Develop and customize home page

Article • 12/05/2024

This article provides links to topics about development.

Overview

The finance and operations apps enable the entire enterprise resource planning (ERP) application suite as a cloud-based solution, for both public and private clouds, as well as on-premises. The apps leverage the speed, simplicity, and cost-effectiveness of working in the cloud, while building on the latest technology from Microsoft. The development experience includes:

- Development tools that are decoupled from any running environment. You develop against local, XML-based files, not the online database.
- Microsoft Visual Studio is the development environment. The Visual Studio environment is customized to provide you with a smooth and familiar experience.
- The X++ compiler generates Common Intermediate Language (CIL) for all features. CIL is the same intermediate language used by other .NET-based (managed) languages, such as the C# programming language.
- You can leverage the browser-based client and the design patterns for forms to provide an improved end-user experience.
- The Application Lifecycle Model (ALM) supports build automation, test automation, and deployment of models to the cloud.

Architecture

- Application stack and server architecture

Getting started

- Get evaluation copies
- Sign up for preview subscriptions
- Deploy and access development environments
- Development system requirements
- Removed or deprecated features
- Deprecated APIs
- Rename a local development (VHD) environment
- Introduction to Azure DevOps (Video)

Application stack and server architecture

Article • 08/12/2022

The application stack is divided into platform models and application-specific models. The platform models are Application Platform, Application Foundation, and Test Essentials. There are many application-specific models. Some examples are Application Suite, Ledger, Retail, and Case Management.

Overview

The application stack and server architecture align with three key pillars:

- New client
- Cloud readiness
- New development stack

The application stack is divided into several models: Application Platform, Application Foundation, Test Essentials, and the application suites. The separation enables new application development on the base foundation models, just as the Fleet Management sample application has been developed. Note the following important points about the changes in the server architecture:

- The services endpoint on the server is now responsible for returning all form and control metadata and data to the browser-based client. There is no longer any remote procedure call (RPC)-based communication with the server. The form objects still run on the server, and rendering has been optimized for browsers and other clients through server and client-side (browser) investments.

- The server, including the application code base, is deployed to an Internet Information Services (IIS) web application. In the cloud, it's deployed to Microsoft Azure infrastructure as a service (IaaS) virtual machines (VMs).

- It is hosted on Azure and is available for access through the Internet. A user can use a combination of clients and credentials to access it. The recommended primary identity provider is OrgID, and the store for the identity is Microsoft Entra ID. The security subsystem uses the same AuthZ semantics for users and roles.

- Two types of clients must be considered for access in the cloud: active clients and passive clients.

Get evaluation copies

Article • 08/12/2022

A public preview is available. You can sign up and deploy a cloud instance of the latest build. This public preview available through Microsoft Dynamics Lifecycle Services (LCS). These links provide more information about how to download and use the public preview:

- Sign up for preview subscriptions
- Service update availability
- Partner Trial
- How can I setup a solution trial instance in Azure with my customization and demo data?
- How do I login to the new AX as a demo user persona?

Feedback

Was this page helpful? Yes No

Provide product feedback

Sign up for preview subscriptions

Article • 08/20/2024

This article explains how to subscribe to the preview/partner offer and deploy an environment. The subscription that you create gives you a Microsoft Online Services test tenant and a Microsoft Dynamics 365 Lifecycle Services project where you can deploy an environment. This article will also help you set up additional users in your Microsoft Online Services tenant and gain experience with service administration capabilities. Here are the skills that you will learn:

- Subscribe, and create a new Microsoft Online test tenant.
- Navigate to Lifecycle Services projects.
- Use various features of Lifecycle Services.
- Add users to Microsoft Microsoft Entra and the client.
- View resources in your subscription email.

Key terms

- **Microsoft Online Services tenant –** A tenant is the group of all subscriptions and users for your organization. The tenant is created at the same time as your first subscription in Microsoft Online Services.

- **Subscription –** A subscription gives you an online cloud environment and experience. It also lets you see how customizations that you develop can be deployed to the cloud.

- **Microsoft Entra ID –** The cloud environment includes Microsoft Entra ID. Microsoft Entra helps you manage users, groups, security roles, and licenses for online applications, much as you manage them for on-premise environments.

- **Users –** Users of the services that your organization has subscribed to are managed in Microsoft Entra ID. Any users in your tenant can be added and assigned to security roles.

- **Developers and administrators –** Developers and administrators are users who also have access to Lifecycle Services that lets them manage projects and environments. These users are also end users.

- **Organizational account –** Users receive Microsoft Entra credentials. These credentials are separate from other desktop or corporate credentials. The

Deploy and access development environments

Article • 03/08/2024

This article describes how to access development instances, configure local development virtual machines (VMs), and find important configurations settings for developers and administrators.

> **Note**
>
> - Microsoft Support may provide limited troubleshooting on Tier 1 development environments.
> - In certain circumstances, a fresh deploy of a Tier 1 environment may be requested by Microsoft Support to resolve an issue.
> - Development environments shouldn't contain business critical data and are considered disposable.
> - Only 120 environments are support per tenant. We recommend that you limit the number of cloud-hosted environments under a specific tenant to allow enough capacity to be able to deploy sandbox and production environments.
> - For cloud-hosted environments older than 6 months, it's advisable to review the supported software list on the Microsoft Dynamics 365 Finance + Operations supported software page.

Definitions

Expand table

Term	Definition
End user	A user who accesses an instance through the web client. The end user must have Microsoft Microsoft Entra credentials to access an instance and must be provisioned/added as a user of that instance.
Developer	A user who develops code through the Microsoft Visual Studio environment. A developer requires Remote Desktop access to development environment (VM). The developer account must be an administrator on the VM.

Configure one-box development environments

Article • 08/12/2022

This article describes recommended configurations of your one-box developer environment.

Setup

1. Start Visual Studio, and on the toolbar, click **Dynamics 365 **> **Options**.
2. Expand the **Microsoft Dynamics** node, and then click **Projects**.
3. Verify that the **Organize projects by element type** check box is selected, and click ***OK***.
4. To view the line numbers in your code editor, select **Tools** > **Options** > **Text Editor** > **All Languages**.
5. Select the **Line numbers** check box.

Debugging

For better performance of the X++ debugger, you might want to turn off IntelliTrace. IntelliTrace collects the complete execution history of an application. It is not supported for X++ debugging and causes performance issues in the IDE when debugging large packages like Application Suite. To turn off Intellitrace, click **Options** > **IntelliTrace** > **Enable IntelliTrace**, clear the check box, and then click **OK**. Note that Intellitrace is only available in the Enterprise version of Visual Studio.

Feedback

Secure one-box development environments

Article • 07/01/2024

This article describes how to help secure one-box developer environments.

One-box developer environments consist of Microsoft Azure resources that are deployed in your own Azure subscription.

When you deploy a one-box developer environment, you can expect the following Azure resources to be deployed:

- 1 virtual machine (VM)
- 5 disks
- 1 load balancer
- 1 regular network interface
- 1 network security group
- 1 virtual network
- 1 public IP address
- 1 storage account
- 1 or more other storage accounts that are prefixed with "dyn," for the storage of product binaries

Because these Azure resources are under your own management, you might have to comply with security and compliance requirements that are specific to your organization. In addition, recommendations from sources such as Microsoft Defender or Azure Well-Architected security assessments might apply to your one-box developer environment. This article outlines some of these recommendations to help you make decisions that best secure your one-box environments.

Default configuration

Out of the box, your one-box developer environment has the following basic security configuration:

- Management ports on your VM are restricted to Microsoft Dynamics Lifecycle Services IP addresses. You can view these restrictions in the network security group in the resource group for your environment. Lifecycle Services uses the Windows Remote Management (WinRM) management port (5986) to do the initial

Development and build VMs that don't allow admin access FAQ

Article • 07/01/2022

This article provides answers to frequently asked questions (FAQ) about virtual machines (VMs) that don't allow administrator access.

How can I install a deployable package?

Whenever possible, use Microsoft Dynamics Lifecyle Services (LCS) to install a deployable package. You can install a deployable package by using the **-devinstall** option. Remember that this option requires manual database synchronization.

For more information about how to install a deployable package, see Install deployable packages from the command line.

Is the finance and operations website accessible when Visual Studio isn't running?

Yes, you can access the finance and operations website when Microsoft Visual Studio isn't running. Microsoft Internet Information Services (IIS) Express is an .exe file that runs as the user. However, when you close Visual Studio, the XPPC agent starts regular IIS (not IIS Express) before it closes. This behavior helps to ensure that you can remotely access the Application Object Server (AOS) instance and the website, even when you sign out or the machine is restarted. We recognize that many people use these developer machines as test machines, and that they expect the AOS instance always to be running. However, IIS Express doesn't support this behavior.

What about the other services?

You can restart Microsoft Windows services such as Microsoft SQL Server, SQL Server Reporting Services (SSRS), SQL Server Integration Services (SSIS), SQL Server Analysis Services (SSAS), Batch, Financial reporting (formerly Management Reporter), and IIS. (For IIS, you must restart the World Wide Web Publishing Service because you can't use iisreset.exe.)

Can I clean up the service volume drive?

Set up the downloadable VHD for first use

Article • 03/08/2024

> **Note**
>
> This article applies to the virtual hard drive (VHD) that was released for versions 10.0.24 and later.

When you first sign in to the virtual machine, the **Application Object Server** will not be ready for use. A script needs to be run that will create self-signed certificates to be used on the virtual machine, and a customer-provided application registration ID for authentication. After successfully running the script, the environment will be ready for use.

Register a new application in Microsoft Entra ID

To register a new application in Microsoft Microsoft Entra ID, follow the steps outlined in Register app or web API. The new app registration should be for a web application, and the following redirect URIs should be added:

- `https://usnconeboxax1aos.cloud.onebox.dynamics.com/`
- `https://usnconeboxax1aos.cloud.onebox.dynamics.com/oauth/`

Once created, make note of the **Application (client) ID**.

Run the setup script

After you sign in with the **localadmin** account, right-click the desktop shortcut **Generate Self-Signed Certificates**, and select **Run as administrator**. When the script prompts for the application ID, provide the **Application (client) ID** created in Microsoft Entra ID.

When the script finishes, the environment is ready for use. At this time, you can run the Admin Provisioning tool to set the administrator account, permissions, and tenant. Make sure that the email provided is for the Microsoft Entra tenant in which the application registration was created.

Rename a local development (VHD) environment

Article • 08/12/2022

A local development (VHD) environment must be renamed for the following scenarios:

- **Accessing a single Microsoft Azure DevOps project across multiple machines:** Azure DevOps is required for version control. In development topologies, multiple virtual machines (VMs) can't access the same Azure DevOps project if they have the same machine name. Azure DevOps uses the machine name for identification. If you're developing on local VMs that were downloaded from Microsoft Dynamics Lifecycle Services (LCS), you might encounter issues.
- **Installing One Version service updates:** One Version service updates, such as 8.1.x, must be installed in VHD environments by using a runbook. To help guarantee that the runbook is completed successfully, the VHD environments must be renamed. Additional steps that are described in this article must also be completed.

Rename the machine

Rename and restart the machine before you start development or connect to Azure DevOps. Make sure that the new name is unique among all the machines that are used with the Azure DevOps project.

Update the server name in SQL Server

Update the server name in Microsoft SQL Server 2016 by running the following commands.

SQL

```
sp_dropserver [old_name];
GO
sp_addserver [new_name], local;
GO
```

In these commands, be sure to replace **old_name** with the old name of the server and **new_name** with the new name. By default, the old name is **MININT-F36S5EH**, but you can run **select @@servername** to get the old name. Additionally, be sure to restart the SQL Server service after the commands have finished running.

Development system requirements

Article • 08/12/2022

This article lists the system requirements for development.

Development environments can be hosted locally or in Microsoft Azure. The build process, X++ compilation, and generation of cross reference information, typically run satisfactorily on machines with 16 GB of memory and two CPU cores. The compiler uses available resources, so more RAM and more cores can speed up compilation, especially if there is contention for the resources from other concurrent processes. If you are running concurrent processes, then we recommend 24 GB of memory with four cores. At a minimum, two CPU cores are recommended because the developer environment contains many components that may be running concurrently. The components include the AOS web application, Visual Studio, Management Reporter, and SQL Server.

Feedback

Migrate a CHE load balancer

Article • 03/26/2025

Microsoft Azure announced the retirement of classic load balancers. To ensure that CHE environments can continue to be provisioned, Microsoft implemented provisioning for new CHE environments and migration for existing CHE environments.

Provision new CHE environments

As of March 28, 2025, all new CHE environments are configured with a load balancer and a public IP address that uses the standard stock-keeping unit (SKU). This transition helps enhance the performance and reliability of our cloud infrastructure.

Migrate existing CHE environments

Before you migrate, it's crucial to ensure that you have the necessary permissions and contributor access to the subscription. This access allows you to upgrade the load balancer without encountering any permission-related issues. After you verify your access, you can set the required variables and run the migration commands.

To migrate existing CHE environments, follow these steps.

1. Set the variables for the subscription ID, resource group name, and load balancer name.

   PowerShell

   ```
   $resourceGroupName = "<resourcegroup-name>"
   $loadBalancerName = "<loadbalancer-name>"
   $subscriptionId = "<subscription-id>"
   $outboundRuleName= "http-outbound-rule"
   $backendPoolName = "vm-backend-pool"
   ```

2. Run the following command to ensure that you specified the correct subscription ID that is associated with the basic load balancer.

   PowerShell

   ```
   Select-AzSubscription -Subscription $subscriptionId
   ```

Version control, metadata search, and navigation

Article • 03/31/2023

This tutorial will walk you through configuring Microsoft Azure DevOps to enable source control on your models. It will also help you learn about other productivity features in the development tools, including the ability to create and organize TODO task, search metadata and source code, navigate between related model elements, and create a project from a model.

Configure your Azure DevOps organization and project

In this section, you'll create a new project in Azure DevOps. This project will host the source code of your model. You'll use the Fleet Management model as an example. If you don't have an Azure DevOps organization, you'll create one.

