content in a repository, such as a Git repo in Azure DevOps, a private or public repo in GitHub, or a project in MkDocs. You can use open-source tools such as DocFx to generate content for your website. In general, working in MarkDown means that you have access to a world of open-source tools and do not have a hard dependency on Microsoft providing you with tools. The Custom Help Toolkit can help you prepare content for publishing. For more information, see Custom Help Toolkit.

If you do not yet have a website that you publish content to, then there are several ways in which you can create such a site. The MkDocs project generates a website for you, but you can also work with a web designer to build a site to host your content. We recommend deploying to an Azure web app.

See Also

Configuring the Help Experience
Extend, Customize, and Collaborate on the Help
User Assistance Model
Development of a Localization Solution