Sign up to Azure DevOps, create an account, and create a new project

Go to https://www.visualstudio.com/ to sign up for Azure DevOps. Click **Sign up**. If you already have an account in Azure DevOps, go to the Create an Azure DevOps team project section later in this article.

1. Sign in with your Microsoft account.

   > **Note**
   >
   > You can also use an organizational account (Microsoft 365 domain).

2. Create an Azure DevOps organization, and select a URL for your account. You'll use this URL to connect from your development computer when you configure source control in Visual Studio. The following image shows an example of the account URL.

Branching overview

Article • 04/07/2023

Branching configurations for X++ repositories (repos) vary, depending on the development team's preference and the finance and operations app lifecycle. If the implementation already has a preferred branching structure, and it meets the minimum branching criteria that are outlined later in this article, you can use it to manage new development. Alternatively, you can evaluate the other branching structure options that are explained in this article.

Considerations

You'll find that a few elements of the X++ development cycle differ from general application development. Keep the following items in mind as you consider how to structure your branches:

- Enterprise resource planning (ERP) systems are business-critical environments. When you design your code management infrastructure, you should prioritize design elements that help minimize both the risk of major production issues and the disaster recovery timeline.
- Because of the complexity of, and interdependency on, the standard system code, it's generally a good idea to do both automated testing and manual testing of **all critical business processes** after each code update. Because this testing can be lengthy, you should plan to have a branch to contain the in-test code for long spans of time while it's being validated.
- Because dev teams frequently focus their collective attention on enhancing specific modules and areas of the product, merge conflicts are relatively frequent for X++. Therefore, you should plan a branching strategy that helps reduce the frequency of collisions and makes collisions easier to fix when they occur.

Minimum branching criteria

At a minimum, any X++ repo branching strategy should support the following criteria:

- **Isolation of untested development code or code that's in development –** Developers should be protected from having teammates accidentally break any active development branch. If the team shares a single development branch, each developer must ensure that they don't commit code that breaks teammates' code. Additional configuration of the version control capabilities should be used for this

X++ in Git

Article • 06/19/2024

Git is the modern standard for version control. It's also Microsoft's default version control system provider that's recommended for general development. X++ developers might be already familiar with the existing guidance to set up Team Foundation Version Control (TFVC) for X++, but many organizations are standardizing on Git. This article introduces the concept of X++ code management in Git and outlines key considerations for teams that want to use Git tools for X++ development.

Considerations and limitations

The following list describes known limitations and workarounds that are required to configure a Git repository (repo) for X++ and set up developer environments for X++ development via Git:

- Dynamics 365 Finance and Operations Tools are build tools that are offered exclusively as a Microsoft Azure DevOps extension. Therefore, for full continuous integration and continuous delivery (CI/CD) support, consider using Azure DevOps for your Git repo.
- More configuration steps are required to successfully map a Git version control system to the PackagesLocalDirectory directory in a development environment, as is also the case with TFVC. The Microsoft Developer blog documents one way to do this configuration.
- Dynamics Lifecycle Services checks for a TFVC repo during the deployment of build environments. Therefore, if your team uses build environments, you'll have to initialize a TFVC repo and connect to it from Lifecycle Services, even if you aren't actively using it for code management.
- Branching strategies in TFVC often differ from branching strategies in Git. Common Git-based branching strategies include trunk-based management and GitFlow or GitHub Flow. Learn more about branching strategies for X++ development.

Resources

- Migrating from TFVC to Git
- Feature comparison between Git and TFVC

Feedback

Build automation that uses Microsoft-hosted agents and Azure Pipelines

Article • 02/27/2025

In Microsoft Azure DevOps, you can automate the process of building X++ code and creating deployable packages on any Windows build agent. These agents include Microsoft-hosted agents. This approach helps you avoid the setup, maintenance, and cost of deploying build virtual machines (VMs). It also lets you reuse the existing setup of build agents to run other .NET build automation.

> **Note**
>
> This feature is limited to compilation and packaging. There is no support for X++ unit testing (SysTest), database synchronization, or other features that require the runtime (Application Object Server [AOS]) or its components.

Prerequisites for building X++ code

Build projects

To use .NET tools to build X++ in Azure DevOps, the Microsoft Build Engine (MSBuild) and custom X++ targets are used. Your X++ source code repository must contain an X++ project for each package that you have to build.

You can optionally use a solution file to group the projects, including C# project dependencies, and provide an explicit build order. If the repository doesn't already contain a project, you can create a project in Visual Studio.

Ensure that a descriptor folder exists, and that the descriptor file for each model is checked into your model metadata folder in Azure DevOps.

> **Note**
>
> When you use an existing X++ project (rnrproj), make sure that you used Visual Studio tools to create it, or to open and save it.
>
> Although a package can contain multiple models, it must always be built in its entirety. Therefore, only one project for just one of the models is required to build

Add license files to a deployable package in Azure Pipelines

Article • 10/03/2023

The article explains how you can add license files to an existing software deployable package when you run build automation in Microsoft Azure DevOps.

When you update an environment by using a deployable package, a license might be required for independent software vendor (ISV) or partner X++ solutions. ISVs can create pipelines to automatically include licenses in release or build pipelines. Customers can create their own pipelines to combine the ISV deployable package and the license file.

This article assumes a working knowledge of Azure Pipelines.

> **Note**
>
> Before you can add these steps to a pipeline, the Dynamics 365 finance and operations Tools extension for Azure DevOps must be enabled and installed in the Azure DevOps account. For more information about how to install an extension for an organization, see Install extensions.

Adding the task to a pipeline

To add the task to your build for the YML or Classic pipeline, search the task list for **Add Licenses to Deployable Package**. The following table describes the options that are available for this task.

Expand table

Input name	Mandatory	Description
Search pattern for license files to add to the package	Yes	A list of license files on the build agent, or a search pattern for files on the build agent. To make the license files available on the build agent, you can add them to source control. Alternatively, they can be downloaded or generated in an earlier step of the pipeline. For more information, see File matching patterns reference.
Filename and path of the	Yes	The path and file name of an existing deployable package zip file that the license files should be added to.

Create deployable packages in Azure Pipelines

Article • 03/12/2025

If you want to deploy customizations to an environment, a deployable package is required in Microsoft Dynamics Lifecycle Services. You can create this package by using Azure Pipelines during a build or release process.

This article assumes a working knowledge of Azure Pipelines.

> **Note**
>
> - Before you can add these steps to a pipeline, the Dynamics 365 finance and operations Tools extension for Azure DevOps must be enabled and installed in the Azure DevOps organization. For more information about how to install an extension for an organization, see Install extensions.
> - This Azure DevOps task requires that the X++ compiler tools be available on the agent. Either run this task on a build virtual machine (VM) agent, or use the Compiler Tools NuGet package. For more information about the NuGet package and how to install it in a pipeline, see Build automation using Microsoft-hosted agents and Azure Pipelines.

> **Note**
>
> Versions 1.* and lower of the task are obsolete. Update the task version to the latest available.

Add the task to a pipeline

To add the task to the build of your YML or Classic pipeline, search the task list for **Create Deployable Package**. The following table describes the options that are available for this task.

Expand table

X++ model-versioning in Azure Pipelines

Article • 08/12/2022

During build automation, X++ model versions can be updated so that they match or are linked to the build number of the pipeline. These updates make it easier for customers to identify the version of the X++ packages that they are running. They also let developers track versions back to the build pipeline and the version of the source code files.

This article assumes a working knowledge of Azure Pipelines.

> **Note**
>
> Before you can add these steps to a pipeline, the Dynamics 365 finance and operations Tools extension for Azure DevOps must be enabled and installed in the Azure DevOps account. For more information about how to install an extension for an organization, see Install extensions.

Add the task to a pipeline

To add the task to the build of your YML or Classic pipeline, search the task list for **Update Model Version**. The following table describes the options that are available for this task.

Expand table

Input name	Mandatory	Description
X++ Source Location	Yes	The path of a parent folder that contains X++ source code. The **Descriptor Search Pattern** option will be run in this path and subfolders. The default value, **$(Build.SourcesDirectory)**, points to the root of the source code repository.
Descriptor Search Pattern	Yes	Provide a file matching pattern to find the descriptor files inside the path that is specified in the **X++ Source Location** option. You can also specify a list of full paths of descriptor files instead of search patterns. For more information, see File matching patterns reference.
Lowest Layer to	Yes	When using search patterns, this task may find descriptors for ISV or partner code in your source control. Those descriptor versions

Download assets by using Azure Pipelines

Article • 03/12/2025

You can automate the download of assets from the Asset library in Microsoft Dynamics Lifecycle Services by using the **Dynamics Lifecycle Services Asset Download** task in Azure DevOps.

This article assumes that you have a working knowledge of Azure Pipelines.

> **Note**
>
> Before you can add these steps to a pipeline, the Dynamics 365 finance and operations Tools extension for Azure DevOps must be enabled and installed in the Azure DevOps organization. For more information about how to install an extension for an organization, see Install extensions.

> **Note**
>
> Versions 1.* and lower of the task are obsolete. Update the task version to the latest available.

> **Note**
>
> (Not required for Version 2.* and later.) Only Version 1.* of the download task requires availability of the MSAL.PS PowerShell library. A task is available to automatically install the tools during pipeline execution. This task can be added anywhere in the stage before the download task. For more information, see Add the MSAL.PS install task to a pipeline.

Add the task to a pipeline

To add the task to the build of your YML or Classic pipeline, search the task list for **Dynamics Lifecycle Services Asset Download**.

The following table describes the options that are available for this task.

Upload assets by using Azure Pipelines

Article • 03/12/2025

You can automate the upload of assets to the Asset library in Microsoft Dynamics Lifecycle Services by using the **Dynamics Lifecycle Services Asset Upload** task in Azure DevOps. This task is available only in **Releases** pipelines.

This article assumes you have a working knowledge of Azure Pipelines.

> **Note**
>
> Before you can add these steps to a pipeline, the Dynamics 365 finance and operations Tools extension for Azure DevOps must be enabled and installed in the Azure DevOps organization. For more information about how to install an extension for an organization, see Install extensions.

> **Note**
>
> Versions 1.* and lower of the task are obsolete. Update the task version to the latest available.

> **Note**
>
> (Not required for Version 2.* and later.) Version 1.* of the upload task requires availability of the MSAL.PS PowerShell library. A task is available to automatically install the tools during pipeline execution. This task can be added anywhere in the stage before the upload task. For more information, see Add the MSAL.PS install task to a pipeline.

Add the task to a pipeline

To add the task to the build of your YML or Classic pipeline, search the task list for **Dynamics Lifecycle Services Asset Upload**.

The following table describes the options that are available for this task.

Expand table

Deploy assets by using Azure Pipelines

Article • 03/12/2025

You can use the **Dynamics Lifecycle Services Asset Deployment** task in Microsoft Azure DevOps to automate the deployment of assets that are stored in the Asset library in Microsoft Dynamics Lifecycle Services to specific environments. However, this task has the following limitations that you should consider:

- The task is available only in **Releases** pipelines.
- The deployment of software deployable packages to production environments can't be automated.
- Software deployable packages can't be deployed to build environments.
- Commerce Cloud Scale Unit (CSU) extension packages and e-commerce packages can't be deployed to local business data environments on-premises.

> **Note**
>
> Commerce Cloud Scale Unit (CSU) extension or e-commerce packages can be deployed to a production, UAT, or Sandbox environment using the **Dynamics Lifecycle Services Asset Deployment**.

This article assumes that you have a working knowledge of Azure Pipelines.

> **Note**
>
> Before you can add these steps to a pipeline, the Dynamics 365 finance and operations tools extension for Azure DevOps must be enabled and installed in the Azure DevOps organization. For more information about how to install an extension for an organization, see Install extensions.

> **Note**
>
> The Azure DevOps pipeline must be run by a user with the **Environment manager** role in the corresponding Dynamics Lifecycle Services project.

> **Note**

Create a Dynamics Lifecycle Services connection in Azure pipelines

Article • 03/08/2024

The Dynamics 365 finance and operations Tools extension for Microsoft Azure DevOps has several pipeline tasks that let you perform actions in Microsoft Dynamics Lifecycle Services. For example, you can upload assets, download assets, and service an environment. For the connection with Dynamics Lifecycle Services to work, you must set up a new service connection in Azure DevOps. This service connection provides the authentication details that are required to connect to Dynamics Lifecycle Services. For more information about service connections in Azure DevOps, see Service connections.

This article assumes that you have a working knowledge of Azure Pipelines.

> **Note**
>
> Before you can add these steps to a pipeline, the Dynamics 365 finance and operations Tools extension for Azure DevOps must be enabled and installed in the Azure DevOps account. For more information about how to install an extension for an organization, see Install extensions.

Prerequisites

You must have the credentials for a user who has access to one or more Dynamics Lifecycle Services projects that you want to interact with from Azure DevOps. Make sure that this user has successfully signed in to Dynamics Lifecycle Services before, and has opened the dashboard for the projects that you want to interact with.

> **Note**
>
> Dynamics Lifecycle Services doesn't support service-to-service authentication. Therefore, only regular user credentials (that is, a user name and password) can be used. Because the pipelines don't run interactively, multifactor authentication must not be set up for the account that you use. We recommend that you set up a separate user account that has limited access and strong credentials that can regularly be rotated for security purposes.

Update Lifecycle Services Connection authentication tasks to MSAL in Azure Pipelines

Article • 03/12/2025

By default, new versions of the Microsoft Azure DevOps tasks for Dynamics 365 support the Microsoft Authentication Library (MSAL) and use it by default for authentication.

Update existing service connections

For existing service connections, the **Authentication Endpoint** setting must be updated to `https://login.microsoftonline.com/organizations`. If you're using a national cloud, learn more in National clouds to find your relevant endpoint. The correct authentication endpoint is set by default when you create a new service connection. The setting must be updated only for national clouds.

Dynamics Lifecycle Services API Endpoint – Provide the endpoint. If your Lifecycle Services project is deployed in local geographies, ensure you're using the correct Lifecycle Services API endpoint address. Use the correct endpoint, learn more in Supported geographies and endpoints.

For more information about how to set up a connection, see Create an Lifecycle Services connection in Azure Pipelines.

Add the MSAL.PS install task to a pipeline

[!NOTE] This isn't required anymore with the latest versions of the tasks.

To add the MSAL.PS install task to the build of your YML or Classic pipeline, search the task list for **Install MSAL.PS to enable authentication**. There are no options or settings for this task. Make sure that this install task is run on every agent before you run any task that requires authentication with Microsoft Dynamics Lifecycle Services,

> **Note**
>
> The MSAL.PS libraries must be installed on every agent that runs tasks that require authentication with Lifecycle Services. If your pipeline consists of multiple stages,

Update the hosted Azure Pipeline for new NuGet packages

Article • 08/12/2022

> **Note**
>
> This article applies to pipelines that were set up for versions 10.0.17 or earlier. This does not apply to the legacy build pipeline that uses the build virtual machine.

Platform updates for version 10.0.18 introduce a new NuGet package. The new package is a result of a package split for the Application Build Reference code. As a result, you have to make changes to pipelines created for 10.0.17 or earlier versions.

Add the new package to packages.config list

The **packages.config** file used for your build already includes three packages:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
    <package id="Microsoft.Dynamics.AX.Platform.DevALM.BuildXpp"
version="7.0.5934.35741" targetFramework="net40" />
    <package id="Microsoft.Dynamics.AX.Application.DevALM.BuildXpp"
version="10.0.761.10019" targetFramework="net40" />
    <package id="Microsoft.Dynamics.AX.Platform.CompilerPackage"
version="7.0.5934.35741" targetFramework="net40" />
</packages>
```

You need to add a fourth package to the list, **Microsoft.Dynamics.AX.ApplicationSuite.DevALM.BuildXpp**. The resulting **packages.config** file should look like the following example, replacing the version number with the version number that your pipeline uses.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
    <package id="Microsoft.Dynamics.AX.Platform.DevALM.BuildXpp"
version="7.0.5968.16973" targetFramework="net40" />
    <package id="Microsoft.Dynamics.AX.Application.DevALM.BuildXpp"
version="10.0.793.16" targetFramework="net40" />
    <package id="Microsoft.Dynamics.AX.ApplicationSuite.DevALM.BuildXpp"
```

Update a legacy pipeline in Azure Pipelines

Article • 08/12/2022

> **Note**
>
> This documentation does not apply to the new build pipeline, even if you run it on the build virtual machine.

An **Azure Pipelines** pipeline explicitly specifies the versions of **Visual Studio** tools such as **MSBuild** and **VS Test**. To continue supporting newer versions of these products, new virtual machines will be deployed with newer versions of **Visual Studio** pre-installed. Any existing pipelines need to be manually updated to use the newer version.

Determine if your pipeline needs to be updated

Build and development virtual machines deployed with version 10.0.13 through version 10.0.20 include **Visual Studio 2017**. Those deployed with version 10.0.21 and later include **Visual Studio 2019**. Support for Visual Studio 2015 was deprecated in April 2021 and support for Visual Studio 2017 was deprecated in April 2022. If your build virtual machine does not include **Visual Studio 2019** you must plan to deploy a new build virtual machine with version 10.0.21 or later, or consider using the new hosted build pipeline.

If your build virtual machine has Visual Studio 2019 installed, you can use the newer versions of **MSBuild** and **VS Test**. If your pipeline was created by a virtual machine deployment prior to version 10.0.21, you will have to manually update the pipeline.

Finally, you can check your build pipeline on the **Build the solution** step. The **MSBuild Version** property indicates the version of **Visual Studio** in use.

Expand table

MSBuild version	Visual Studio version
MSBuild 14.0	Visual Studio 2015
MSBuild 15.0	Visual Studio 2017
MSBuild 16.0	Visual Studio 2019

End-to-end scenario for the Fleet Management sample application

Article • 08/12/2022

This tutorial walks you through an end-to-end scenario that the Fleet Management sample application is designed to support.

In this tutorial, you'll take a tour of the Fleet Management sample. The overviews in this tutorial provide some background knowledge and contextual info. You'll walk through an end-to-end scenario that this sample application is designed to support. This is information that you should have before proceeding to other tutorials.

Prerequisite

- You must first be provisioned as an end user before you start this tutorial.
- This tutorial mainly explores the FleetManagement Migrated project and the application that it builds.

Installing the demo data

To work with the sample, you must install the provided demo data.

1. In the virtual machine (VM), open Microsoft Edge and navigate to the application's base URL.

2. Sign in.

3. On the dashboard, open the navigation pane and go to **Fleet Management** > **Setup**> **Fleet setup**.

Fleet Management sample application

Article • 06/12/2023

This article is an overview of the Fleet Management sample application.

The Fleet Management sample application has been provided to showcase development and foundation capabilities. Fleet Management represents a solution that an ISV might create for a car-rental agency. Fleet Management data includes vehicles which are available for renting, and customers who can rent and return these vehicles. Employees can also run a maintenance workflow on these vehicles.

For some tutorials, you will need to create the FleetManagement solution if it is not on your computer. The steps to create it are listed in End-to-end scenario for the Fleet Management sample application.

For some tutorials, you must download the Fleet Management tutorial code and other artifacts from https://github.com/Microsoft/FMLab .

Fleet Management is provided as a Visual Studio solution that demonstrates platform capabilities, such as:

- Forms
- Workflow
- Security
- Labels
- Resources
- Data
- Business Intelligence
- Extensions

The Fleet Management solution includes two separate projects: one for the base model and the other one for extensions to the base model. The project named FleetManagement Migrated demonstrates how a migrated application might appear after migrating code from Dynamics AX 2012. This version shows how forms that have been migrated from Microsoft Dynamics AX 2012 R3 work on a web client. These forms have been created using automated migration tools and some other manual migration steps in Visual Studio. These forms bind to X++ tables and use the X++ programming model. The project named FleetManagement Discounts (or FleetManagementExtension) demonstrates how to use extensions to customize an application. This project extends the Fleet Management sample by extending controls and tables, handling data events, and replacing business logic using a plug-in. The tutorials that accompany this article provide a more-detailed look at the Fleet Management sample. These include a Fleet

Development tools in Visual Studio

Article • 04/14/2023

What are the development tools?

Application development is carried out in Visual Studio. The development tools support all of the development tasks, including debugging and local testing scenarios. Visual Studio is the exclusive integrated development environment (IDE) for development. All of your application development work will be performed with it. This section is an overview of the main features that are added to Visual Studio when the development tools are installed.

Visual Studio 2019 is supported beginning with platform updates for version 10.0.21 of finance and operations apps.

While working in Visual Studio, you will receive recurring feedback requests regarding new features. To prevent the feedback requests from appearing in Visual Studio, run the following PowerShell command from a developer's machine: Set-ItemProperty HKCU:\Software\Microsoft\Dynamics\AX7\Development\Configurations -Name ProvideFeedback -Value "No"

Application Explorer

In Visual Studio, the model store is represented by the Application Explorer. On the **View** menu, click **Application Explorer** to open it. Use the Application Explorer to browse and interact with the elements in the model store that define the applications. The following illustration shows the Application Explorer. For more information, see Application Explorer.

Development tools tutorial

Article • 06/14/2023

This tutorial tours the Fleet Management solution in Visual Studio and introduces you to the development tools.

In this tutorial, you'll take a tour of the Fleet Management solution in Visual Studio. You'll see how a project is organized in the Visual Studio development environment. Much of what you'll see uses standard Visual Studio features, plus you will notice that we've added some new customized features. Along the way we'll point out some of these new and customized features and how they ease development. This tutorial will focus on:

- Visual Studio projects and their features.
- Files used in development.

Prerequisites

This tutorial requires you to access the environment using Remote Desktop and to be provisioned as an administrator for the instance.

Setup

1. Start Visual Studio using **Run as an administrator**.
2. On the **File** menu, point to **Open** and then click **Project/Solution**.
3. Browse to the **Desktop,** and then open the **FleetManagement** folder. If the solution file is not on your computer, the steps to create it are listed in End-to-end scenario for the Fleet Management sample application.
4. Select the solution file named **FleetManagement**. The file type listed is Microsoft Visual Studio Solution (**SLN** file).
    - The fleet management solution file is available on the downloadable
    - VHD.
5. Click **Open**. The solution may take some time to load.

View the FleetManagement model

Application checker

Article • 08/12/2022

The application checker tool is a set of technologies that allow you to gain insight into your application code (both source and metadata) in ways that have not been possible before. The technology is based on representing both source code and metadata in XML and providing rich search facilities by using the XQuery language to express declarative queries over the source code. The current implementation runs inside a BaseX repository, which runs locally on the developer's box.

For information about how to install and use the application checker, see Dynamics365FO-AppChecker.

Feedback

Application Explorer

Article • 08/12/2022

This article reviews Application Explorer, and the various views and filtering methods in it. The article also describes how to work with elements in Application Explorer.

Application Explorer

Application Explorer is the tool that you use to find the elements that you want to add to a project so that you can work with them. To access Application Explorer, on the **View** menu, click **Application Explorer**. An important difference between Application Explorer and the Application Object Tree (AOT) in the MorphX environment of Microsoft Dynamics AX 2012 is that you don't use Application Explorer to add or edit model elements. Instead, you use it to view elements, view code, find references to a selected element, and add elements to a project. To create, design, edit, and build model elements, you must use a finance and operations project.

Application Explorer views

The content in Application Explorer can be organized in two ways. In the *classic view*, all the elements from every model are grouped according to type. This view resembles the way that the AOT was organized in Dynamics AX 2012. The following illustration shows the classic view.

Build and debug projects

Article • 06/11/2024

In this tutorial, you'll learn about using the tools in Visual Studio to analyze and debug code in the Fleet Management application. You'll go through a simple developer scenario in which you will set breakpoints, modify some code, and build the result.

Prerequisites

Previous experience with code and Visual Studio is helpful to get the full benefit of this tutorial. This tutorial requires you to access the environment using Remote Desktop and that you be provisioned as an administrator on the instance.

Key concepts

- The debugger in Visual Studio is used to analyze and debug code for your projects.
- The standard features of the Visual Studio debugger are available to use when you're examining the running application. These features include modifying values of variables, setting breakpoints, and so on.
- IntelliSense and other features of Visual Studio are vital to efficient code editing and comprehension.
- Development is an iterative process. After making modifications to the code, the project will be built, and changes can be tested.

Scenario

The rental company has had unfortunate events when customers rent cars using credit cards that are past the expiration dates. You, the developer, are tasked with revising the application to help prevent this situation. You'll identify the problem by using the debugger in Visual Studio. After the problem is identified, you'll edit some code to implement a fix. Finally, you'll build the project and validate that the fix was successful.

Run to a breakpoint

1. On the desktop, double-click the Visual Studio shortcut to open the development environment.

2. Open the FleetManagement solution. On the **File** menu, point to **Open**, and then click **Project/Solution**.

Build operations

Article • 08/12/2022

This article reviews the process to build projects and full build of model packages.

The elements of a model must be built so that they can be used by the application. You can build the elements in a project. You can also build all the elements in a model. The following actions are performed during a build operation:

- Metadata validation
- X++ code validation
- Best practice checks
- Report RDL generation
- Compilation, IL generation, and creation of the .NET assemblies
- Label assembly generation and deployment of other resource files
- Database synchronization

Build a project

When you build a project, only those elements that are new or that have changed are built. To build a project, follow these steps.

1. In Solution Explorer, select the project.
2. On the **Build** menu, click **Build <project name>** to start the build process. Alternatively, right-click the project in Solution Explorer, and then click **Build**.

During the build process, you might notice that some elements that are built aren't part of the project. This behavior is required because of the way that assemblies are created. When you build an element, you're actually building the .NET module that the element is included in. A single .NET module contains multiple model elements, and a single assembly contains multiple .NET modules. The assembly can be created only if all the .NET modules in the assembly have been built and are up to date. If any elements in any of the .NET modules for an assembly haven't been built or aren't up to date, they will be built, even if they aren't included in the current project.

> ⓘ **Note**
>
> If you delete an element from a project, you must rebuild the project or perform a full build on the model before the deletion takes effect.

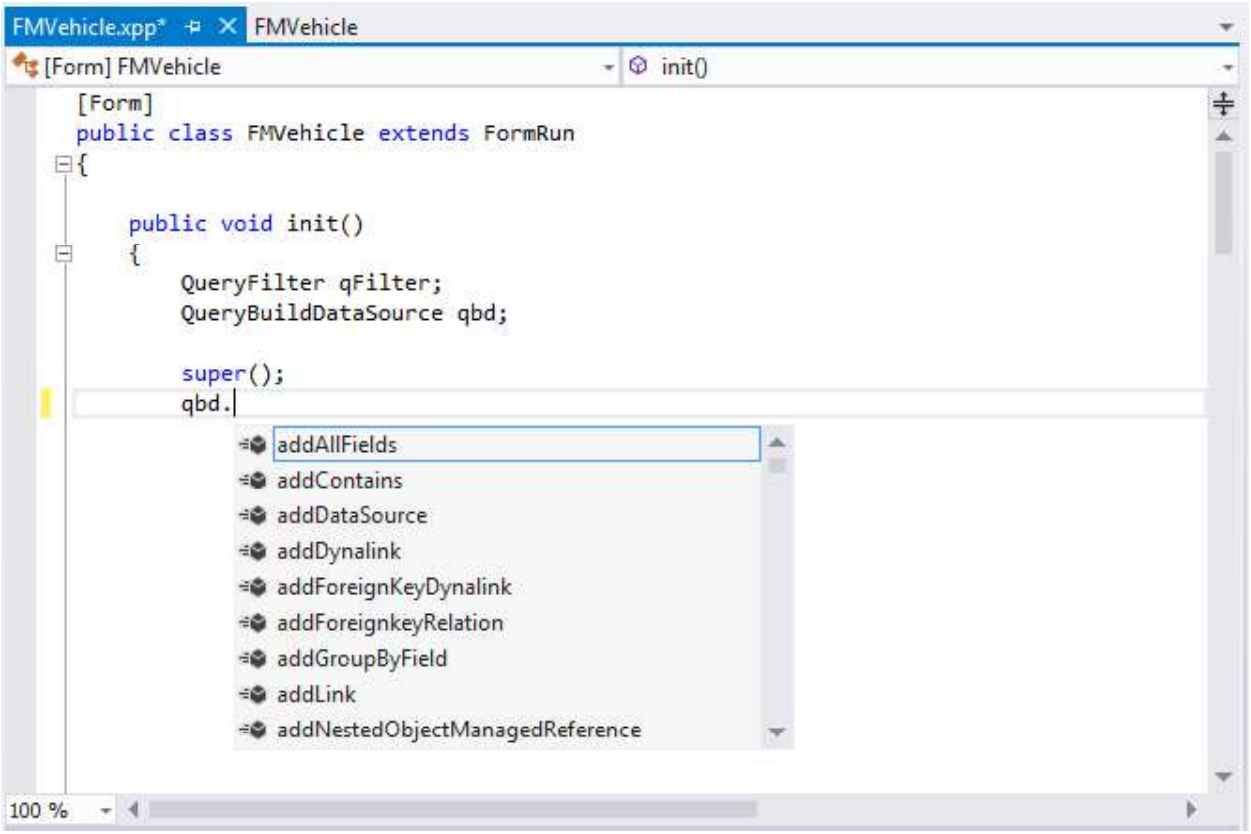
Code editor features

Article • 08/12/2022

This article describes the code editor for Visual Studio.

Code editor

You use the code editor in Microsoft Visual Studio to write the X++ code for your applications. The X++ language is fully integrated into the Visual Studio environment. As you write your X++ code, you will see the familiar features of the Visual Studio code editor. For example, IntelliSense is displayed to help you write the code. You can also navigate to methods and classes in the code editor by using the navigation drop-down menus at the top of the code editor window.



Other features, such as collapsible sections, are also available.

Opening the element designer

You can open the element designer that corresponds to the current X++ source code by right-clicking in the code editor and then selecting **Open Designer**.

Create models and data model elements overview

Article • 08/12/2022

In this tutorial, you'll use Visual Studio's Dynamics 365 menu to create a new model named **Fleet Management tutorial**. You'll also create and edit new model elements.

Prerequisites

This tutorial requires that you have access to an environment, and that you be provisioned as an administrator

Keywords

- **Model** - You configure your model to refer to two other models. This enables your model to reference metadata and code elements that are in other packages.
- **Project -** You create a project and then associate your project to your new model. You add elements to your project, which are also added to your model. Specifically, you add an extended data type (EDT). You also add a table that you populate with fields and a method.
- **Designer -** Each time you add an item to your project, a designer is displayed that is tailored to the item type you selected. The **Properties** window adjusts each time a different node of the designer is highlighted. You make updates in the designers and in the **Properties** window.
- **EDT -** Extended data type.

Create the Fleet Management tutorial model

1. Start Visual Studio using **Run as administrator**.

2. From the **Dynamics 365** menu, select **Model Management** > **Create model** to open the **Create model** wizard.

3. Enter the following values for model parameters.

Expand table

Naming guidelines for extensions

Article • 06/13/2024

High level guidance: use prefixes to reduce conflicts and improve identification

Naming model elements

Every element in a model must have a name that is unique across all models at installation time. However, at installation time, you don't know the names of all the models that your model might be installed together with. To accommodate this situation, every element name should include a prefix that is specific to your solution. By including this prefix when you name elements in your model, you significantly reduce the risk of naming conflicts.

- If a model contains multiple solutions, each solution in the model can be identified by a different prefix.
- You must carefully choose the prefix to minimize the risk that other models from other parties use the same prefix for their elements.

When you extend functionality in other models, elements that are being extended already contain a prefix. However, you should not add your prefix to the extension elements, so that the names include multiple successive prefixes. Instead, you should include your prefix or another term or abbreviation as an infix when you name extension elements.

Naming extensions

An extension element, such as a table extension, view extension, or form extension, must have a unique name that minimizes the risk of conflicts with extensions in other models. To minimize the risk of conflicts, the name should include a term, abbreviation, or infix that distinguishes the extension from other extensions to the same element in other models.

- Include either the name of the model where the extension element resides or the prefix that the extension is associated with. For example, a Warehousing module extends the HCMWorker table and uses the **WHS** prefix in the name of all other elements. In this case, the extension might be named **HCMWorker.WHSExtension**. Notice that the prefix that is used to name other elements in the module is inserted as an infix in the name. As another example, an extension of the ContactPerson table in the **ContosoCustomizations** model might be named

Turn off model customization and deprecate functionality

Article • 06/13/2024

This article describes the process of disabling customization of a model. By following this process, you make it ineligible for over-layering. Developers will still be able to extend that model. This article also describes how you can deprecate obsolete functionality.

Typically, you build an application in models that depend on other models. At the very least, your models will depend on the Application Platform model that is provided. Finance and operations applications run on the Microsoft Azure cloud platform. In other words, it runs off-premises, in data centers that are managed by Microsoft. Because we can't support changes from a large number of vendors in the fundamental models, your applications can no longer over-layer artifacts in those models. Therefore, to build your applications, you must use extensions instead of over-layering. Even though all the artifacts in the models that you depend on are available for documentation purposes, you can't compile someone else's intellectual property and run it in the cloud.

Locking customizations

Transitioning a model from over-layering to extensions involves three steps:

1. You set a property that causes instances of over-layering to be flagged as warnings. This step is sometimes known as "soft-locking."
2. You have a period when you can burn down the warnings by using extensions instead of over-layering.
3. When you have resolved all the warnings, it's time to make over-layering a hard error that causes compilation to fail. This step is known as "hard-locking." When a model is hard-locked, the tooling that is required for over-layering can't be used for that model. Additionally, you can't have more than one model in a given package.

Currently, there is no tooling that you can use to manipulate the property that disables customizations. Instead, you must add the **Customization** XML element to the model descriptor file, as shown in the following example. You can find the model descriptor file at ...\<packages>\<package name>\Descriptor\<model name>.xml.

XML

Customization Analysis Report (CAR)

Article • 05/30/2023

This article describes how to generate a Customization Analysis Report for your model. It also describes some best practice rules that are included in the report, and provides suggestions for fixing errors and warnings that are associated with these rules.

What is the Customization Analysis Report?

The Customization Analysis Report is a tool that analyzes your customization and extension models, and runs a predefined set of best practice rules. The report is one of the requirements of the solution certification process. The report is in the form of a Microsoft Excel workbook.

How to generate the report

The xppbp.exe tool is located in c:\packages\bin or I:\AosService\PackagesLocalDirectory\bin. To generate the Customization Analysis Report, run the following command in a development environment.

Console

```
xppbp.exe -metadata=<local packages folder> -all -model=<ModelName> -
xmlLog=C:\BPCheckLogcd.xml -module=<PackageName> -car=<reportlocation>
```

If your custom model references an ISV model, then you must include the `-PackagesRoot` parameter, for example:

Console

```
-packagesroot=K:\AosService\PackagesLocalDirectory
```

Example

Console

```
xppbp.exe -metadata=C:\Packages -all -model="MyAppSuiteCustomizations" -
xmlLog=C:\temp\BPCheckLogcd.xml -module="ApplicationSuite" -
car=c:\temp\CAReport.xlsx
```

Element designers

Article • 08/12/2022

This article reviews the element designers and explains how to use them. The tools contain designers for each kind of element in the program. You use these designers when you create or modify elements.

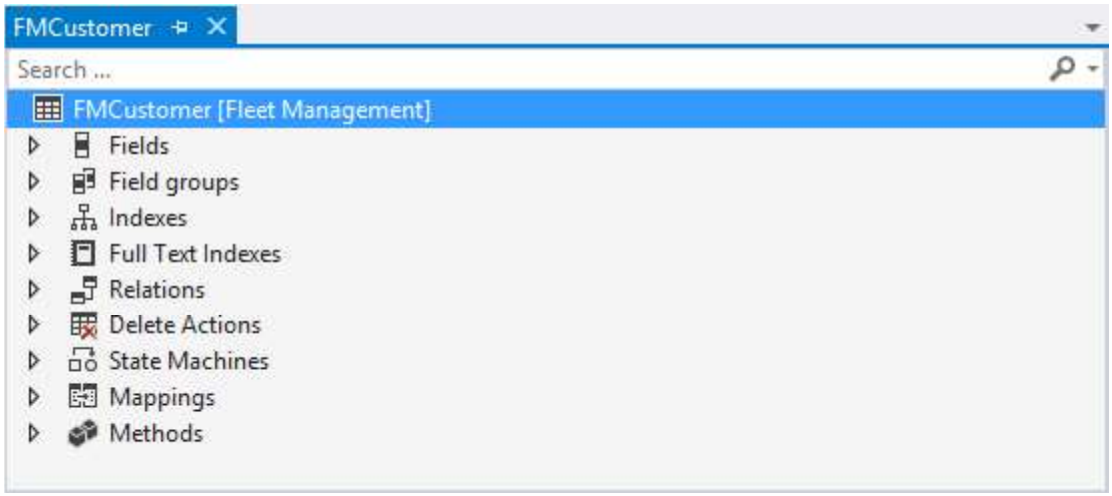
Open an element designer

To open an element designer, follow these steps.

1. Find the element in the project or in Application Explorer.
2. Right-click the element. In Application Explorer, click **Open designer**. In the project, click **Open**.
3. Expand the nodes in the element designer to see the details about the element.

Node properties

When you select the individual nodes in the element designer, the **Properties** pane in Visual Studio shows the various properties for that node. Most of the characteristics of an element are controlled by these properties. For example, the following illustration shows the element designer for the FMCustomer table. Notice that the top-level node is selected.



The following illustration shows the set of properties for the table, which corresponds to the top-level node that is selected.

Commands for determining how elements are used

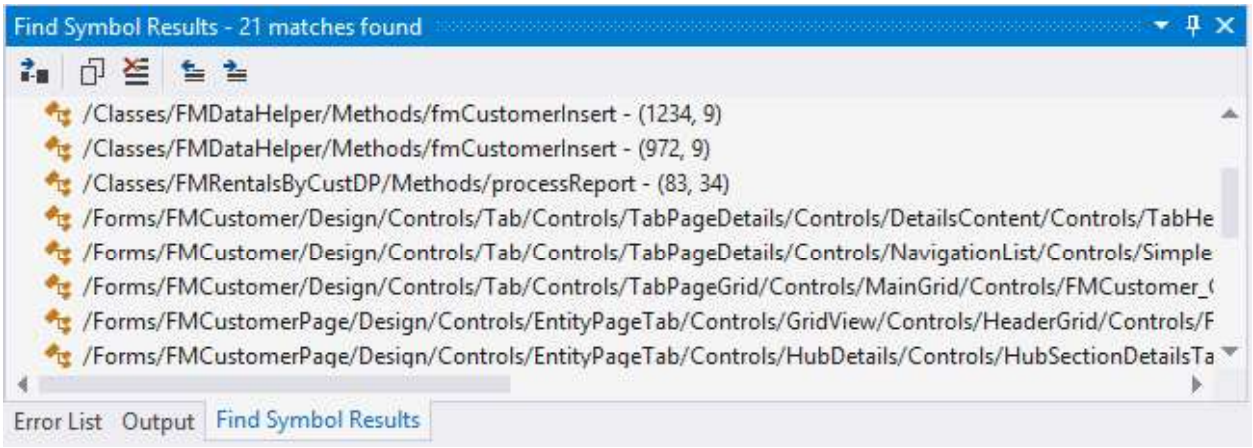
Article • 08/12/2022

This article reviews the commands that have been added to the Microsoft Visual studio Tools to help you determine how elements are used in an application.

Because of the large number of elements in a typical application, commands have been added to the Microsoft Visual Studio Tools to make it easier to determine how an element is used.

Finding where elements are used

During build operations, cross-reference information is generated that can be used to show how elements are used. You can right-click an element and then click **Find References** to display a list of the locations where that element is used. When you click one of the items in the list, the designer for the element opens.



Viewing a reference diagram

When you right-click some higher-level elements, such as tables, the **View Reference** command is available. This command produces a graphic that shows the elements that are related to the current element. You can right-click the items in the graphic and then click **Go To Definition** to navigate to those elements.

Export and import models

Article • 08/12/2022

Model files let you distribute models to customers and partners, and can be installed in development environments. They are key components of a Lifecycle Services (LCS) solution. Model files contain a model descriptor file, metadata, source code, and referenced .NET assemblies (when applicable). This article describes how to export a model into a model file, install a model file, and delete a model in a development environment.

Export a model into a model file for distribution

To export an existing model into a model file, use the ModelUtil.exe tool and the **-export** directive. This tool is located in the packages bin folder (typically, c:\packages\bin or i:\AosService\PackagesLocalDirectory\bin).

Console

```
ModelUtil.exe -export -metadatastorepath=[path of the metadata store] -
modelname=[name of the model to export] -outputpath=[path of the folder
where the model file should be saved]
```

Example

Console

```
ModelUtil.exe -export -metadatastorepath=c:\packages -
modelname="FleetManagement" -outputpath=c:\temp
```

The preceding example creates an .axmodel file under c:\temp. Typically, you then upload the model file to the Asset Library of the customer project or the Microsoft Dynamics Lifecycle Services (LCS) solution project.

Install a model in a development environment

To install a model file in a development environment, use the ModelUtil.exe tool and the **-import** directive.

Console

Metadata search in Visual Studio

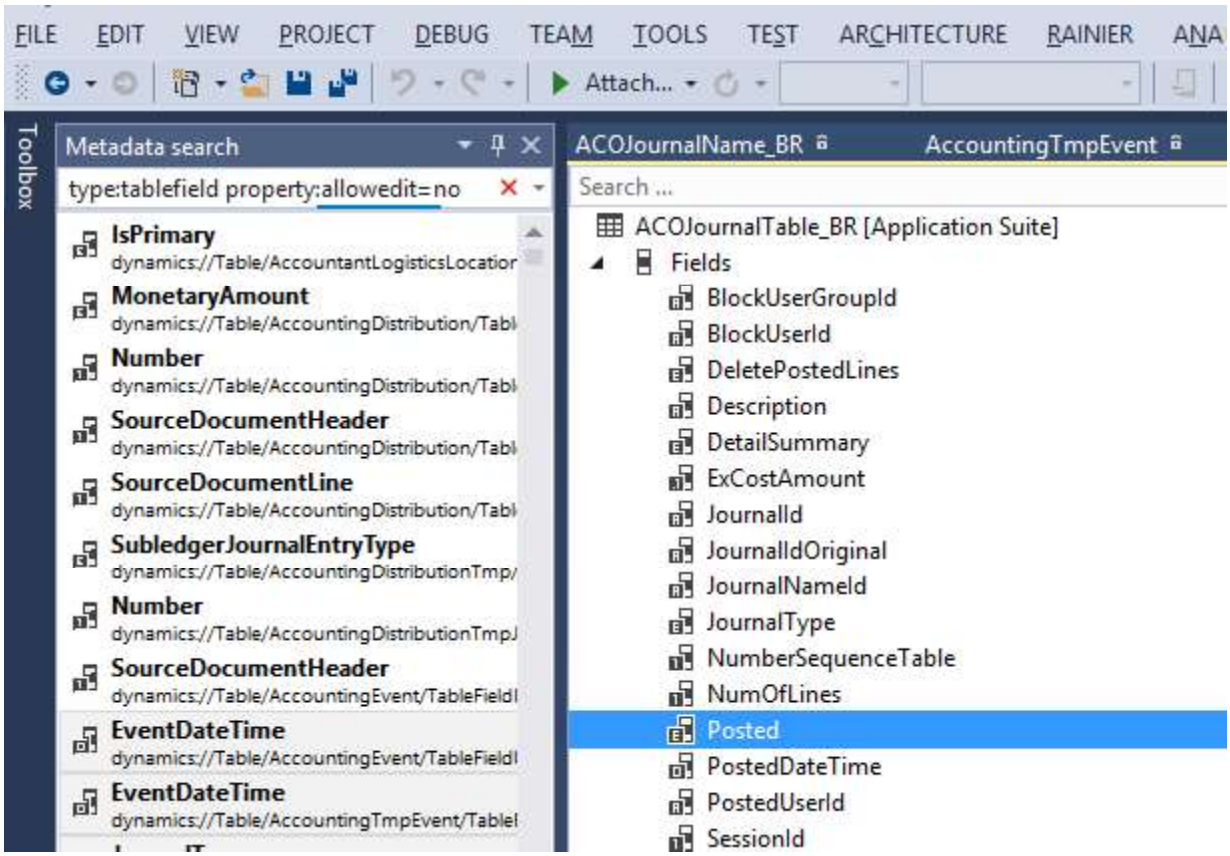
Article • 08/12/2022

This article describes how to use metadata search to search your code and metadata for arbitrary patterns and content.

Given the large volume of the code base and metadata, it is often necessary to find things in the code that meet a certain criteria. For example, you might not know the name of the metadata element that contains the pattern or meets the criteria. Metadata search is exposed in Visual Studio through two user interfaces: the Metadata Search tool window and the Navigate To window.

Metadata search tool window

You can access the Metadata search tool window from the **Dynamics 365** > **Metadata Search** menu command. Enter your search query to start the search. Results will start populating in the window asynchronously as you type. You can double-click any result line to navigate to the corresponding X++ code or metadata that matches your search query.



You can also select one or more results, right-click, and then add these elements to a project. You don't need to wait for the search to complete before you start interacting with the search results.

Models and packages

Article • 08/12/2022

This article describes the concept of models and packages. It also explains how to use the development tools in Microsoft Visual Studio to create new models, how to update the parameters of existing models, and how to visualize dependencies between models.

To work with models in the model store, you use tools in Microsoft Visual Studio. You can create new models and change parameters for existing models.

Conceptual overview

A model is a group of elements, such as metadata and source files, that typically constitute a distributable software solution and includes customizations of an existing solution. A model is a design-time concept, for example a warehouse management model or a project accounting model. A model always belongs to a package. A package is a deployment and compilation unit of one or more models. It includes model metadata, binaries, and other associated resources. One or more packages can be packaged into a deployable package, which is the vehicle used for deployment on runtime environments.

Creating a new model

You use the **Create model** wizard to create new models. You can access this wizard from **Model Management** on the **Dynamics 365** menu. You can create two types of models:

- **A model that is deployed in its own package –** You can use this type of model to create new model elements, and extend the metadata and business logic of referenced models. The wizard lets you select the referenced models. This type of model is compiled into its own assembly and binaries, and will simplify and reduce the cost of upgrades, deployment, and application lifecycle management in general.
- **A model that is a part of an existing package –** You can use this type of model to temporarily use legacy features such as overlayering source code and metadata. This feature is considered legacy and is supported only to upgrade from legacy versions.

In the **Create model** wizard, select **usr** for the layer. This layer will store user customizations. If needed, you can patch your customizations using the **usp** layer. If

Finance and operations project type in Visual Studio

Article • 08/12/2022

The finance and operations project type is part of the development tools. This project type resembles other projects in Visual Studio. It helps you organize and manage the elements that you're working with for a model. For example, the project can have folders that help you group the elements. A Visual Studio solution can contain multiple projects. There is one important constraint for a project: it can contain elements from only one model. If you must work with elements from different models, you must use multiple projects in your Visual Studio solution.

Create a new project

To create a new, empty project, follow these steps.

1. On the **File** menu, point to **New**, and then click **Project**.
2. In the list of template types, expand the **Installed** node.
3. Expand the **Templates** node.
4. Select the **finance and operations** category.
5. Select the **Operations Project** template.
6. Enter the name and location for the new project.
7. Specify whether you want to create a new solution or add the project to the current solution.
8. Click **OK**.

Every project has several important properties. To set the properties for a project, right-click the project in Solution Explorer, and then click **Properties**. The following table describes these properties.

Expand table

Property	Description
Startup Object type	The type of object that will be used as the Startup Object when the project is run. The following types are available: Form Class Output menu item
Startup Object	The object that will be invoked when the project is run.

Tools add-ins for Visual Studio

Article • 08/12/2022

This article reviews the Add-ins infrastructure that has been added to Microsoft Visual Studio, so that developers can more easily add tools for development.

A lot of great tools have been added to Microsoft Visual Studio to support development. However, there will always be additional tools to meet specific requirements. To make it easier to add these additional tools, an **Add-ins** infrastructure has been provided for developers. The additional tools are available in two places:

- The **Add-ins** submenu on the **Dynamics 365** menu
- The **Add-ins** submenu on the shortcut menu in the element designer

To make it easier to create your own add-ins, you can select the **Dynamics Developer Tools Add-in** project type when you create a new project in Visual Studio. This project type has the infrastructure that is required to implement an add-in.

For more information on add-ins, see:

- Visual Studio add-ins that support form patterns

Feedback

Was this page helpful? Yes No

Provide product feedback

Update the Visual Studio development tools

Article • 05/17/2024

This article explains how to update the development tools.

Use this tutorial to update your Visual Studio development tools with a new version. It explains how to uninstall your existing Visual Studio development tools and install the new extension. The new extension is in the form of an installable VSIX file. This file is a part of the binary hotfix available on the Dynamics 365 Lifecycle Services site. The VSIX file is located in the **DevToolsService\Scripts** folder of the binary hotfix package.

While working in Visual Studio, you might receive recurring feedback requests regarding new features.

To prevent the feedback requests from appearing in Visual Studio, run the following PowerShell command from a developer's machine:

PowerShell

```
Set-ItemProperty
HKCU:\Software\Microsoft\Dynamics\AX7\Development\Configurations  -Name
ProvideFeedback  -Value "No"
```

> **Note**
>
> You do not need to follow the instructions in this article if you are upgrading your finance and operations platform to Platform update 4 or newer. It is an automatic step that is part of the platform upgrade process.

Prerequisites

To enable the use of finance and operations apps development tools, the **Visual Studio extension development** workload must be installed in your Visual Studio with the **Modeling SDK** option included.

If the workload isn't installed, follow these steps to add it.

1. Open your Visual Studio Installer app and select **Modify** on your Visual Studio installed app.

Visual Studio requirements for X++

Article • 07/02/2024

This article lists the Visual Studio components that are required to run the **Visual Studio** extension for X++.

> **Note**
>
> We don't recommend installing Visual Studio manually on the downloadable virtual hard drive (VHD) or virtual machines deployed from Lifecycle Services (LCS). Instead, we strongly recommend that you download or deploy a new virtual machine. Virtual machines deployed with versions 10.0.13 or later all have Visual Studio and its prerequisites installed, and come with other updates outside of Visual Studio to help keep the machines compatible and secure.

Visual Studio editions

The supported editions of Visual Studio include:

- Visual Studio 2022 Professional
- Visual Studio 2022 Enterprise

> **Note**
>
> Different Visual Studio editions have different licensing requirements and costs. For more information, see Visual Studio.

Required Visual Studio components

The following table lists the required Visual Studio components.

Expand table

Type	Name	Required	Notes
Workload	.NET desktop development	Yes	
Individual	Modeling SDK	Yes	

Debug X++ code by using the debugger in Visual Studio

Article • 08/12/2022

This article reviews how you can debug X++ code by using the debugging feature in Microsoft Visual Studio.

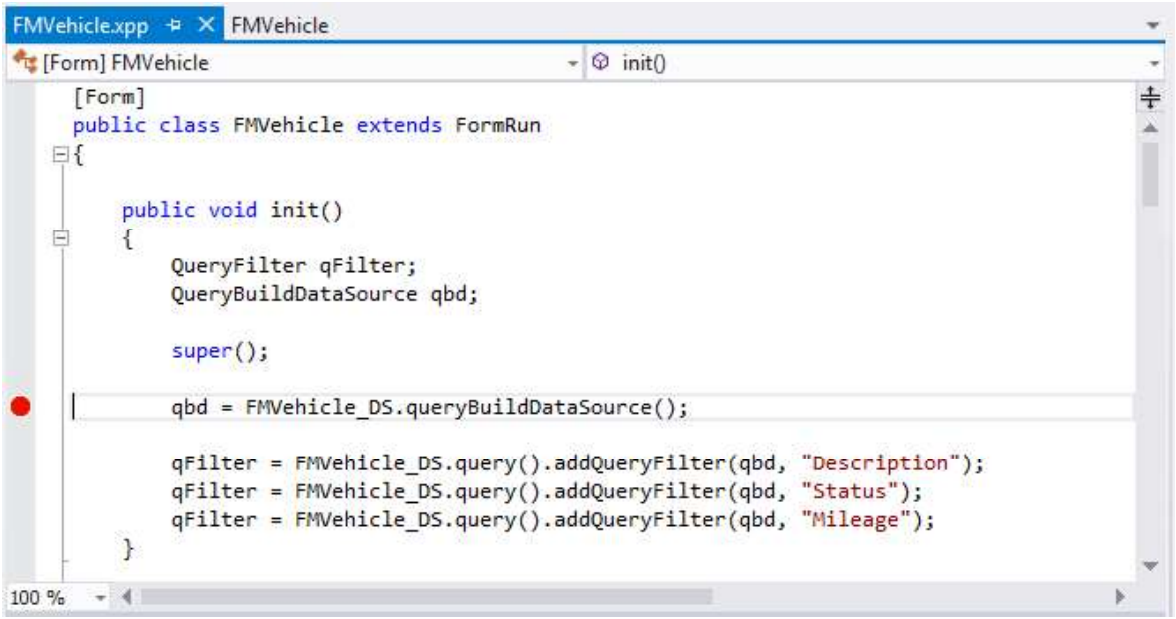
To debug X++ code, you use the debugger in Microsoft Visual Studio. The process is similar to the process that is used for any other application that is created in Visual Studio. For example, the standard tools for examining the application are available when your code is stopped at a breakpoint.

Debug your code

To debug X++ code, follow these steps.

1. In Visual Studio, open the X++ code to debug.

2. Find the line or lines where you want execution to stop, and set breakpoints in those lines. To set a breakpoint in a line, click in the left column of the code editor or press F9 while the cursor is on that line. A red dot indicates that a breakpoint has been set.



3. Set a startup project and a startup object. Startup objects can be any form, any class that has the **main** method, or any menu item. You can set the startup object in the **Properties** pane for the project. Alternatively, right-click the element in Solution Explorer, and then click **Set as Startup Object**.

EventHandlerResult classes in request or response scenarios

Article • 08/12/2022

Delegate methods and delegate handler methods can be declared to support a request/response scenario, where the delegate calling logic requests the subscribers to provide a response. To support this scenario the **EventHandlerResult** class is most often passed as a parameter, and the delegate handler methods provide their result using one of the result methods on the class. However, the **EventHandlerResult** class can only contain a single result, so if multiple subscribers provide their individual result, the last respondent wins, and the results from the previous subscribers are overwritten.

Before the functionality described in this article was introduced (platform update 5), there was no mechanism to ensure that, at most, a single subscriber provided a result, and that no results were lost if there were multiple subscribers.

Ensuring, at most, one response

In platform update 5, the **EventHandlerResult** class has an additional static constructor which ensures that the logic fails if more than one subscriber provides a result. The new constructor is named **newSingleResponse**. When instantiating an **EventHandlerResult** object using this method, the framework will throw an exception as soon as a second delegate handler method attempts to provide a result.

```
X++
```

```
EventHandlerResult result = EventHandlerResult::newSingleResponse();
this.validateWarehouseTypeDelegate(this.WarehouseType, result);
```

IEventHandlerResultValidator interface

The validation in the **EventHandlerResult** class is handled by injecting an object of a type that implements the **IEventHandlerResultValidator** interface. When instantiating the **EventHandlerResult** object using the **newSingleResponse** static constructor, an **EventHandlerSingleResponseValidator** object is instantiated and injected into the **EventHandlerResult** object, and the injected object becomes responsible for validating any result provided to the **EventhandlerResult** object. Other validation classes can be implemented by having the class implement the **IEventHandlerResultValidator** interface, and injecting it into the **EventHandlerResult** class by instantiating the

Write business logic by using C# and X++ source code

Article • 06/13/2024

The primary goal of this tutorial is to illustrate the interoperability between C# and X++. In this tutorial, you'll write business logic in C# source code and in X++ source code.

In this tutorial, you'll write business logic in C# source code and in X++ source code. You'll get experience with the following:

- New tools in Visual Studio.
- The handling of events in C#.
- The use of Language Integrated Query (LINQ) in C# to fetch data.

Prerequisite

This tutorial requires that you access the environment using Remote Desktop, and be provisioned as an administrator on the instance.

> ⓘ **Note**
>
> Debugging support for the C# project does not work if the **Load symbols only for items in the solution** check box is selected. Since this option is selected by default, it must be changed prior to running the lab. In Visual Studio, click **Dynamics 365** > **Options**, and clear the **Load symbols only for items in the solution** check box.

Scenario

Too many cars have been rented to drivers who have a history of unsafe driving habits. The Fleet Management rental company needs to check driving records from external sources. Upper management has decided to subscribe to a service that is hosted by the Department of Transportation (DOT), which is the legal entity that manages drivers' licenses and associated information. This service retrieves the number of citations for the given unique license number. It's not easy to call external services directly from X++ source code. Visual Studio has tools for generating the "code-behind" (in C#) that calls the services, and these tools make the development effort easy. The obvious choice would be to leverage Visual Studio to write the code. However, in this tutorial your code won't actually call an external service, because the logistics are beyond the scope of the

Language Integrated Query (LINQ) provider for C#

Article • 06/13/2024

This article discusses the LINQ provider.

LINQ (Language Integrated Query) is a set of classes and methods that enable you to access data that is stored in a variety of places and formats. The LINQ framework is the standard for accessing data in managed languages. LINQ presents to programmers a unified and consistent API for data access from heterogeneous data sources, such as:

- In-memory object graphs
- Active Directory entries
- Flickr pictures and XML
- SQL Server

The LINQ provider allows the user to access business data by using .NET managed languages.

Two syntactical mechanisms for accessing LINQ

There are two syntactical approaches for using LINQ, as described in the following table.

Expand table

Approach	X++	C# and Visual Basic
LINQ by standard method call syntax.	Impractical. Language support for generics is vital for LINQ and is not supported in X++.	Available, requires lambda syntax.
LINQ by specialized syntax that is understood by the compiler.	Not available.	Available, easier to use.

There are two syntactic mechanisms for accessing the LINQ provider in C# (or in Visual Basic):

- By standard, or fluent, method call syntax.
- By specialized syntax that the C# compiler has been enhanced to understand as equivalent to the LINQ method calls. (Such syntax is sometimes called “syntactic sugar”.)

Write best practice rules

Article • 06/13/2024

This article describes how you can author best practice rules in C#, for both metadata and X++ code. Best practice checks are run by the compiler. You can run them in daily builds to catch objectionable practices that are unacceptable in shipping code. The features can also be used to author simple one-of tools to gather information about the application.

You can also use best practice rules to author simple tools that gather information about your application. The framework is built on top of a managed framework called XLNT (shorthand for X++ LaNguage Toolkit). You can use the framework to build custom tools that extract information from, and modify, X++ code. There are two types of best practice rules: rules that deal with metadata and rules that deal with source code.

Code Best Practice framework

The Code Best Practice Framework (CBPF) enables you to write your own tools for analyzing X++ source code. These rules diagnose things that you consider to be problems with X++ source code. This section describes the foundation of the Best Practice functionality. This information is helpful for understanding the later sections that describe creating your own rules in greater detail. It is also helpful to developers who want to code rules that are more complex than those demonstrated in this document. The CBPF API lets you focus on the rule you are expressing, without having to deal with infrastructure issues. You don't need to read tokens and piece them together to create something intelligible from them. Instead, the CBPF provides the following parts:

- A parser that builds an Abstract Syntax Tree (AST) from X++ source code.
- A pipeline that runs a sequence of passes over the X++ code.
- A number of prebuilt passes. The first pass is the parsing of the source code.
- Infrastructure to read metadata.

Because rules are based on ASTs, it is important to understand that concept before starting to write rules.

The parser and ASTs

The parser reads X++ code and produces an AST from it if it does not contain egregious syntax errors. The parser has a built-in error recovery scheme, so it can recover from

Application Explorer properties

Article • 08/12/2022

This article describes the properties that appear in the Properties window of Microsoft Visual Studio for items in Application Explorer.

Many nodes in Application Explorer represent elements that have properties associated with them. You can read or modify these properties in the **Properties** window of Microsoft Visual Studio.

System and common properties

Most application objects in Application Explorer have a standard set of system properties. These system properties are read-only. You can use the **Properties** window to view the properties for any item in Application Explorer. To open the **Properties** window, right-click a node in Application Explorer, and then click **Properties**. On the **Categories** tab of the **Properties** window, many system properties are listed under the **Statistics** node. This article lists additional common properties that are repeated on many, but not all, Application Explorer nodes. The following table shows the system properties that are found on almost all Application Explorer nodes. All these system properties are read-only.

Expand table

Property	Description
ChangedBy	The user who last changed the object (often the release version).
ChangedDate	The date when the object was last changed.
ChangedTime	The time when the object was last changed.
CreatedBy	The user who created the object.
CreationDate	The date when the object was created.
CreationTime	The time when the object was created.

The following table shows other common properties that are found on many, but not all, Application Explorer nodes.

Expand table

SysSetupConfigAttribute attribute

Article • 08/12/2022

X++ classes that implement the `SysSetup` interface are processed as part of database synchronization. Custom X++ classes that implement `SysSetup` are also run as part of database synchronization.

Attributes provide a powerful way to associate metadata, or declarative information, with code such as assemblies, types, methods, and properties. After an attribute is associated with a program entity, it can be queried at runtime by using reflection.

This article describes how to use the new `SysSetupConfigAttribute` attribute that the platform updates for version 10.0.23 of finance and operations apps introduce for X++ classes that implement the `SysSetup` interface.

Usage

The `SysSetupConfigAttribute` attribute must be added for all X++ classes that implement the `SysSetup` interface. It accepts two parameters:

- **ContinueOnError –** This parameter is of the `bool` type. If execution of the X++ class fails during synchronization, database synchronization will either fail or continue with the next steps, depending on the value of this parameter (`true` or `false`).
  - **true –** Database synchronization will continue with the next steps.
  - **false –** The overall database synchronization operation will fail and can't be resumed until the underlying issue is fixed.
- **Timeout –** This parameter is of the `int` type, and the range of values is from 1 to 600 seconds. It defines the time range that the database synchronization operation will run the `SysSetup` class during.

In the following code example, the `ContinueOnError` parameter is set to `true`, and the `Timeout` parameter is set to `300`.

X++

```
[SysSetupConfigAttribute(true, 300)]
class DemoClass implements SysSetup
{
```

SysSetup

Article • 05/01/2023

SysSetup is an interface that's used to identify the X++ classes that run during database synchronization. The X++ classes that implement the SysSetup interface run during the **PostTableAndViewSyncActions** step of database synchronization.

Onboard to SysSetup

1. In Microsoft Visual Studio, select **View** > **Application explorer (AOT)**.
2. Create a new class that implements the **SysSetup** interface.
3. Add the **loadData()** method in the class. The code in this method is run during database synchronization. (The code can also be a reference to another class or method.)
4. Add the SysSetupConfig and **SysSetupTable** attributes to the class. The **SysSetupTable** attribute takes the related table as the input.

The completed class should resemble the following example.

X++

```
[
    SysSetupTable(tablestr(DemoTable)),
    SysSetupConfig(true, 300, 1.1)
]
class DemoSetup implements SysSetup
{
    public void loadData()
    {
        <your code here...>
    }
}
```

> **Note**
>
> If you run database synchronization from Visual Studio, the SysSetup classes won't be run. For information about how to run database synchronization from the command line, see the next section.

Validate new classes

X++ language reference

Article • 12/16/2024

X++ is an object-oriented, application-aware, and data-aware programming language used in enterprise resource planning (ERP) programming and in database applications. It provides system classes for a broad range of system programming areas, highlighted in the following table.

Expand table

X++ language feature	Description
Classes	In addition to system classes, there are also application classes for managing many types of business processes. Reflection on classes is supported.
Tables	X++ programmers can access the relational tables. X++ includes keywords that match most of the keywords in standard SQL. Reflection on tables is supported.
User interface	Manipulation of user interface items, such as forms and reports.
Best practice checks	X++ code is checked for syntax errors during compile time. The compile process also performs best practice checks. Violations of best practices can generate compiler messages.
Garbage collection	The X++ runtime execution engines have automatic mechanisms to discard objects that are no longer referenced, so that memory space can be reused.
Interoperability	Interoperability between classes written in X++ and in C# (or other .NET Framework languages) is supported.
File manipulation	File input and output are supported, including XML building and parsing.
Collections	Dynamic arrays are supported and the X++ includes several collection objects.

X++ compiles to Microsoft .NET CIL (Common Intermediate Language)

X++ source code is compiled to Microsoft .NET CIL (Common Intermediate Language). CIL is what the .NET compilers for C# and Visual Basic generate. The advantages of compiling to CIL include:

- Your code runs much faster than in previous versions (AX2012 and earlier).

X++ variables

Article • 08/12/2022

This article describes variables in X++.

- A *variable* is an identifier that points to a memory location where information of a specific data type is stored. The size, precision, default value, implicit and explicit conversion functions, and range depend on the variable's data type.
- The *scope* of a variable defines the area in the code where an item can be accessed.
- *Instance variables* are declared in class declarations, and can be accessed from any methods in the class or from methods that extend the class.
- *Local variables* can be accessed only in the block where they were defined.
- When a variable is declared, memory is allocated, and the variable is initialized to the default value.
- You can assign values to both static fields and instance fields as part of the declaration statement.
- Variables can be declared anywhere in a code block in a method. They don't have to be declared at the beginning of a method.
- *Constants* are variables where the value can't be changed when the variable is declared. They use the **const** or **readonly** keyword.
- Constants differ from *read-only fields* in only one way. Read-only fields can be assigned a value only one time, and that value never changes. The field can be assigned its value either inline, at the place where the field is declared, or in the constructor.

When you declare variables of managed types that aren't authored in X++, you have two options. You can fully qualify the type names in the declaration by including the full namespace, or you can add a **using** statement to your file and then omit the namespace from the type name.

Variable examples

X++

```
// An example of two valid variable names.
str variableName;
CustInfo custNumber;

// An example of simultaneously declaring and initializing a variable.
real pi = 3.14159265359; // Assigns value of pi to 12 significant digits.

// An example of initializing an object by using the new method on the
```

X++ Primitive data types

Article • 08/12/2022

This article describes primitive data types in X++. The primitive data types in X++ are **anytype**, **boolean**, **date**, **enum**, **guid**, **int**, **int64**, **real**, **str**, **timeOfDay**, and **utcdatetime**.

anytype

The **anytype** data type is a placeholder for any data type. You should use variables of this type only as arguments and return values.

To use **anytype** as a variable, you must first assign a value to it, otherwise, a run-time error occurs. After you've assigned a value to **anytype**, you can't convert it to another data type.

Although you can use **anytype** variables in expressions, they're usually used as arguments and return types. The size, precision, scope, default value, and range of **anytype** depend on the conversion type that you assign to it. You can use **anytype** just as you use the data type that you convert it to. For example, if you assign an integer, you can then apply relational and arithmetic operators to the variable.

An **anytype** variable is automatically converted to a date, enumeration (enum), integer, real, string, extended data type (EDT) (record), class, or container when a value is assigned to the type. Additionally, the following explicit conversion functions can be used: **any2date**, **any2enum**, **any2int**, **any2real**, and **any2str**. You can't change the variable to another data type after you've converted it to **anytype**.

anytype examples

X++

```
// An example of using anytype variables.
public static str range(anytype _from, anytype _to)
{
    return queryValue(_from) + '..' + queryValue(_to);
}

// Another example of using anytype variables.
void put(int position, anytype data)
{
    record = conPoke (record, position, data);
}

public void AnytypeMethod()
```

X++ composite data types

Article • 08/12/2022

This article describes composite data types in X++. The composite data types in X++ are arrays, containers, classes as data types, delegates as data types, and tables as data types.

Array

An *array* is a variable that contains a list of items that have the same data type. The elements of an array are accessed by using integer indexes. You use a separate statement to initialize each element in an array. When you use a container data type or an array object to create a collection, you can initialize multiple elements by using a single statement. By default, all the items in an array have the default value of the data type in the array. There are three kinds of arrays: *dynamic arrays*, *fixed-length arrays*, and *partly on disk arrays*.

- **Dynamic arrays –** These arrays are declared by using an empty array option. In other words, they have only brackets ([]).
- **Fixed-length arrays –** These arrays can hold the number of items that is specified in the declaration. Fixed-length arrays are declared like dynamic arrays, but a length option is included in the brackets.
- **Partly on disk arrays –** These arrays are declared as either dynamic arrays or fixed-length arrays that have an extra option that declares how many items should be held in memory. The other items are stored on disk and are automatically loaded when they are referenced.

X++ supports only one-dimensional arrays. However, you can mimic the behavior of multiple array indexes. (For more information, see the Multiple array indexes section). Variables in objects and tables can be declared as arrays. For example, this functionality is used in address lines in the standard application. An array collection class lets you store objects in an array.

Array indexes begin at 1. The first item in the array is referenced as [1], the second item is referenced as [2], and so on. The following syntax is used to access an array element: **ArrayItemReference** = **ArrayVariable** [**Index**]. In this syntax, **ArrayVariable** is the identifier of the array, and **Index** is the number of the array element. **Index** can be an integer expression. Item zero [0] is used to clear the array. If a value is assigned to index 0 in an array, all elements in the array are reset to their default value.

An assignment of one entire array to another is performed by reference.

X++ collection classes

Article • 08/12/2022

The X++ language syntax provides two composite types: arrays and containers. These composite types are useful for aggregating values of primitive types. However, you can't store class objects in arrays or containers.

Collection classes are used to store objects. They let you create arrays, lists, sets, maps, and structs that can hold any data type, even objects. For maximum performance, the classes are implemented in C++ (they are system classes). Collection classes were previously known as *foundation classes*. The collection classes are **Array**, **List**, **Map**, **Set**, and **Struct**.

- Array – This class resembles the **array** type in the X++ language, but it can hold values of any single type, even objects and records. Objects are accessed in a specific order.
- List – This class contains elements that are accessed sequentially. Unlike an array, the **List** class provides an **addStart** method. Like the **Set** class, the **List** class provides the **getEnumerator** and **getIterator** methods. You can use an iterator to insert and delete items from a **List** object.
- Map – This class associates a key value with another value.
- Set – This class holds values of any single type. Values aren't stored in the sequence in which they are added. Instead, the **Set** object stores the value in a manner that optimizes performance for the **in** method. A **Set** object ignores any attempt to add a value that the **Set** object is already storing. Unlike the **Array** class, the **Set** class provides the **in** and **remove** methods.
- Struct – This class can contain values of more than one type. It's used to group information about a specific entity.

The constructor for every collection class except **Struct** takes a type parameter that is an element of the **Types** system enum. The collection instance can store items of that type only. The **Types::AnyType** enum element is a special case that can't be used to construct a collection object, such as a **Set** object. The **null** value can't be stored as an element in a **Set** object. Additionally, **null** can't be a key in a **Map** object. You can iterate through a collection object by using an iterator or enumerator. Here are typical examples that show how you can obtain an iterator.

X++

```
new MapIterator(myMap)
myMap.getEnumerator()
```

X++ extended data types

Article • 08/12/2022

This article describes extended data types in X++.

Extended data types are user-defined types that are based on the **boolean**, **int**, **int64**, **real**, **str**, and **date** primitive data types, and on the **container** composite type. An EDT is a primitive data type or container that has a supplementary name and additional properties. For example, you can create a new EDT that is named **Name** and base it on a string. You can then use the new EDT in variable and field declarations in the development environment.

You can also base EDTs on other EDTs. EDTs are standard data types, but they have a specific name and additional properties. EDTs undergo the same value and type conversions as the standard data types that they are based on. Here are the benefits of EDTs:

- Code is easier to read, because variables have a meaningful data type. For example, the data type is **Name** instead of **str**.
- The properties that you set for an EDT are used by all instances of that type. Therefore, EDTs help reduce work and promote consistency. For example, account numbers (**AccountNum** data type) have the same properties throughout the system.
- You can create hierarchies of EDTs. The EDTs can inherit the appropriate properties from the parent, and you can change other properties. For example, the **ItemCode** data type is used as the basis for the **MarkupItemCode** and **PriceDiscItemCode** data types.

Create an EDT

This feature isn't implemented as a language construct. To create an EDT, follow these steps.

1. In Solution Explorer, right-click on the project, point to **Add**, and then click **New item**.
2. In the **Add New Item** dialog box, select **Installed** and then **Artifacts** in the left pane.
3. In the middle pane, select the EDT type to create.
4. Enter a name, and then click **Add**.

Comments, using, and print statements

Article • 08/12/2022

This article describes statements in X++.

Comments

It's a good practice to add comments to your code. Comments make a program easier to read and understand. Comments are ignored when the program is compiled. Your comments can use either the **//** style or the **/*** style. However, a best practice is to use the **//** style for comments, and even for multiline comments.

X++

```
// This is an example of a comment.
/* Here is another example of a comment. */
```

print statements

You use the **print** statement to output text through **System.Diagnostics.WriteLine** to the Visual Studio **Output** window. During testing, the **print** statement is an alternative to the **Global::info** method, which shows text in the **Infolog** window. The following table compares the **print** statement and the **info** method.

Expand table

Feature	print statement	info method
Ease of invocation	The **print** statement automatically converts various data types into strings. It can convert multiple data types in one invocation.	The **info** method requires that the input parameter be a string.
Ability to copy contents to the clipboard	Text is easily copied from the **Output** window to the clipboard.	Text is easily copied from the **Infolog** window to the clipboard.
Typical usage	The **print** statement is used for convenience during testing. It can help you debug small issues without having to run a formal debugger.	The **info** method is appropriate for use in production.

Example of a print statement

X++ conditional statements

Article • 08/12/2022

This article describes conditional statements in X++. The conditional statements are **if**, **if**...**else**, **switch**, and the ternary operator (**?**). You use conditional statements to specify whether a block of code is executed. Different conditional statements offer advantages in different situations.

if and if...else statements

The **if** statement evaluates a conditional expression, and then executes a statement or a set of statements if the conditional expression is evaluated as **true**. You can use the **else** clause to provide an alternative statement or set of statements that is executed if the condition is evaluated as **false**. The syntax for an **if**...**else** statement is:

if (*expression* **)** *statement* [**else** *statement*]

In this syntax, both occurrences of *statement* can be compound statements (statements enclosed in braces). The *expression* in the parentheses (the conditional expression) can be any valid expression that is evaluated as **true** or **false**. All numbers except 0 (zero) are **true**. All non-empty strings are **true**. You can nest **if** statements. However, if the nesting of **if** statements becomes too deep, you should consider using a **switch** statement instead.

Examples of if and if...else statements

X++

```
// if statement
if (a > 4)
{
   info("a is greater than 4");
}

// if... else statement
if (a > 4)
{
   info("a is greater than 4");
}
else
{
   info("a is less than or equal to 4");
}
```

X++ loop statements

Article • 08/12/2022

This article describes loop statements in X++.

There are three loop statements: **for**, **while**, and **do**...**while**. A loop repeats its statement until the condition that is set for the loop is **false**. Within the loop statements, you can use **break** and **continue** statements.

for loops

The syntax of a **for** loop is:

for (*initialization* ; *test* ; *increment*) { *statement* }

The **for** loop repeatedly executes **statement** for as long as the conditional expression *test* is **true**. *statement* can be a block of statements. The body of the **for** loop (*statement*) might be executed zero or more times, depending on the results of *test*.

A **for** loop differs from other loops because an initial value can be assigned to a control variable, and because there is a statement for incrementing or decrementing the variable. These additions make a **for** loop especially useful for traversing lists, containers, and arrays because they have a fixed number of elements.

You can also apply a statement to each element and increment your way through the elements, setting the condition to test for the last element.

Example of a for loop

In the following code example, the items in an array of integers are printed.

X++

```
int integers[10];
for (int i = 0; i < 10; i++)
{
    info(int2str(integers[i]));
}
// The output is a series of 0's.
```

while loops

X++ exception handling

Article • 09/29/2022

This article describes exception handling in X++. You handle errors by using the **throw**, **try**...**catch**, **finally**, and **retry** statements to generate and handle exceptions.

An *exception* is a regulated jump away from the sequence of program execution. The instruction where program execution resumes is determined by `try...catch` blocks and the type of exception that is thrown. An exception is represented by a value of the **Exception** enumeration, or an instance of .NET's `System.Exception` class or a derived class. One exception that is often thrown is the **Exception::error** enum value. A common practice is to write diagnostic information to the Infolog before the exception is thrown.

The **Global::error** method is often the best way to write diagnostic information to the Infolog. For example, your method might receive an input parameter value that isn't valid. In this case, the method can throw an exception to immediately transfer control to a **catch** code block that contains logic for handling this error situation. You don't necessarily have to know the location of the **catch** block that will receive control when the exception is thrown.

throw statements

You use the **throw** keyword to throw an **Exception** enum value. For example, the following statement throws an error exception.

```
X++
```

```
throw Exception::error;
```

Instead of throwing an enum value, a best practice is to use the output of the **Global::error** method as the operand for **throw**.

```
X++
```

```
throw Global::error("The parameter value is invalid.");
```

The **Global::error** method can automatically convert a label into the corresponding text. This functionality helps you write code that can be localized more easily.

```
X++
```

SQL connection error X++ exception

Article • 08/12/2022

This article describes the SQL connection error exception types in X++.

TransientSqlConnectionError X++ exception

During an X++ SQL query execution, when a transient SQL connection error occurs on the server side, a TransientSqlConnectionError X++ exception will occur. Depending on the application requirements, the application should catch and handle the exception.

This exception usually occurs during a large transaction or when the database is under a lot of processing pressure.

The TransientSqlConnectionError exception is not catchable within the transaction. The X++ transaction that encounters this exception is canceled (calling **ttsAbort**) before the exception occurs. This means that you need to use the catch block to identify the transient SQL connection error instead of a generic X++ error exception, and then retry the outermost transaction or retry application code logic in a new session. This exception allows the application to be designed for transient server failures.

If an application transaction takes a long time to process, you can use multiple incremental delays to catch the TransientSqlConnectionError exception. Retrying your application code in a new session is most likely to succeed after you have caught the exception.

Example

X++

```
public static void LargeTransactionWrapper()
{
    try
    {
        LargeTransaction();
    }
    catch (Exception::TransientSqlConnectionError)
    {
        info("Caught transient SQL connection error, ttslevel=" +
int2Str(appl.ttsLevel()));
        // At this point, transaction is canceled
        // Code that indicates retry is possible
    }
    finally
```

X++ operators

Article • 08/12/2022

This article describes the operators supported in X++.

Assignment operators

An assignment changes the value of a variable or field. The following table shows the X++ assignment operators. There is no difference between prefix and postfix operators.

Expand table

Operator	Description
`=`	Assign the expression on the right of the equal sign to the variable on the left.
`+=`	Assign the current variable value plus the expression on the right to the variable on the left.
`++`	Increment the variable by 1.
`-=`	Assign the current variable value minus the expression on the right to the variable on the left.
`--`	Decrement the variable by 1.

Code examples for assignment operators

X++

```
// An example of assignment operators and their output.
static void Example1()
{
    int i = 1;
    // Using the = operator. i is assigned the value of i, plus 1. i = 2.
    i = i + 1;
    info(strFmt("Example 1: The result is "), i); // The result is 2.
}

static void Example2()
{
    int i = 1;
    // Using the += operator. i is assigned the value of i, plus 1.
    // i = 2 (i = i + 1).
    i += 1;
    info(strFmt("Example 2: The result is "), i); // The result is 2.
```

Operator precedence

Article • 06/13/2024

This article describes operator precedence.

The order in which a compound expression is evaluated is important. If you do not explicitly tell the compiler the order that you want operations to be performed in, the order is based on operator precedence. You can use parentheses `( )` to explicitly tell the X++ compiler how you want an expression to be evaluated.

Consider the expression `x + y / 100`, which gives a different result depending on whether the addition or the division is performed first. Because the division operator has a higher precedence than the addition operator, the compiler evaluates `y/100` first. So, `x + y / 100` is equivalent to `x + (y / 100)`. If you add parentheses to make the expression `(x + y)/ 100`, then `x + y` is evaluated first.

To make your code easy to read and maintain, be explicit, and indicate with parentheses which operators should be evaluated first.

Order of operator precedence

The operators in the following table are listed in precedence order. The higher in the table an operator appears, the higher precedence it has. Operators with higher precedence are evaluated before operators with a lower precedence. The operator precedence of X++ is not the same as other languages, for example C# and Java.

Expand table

Operators in precedence order	Syntax
unary operators	`- ~ !`
multiplicative, shift, bitwise AND, bitwise exclusive OR	`* / % DIV << >> & ^`
additive, bitwise inclusive OR	`+ -`
relational, equality	`< <= == != > >= like as is`
logical operators (AND, OR)	`&& ||`
conditional	`? :`

Classes and methods

Article • 08/12/2022

This article describes how to create and use classes in X++.

A *class* is a software construct that defines the data and methods of the instances that are later constructed from that class. The *class* is an abstraction of an *object* in the problem domain. The instances that are constructed from the *class* are known as *instances* or *objects*. This article uses the term *instance*. The data represents the state of the object, whereas the methods represent the behavior of the object.

Variables contain the data for the class, and are called *fields*. Every instance that is constructed from the class declaration has its own copy of the variables. These variables are known as *instance variables* or *instance fields*. This article will use the term *field* in most cases.

Methods define the behavior of a class. They are the sequences of statements that operate on the data (instance fields). By default, methods are declared to operate on the instance fields of the class. These methods are known as *instance methods* or *object methods*.

You can declare *static methods* and *static fields*, that do not have access to *instance fields*. These are described in X++ static classes.

Declare a class

You must use the **Add new item** dialog in Visual Studio to add a class to your project.

1. In Server Explorer, right-click the project, and then click **Add**.
2. In the **New Item** dialog box, select **Installed** > **Dynamics 365 Items** > **Code** in the left navigation. Then select **Class**, and then enter a name for the class.
3. Click **Add**.

All classes are public. If you remove the **public** modifier, the system still treats the class as public. You can specify other modifiers on the class declaration, such as **final** and **extends**.

Fields

Instance fields are **protected** by default. This means that they can only be accessed in the same class or a derived class. You can modify an instance field declaration by using

X++ inheritance

Article • 08/12/2022

This article describes inheritance in X++, including how to create a subclass and override a method.

Creating a subclass

Subclasses are classes that extend or inherit from other classes. A class can extend only one other class. Multiple inheritance isn't supported. If you extend a class, the subclass inherits all the methods and variables in the parent class (the *superclass*). Subclasses let you reuse existing code for a more specific purpose. Therefore, they help save you time during design, development, and testing. To customize the behavior of a superclass, override the methods in a subclass. A superclass is often known as a *base class*, and a subclass is often known as a *derived class*.

Subclass example

The following example first creates a class that is named **Point**. It then extends the **Point** class to create a new class that is named **ThreePoint**.

X++

```
class Point
{
    // Instance fields.
    real x;
    real y;

    // Constructor to initialize fields x and y.
    void new(real _x, real _y)
    {
        x = _x;
        y = _y;
    }
}

class ThreePoint extends Point
{
    // Additional instance fields z. Fields x and y are inherited.
    real z;

    // Constructor is overridden to initialize z.
    void new(real _x, real _y, real _z)
    {
        // Initialize the fields.
```

X++ static class members

Article • 08/12/2022

This article describes static class members in X++. In general, static methods are intended for these cases:

- The method has no reason to access the member variables that are declared in the class.
- The method has no reason to call any instance (non-static) methods of the class.

You declare static class members by using the **static** keyword. The **static** keyword instructs the system to create only one instance of the method, regardless of the number of instances of the class there are. This one instance is used throughout your session.

Static methods

This section describes a scenario where a software key type is used to help prevent piracy. Each instance of a software key can have its own unique value. Because all software keys must conform to the rules of software key design, the logic that tests for software key conformance is the same for all software keys. Therefore, the method that contains the conformance validation logic should be static.

Here is an example of a method that is declared by using the **static** keyword.

X++

```
public class SoftwareKey
{
    static public boolean validateSoftwareKey(str _softwareKeyString)
    {
        // Your code here.
        return false;
    }
}
```

In the following example, you don't have to construct an instance of the **SoftwareKey** class before you call a static method on the class. When you want to call the static **validateSoftwareKey** method, the syntax starts with the name of the class that contains the method. A pair of colons (::) is used to connect the class name to the static method name.

X++

X++ interfaces

Article • 08/12/2022

An *interface* specifies a set of public instance methods. An interface defines and enforces similarities between unrelated classes without having to derive one class from the other.

All interfaces are public, even if you don't explicitly add the **public** keyword in front of the **interface** keyword in interface declaration. The methods on an interface are also public. Explicit inclusion of the keyword **public** is optional.

To create an interface, follow these steps.

1. In Server Explorer, right-click the project, and then click **Add**.
2. In the **New Item** dialog box, select **Interface**, and then enter a name for the interface.
3. Click **Add**.

When you add the **implements** keyword on a class declaration, the class must declare and define the methods that are specified by the interface. A class declaration can implement multiple interfaces. List the interfaces after the single occurrence of the **implements** keyword, and separate the interface names by using commas.

All interface methods that a class implements must be explicitly declared as **public**. A class that implements an interface must also be declared as **public**. An interface can extend another interface by using the **extends** keyword, however, an interface can't extend more than one interface.

It is customary to preface the name of an interface with `I`.

Interface example

In the following code example, the **Automobile** class implements the **IDrivable** interface. The **is** keyword tests whether a class implements an interface.

X++

```
interface IDrivable
{
    int getSpeed()
    {
    }

    void setSpeed(int newSpeed)
```

X++ class library

Article • 08/12/2022

This article describes the library of classes in X++.

There are two kinds of classes: *application classes* and *system classes*.

- **Application classes –** These classes are implemented in X++. They are available in the **Code** > **Classes** node in Application Explorer.
- **System classes –** These classes are sometimes known as *kernel classes*. They are listed under the **System Documentation** > **Classes** node in Application Explorer. The source code for these classes isn't available. For a list of the system classes, see API, class, and table reference.

Typical structure of an application class

The following code block types are standard for application classes:

- **class and variable declarations**: The class declaration contains modifiers, such as **public**, **private**, and **extends**.
- **variable declarations**: These are the field members for objects that are constructed from the class. When you type the keyword **this** on a class instance variable, IntelliSense can show a list of the members.
- **new** method: This method creates an instance of the class. The constructor can be called only by using the **new** keyword. Derived classes can call the **new** method of their constructor by calling the **super** method reference. For more information, see X++ inheritance.
- **finalize** method: This method finalizes an instance of the class. This method is the destructor method. However, it's a destructor by convention only. The system doesn't automatically call the **finalize** method during garbage collection.

Additional methods for a class have the following types:

- Instance methods
- Static methods
- Main methods

Methods can be created on many kinds of items. Here are some examples:

- Classes
- Maps
- Views

Events and delegates

Article • 08/12/2022

This article describes event terminology and keywords in X++.

You can use the event design pattern to make your code more modular and reusable. The term *event* is a metaphor that explains how delegates are used. When something important occurs during a program run, other modules might have to process the occurrence. These important occurrences are known as *events*. When an event occurs, the program tells its notifier for the event that the notifier must send notifications about the event. A notification must be sent to all the event handlers that are subscribers of the notifier. When the program tells its notifier to send the notifications, we call that process *raising* an event.

A delegate can be defined in a table, form, or query, and not just in a class.

The following table shows the terms that are used to describe the event metaphor.

Expand table

Term	Description
Event	An important occurrence in a program module where additional modules must process the occurrence.
Notifier	The program element that sends information about the event to all the event handlers that are subscribed to the notifier.
Subscriber	The program functions or methods that are subscribed to an event notifier.
Event handler	The methods that subscribe to an event notifier. Only the appropriate kind of methods can be event handlers.

Keywords that are used for programming that uses delegates

The following table shows the keywords that describe the use of delegates.

Expand table

X++ data selection and manipulation overview

Article • 09/28/2022

You can use SQL statements, either interactively or in source code, to retrieve and modify data that is stored in the database. You can use the **select** statement and API methods for these tasks:

- Select data: Select the data to view or modify.
  - **select statement –** Fetch records.
- Insert data: Add one or more new records to a table.
  - **insert and doInsert methods** – Insert one record at a time.
  - **insert_recordset, RecordInsertList.insertDatabase, and RecordSortedList.insertDatabase methods** – Insert multiple records at the same time.
- Update data: Modify the data in existing table records.
  - **update and doUpdate methods –** Update one record at a time.
  - **update_recordset statement –** Update multiple records at the same time.
- Delete data: Remove existing records from a table.
  - **delete and doDelete methods –** Delete one record at a time.
  - **delete_from statement –** Delete multiple records at the same time.

Here are some other statements that you will use in data access:

- while select statement
- select expression
- next statement

Transactional integrity helps prevent data corruption and improve scalability.

The Conversion of operations from set-based to record-by-record article provides information about how you can use the record set–based statements and methods more efficiently.

You can also use the SysDa classes to retrieve and modify data. The extensible SysDa API provides almost all the data access possibilities that are available in X++.

The **executeQueryWithParameters** API can help mitigate a SQL injection attack.

Select statement

Article • 02/06/2023

The **select** statement fetches or manipulates data from the database.

- All **select** statements use a table variable to fetch records. This variable must be declared before a **select** statement can be run.
- The **select** statement fetches only one record, or field. To fetch or traverse multiple records, you can use the **next** statement or the **while select** statement.
  - The **next** statement fetches the next record in the table. If no **select** statement precedes the **next** statement, an error occurs. If you use a **next** statement, don't use the **firstOnly** find option.
  - It's more appropriate to use a **while select** statement to traverse multiple records.
- The results of a **select** statement are returned in a table buffer variable.
- If you use a field list in the **select** statement, only those fields are available in the table variable.

Select example

The following example fetches all the columns in the first row of the CustTable table and prints the value in the **AccountNum** column of that row.

```
X++
```

```
CustTable custTable;
select * from custTable;
info("AccountNum: " + custTable.AccountNum);
```

For more examples of data selection, see Select data.

Insert example

The following example inserts a new record into the CustTable table. The **AccountNum** column of the new record is set to **2000**, and the **CustGroup** column is set to **1**. Other fields in the record will be blank.

```
X++
```

Select data

Article • 08/12/2022

The **select** statement fetches or manipulates data from the database.

- All **select** statements use a table variable to fetch records. This variable must be declared before a **select** statement can be run.
- The **select** statement fetches only one record, or field. To fetch or traverse multiple records, you can use the **next** statement or the **while select** statement.
  - The **next** statement fetches the next record in the table. If no **select** statement precedes the **next** statement, an error occurs. If you use a **next** statement, don't use the **firstOnly** find option.
  - It's more appropriate to use a **while select** statement to traverse multiple records.
- The results of a **select** statement are returned in a table buffer variable.
- If you use a field list in the **select** statement, only those fields are available in the table variable.

The following example fetches all the columns in the first row of the CustTable table and prints the value in the **AccountNum** column of that row.

X++

```
CustTable custTable;
select firstonly custTable; //this is a short notation for 'select firstonly
* from custTable;'
info("AccountNum: " + custTable.AccountNum);
```

The following example prints the value in the **AccountNum** column of each row in the CustTable table.

X++

```
CustTable custTable;
while select custTable
{
    info("AccountNum: " + custTable.AccountNum);
}
```

The following example prints the value in the **AccountNum** column of the first two rows that are returned by the **select** statement.

Insert data

Article • 08/12/2022

You can use SQL statements, either interactively or in source code, to insert one or more rows into tables that are stored in the database.

- **insert method** – Insert one row at a time.
- **doInsert method** – Insert one row at a time.
- **insert_recordset statement** – Copy multiple records directly from one or more tables into another table in one database trip.
- **RecordInsertList.insertDatabase** – Insert multiple rows at the same time in one database trip. Use this construct when you don't have to sort the data.
- **RecordSortedList.insertDatabase** – Insert multiple rows at the same time in one database trip. Use this construct when you want a subset of data from a specific table, and you want that data to be sorted in an order that doesn't currently exist as an index.

RecordSortedList, **RecordInsertList**, and **insert_recordset** let you insert multiple records. By using these methods, you reduce communication between the application and the database. Therefore, you help increase performance. In some situations, record set–based operations can fall back to record-by-record operations. For more information, see Conversion of operations from set-based to record-by-record.

insert method

The **insert** method inserts one record at a time. It generates values for the **RecId** field and system fields, and then inserts the contents of the buffer (that is, the column values) into the database.

- Don't use a **select** statement on the table variable before you call the **insert** method.
- The **insert** method doesn't handle all the key field requirements and table dependencies. You must write code to handle them.

Here is how the **insert** method works:

- Only the specified columns of the rows that have been selected by the query are inserted into the named table.
- The columns of the table that is copied from and the columns of the table that is copied to must be type-compatible.

Update data

Article • 08/12/2022

You can use SQL statements, either interactively or in source code, to update one or more rows in a table that is stored in the database.

- **update method** – Update the current record with the contents of the buffer. Also update the appropriate system fields.
- **doUpdate method** – Update one row at a time.
- **update_recordset statement** – Update multiple records in one database trip. By using the **update_recordset** statement, you reduce communication between the application and the database. Therefore, you help increase performance. In some situations, record set–based operations can fall back to record-by-record operations. For more information, see Conversion of operations from set-based to record-by-record.

update method

The **update** method updates the current record with the contents of the buffer. It also updates the appropriate system fields. The optional **where** clause specifies a condition that the **update** method tests as it processes each row of the table. Only those rows that test **true** against the condition are updated with the new values.

The following example selects the CustTable table for update. Only records where the value of the **AccountNum** field equals **4000** are updated. Because there is no call to **next**, and this example doesn't use a **select while** statement, only one record is updated. The value of the **CreditMax** field is changed to **5000**.

```
X++
```

```
CustTable custTable;
ttsBegin;
    select forUpdate custTable
        where custTable.AccountNum == '4000';
    custTable.CreditMax = 5000;
    custTable.update();
ttsCommit;
```

doUpdate method

Delete data

Article • 08/12/2022

You can use SQL statements, either interactively or in source code, to delete one or more rows from tables that are stored in the database.

- **delete method** – Delete one row at a time.
- **doDelete method** – Delete one row at a time.
- **delete_from statement** – Delete multiple rows at the same time. By using the **delete_from** statement, you reduce communication between the application and the database. Therefore, you help increase performance. In some situations, this set-based operation can fall back to a record-by-record operation. For more information, see Conversion of operations from set-based to record-by-record.

delete method

The **delete** method deletes the current record from the database. To use this method, use a **where** clause to specify the rows to delete. One record at a time is then removed from the specified table.

The **delete** method can be overridden. For example, you might want to add extra validation before records are deleted. If you override the **delete** method, you can run the original (base) version of the **delete** method by calling the **doDelete** method. Therefore, a call to the **doDelete** method is equivalent to a call to **super()** in the **delete** method.

In the following example, all records in the NameValuePair table that satisfy the **where** clause (that is, all records where the value of the **Name** field equals **Name1**) are deleted from the database. One record is deleted at a time.

X++

```
ttsBegin;
    NameValuePair nameValuePair;

    while select forUpdate nameValuePair
        where nameValuePair.Name == 'Name1'
    {
        nameValuePair.delete();
    }
ttsCommit;
```

While select statement

Article • 08/12/2022

A **while select** statement is used to handle data. It's the most widely used form of the **select** statement. The **while select** statement loops over many records that meet specific criteria, and can run a statement on each record. The syntax of a **while select** statement resembles the syntax of a **select** statement, but the statement is preceded by **while select** instead of **select**.

- Typically, when you use the **while select** statement for data manipulation, you use it in a transaction to ensure data integrity.
- The results of the **while select** statement are returned in a table buffer variable.
- If you use a field list in the **select** statement, only those fields are available in the table variable.
- If you use aggregate functions, such as **sum** or **count**, the results are returned in the fields that you perform the **sum** or **count** over. You can count, average, or sum only integer and real fields.
- The **select** statement itself is run only one time, immediately before the first iteration of the statements in the loop.
- Any Boolean expressions that are added to the **while select** statement (for example, **iCounter** < **1**) are tested only one time. This behavior differs from the behavior of the **while** statement in languages such as C++ and C#. For example, the following loop can have more than one iteration.

```
X++
```

```
int iCounter = 0;
BankAccountTable xrecBAT;

while select * from xrecBAT
    where iCounter < 1
{
    iCounter++;
    info(strFmt("%1 , %2", iCounter, xrecBAT.AccountID));
}
```

The following example prints the **AccountNum** and **SalesGroup** values of every customer in the CustTable table whose account number is within a specified range.

```
X++
```

Write select statements as expressions

Article • 08/12/2022

You can use a **select** statement as an expression. This type of **select** statement is known as an *expression **select** statement*.

- You can't use a table buffer variable in an expression **select** statement.
- The name of the table must be used in the **from** clause.
- The **join** keyword isn't supported.
- The table name can't be used to qualify a field name in the **order by** clause.
- In a **where** clause, the table name must be used as a qualifier of the field.
- You can mention only one table in an expression **select** statement. Therefore, subselects aren't supported as a workaround for the unsupported **join** keyword.
- The only column that can be filled with data is the column that is named before the **from** clause in the **select** clause.
- After the closing parenthesis, the name of a column is used to reference the data value.

The following expression returns the value in the **AccountNum** column of the first row in the CustTable table (if a row exists).

```
X++
```

```
str accountNum = (select AccountNum from CustTable order by AccountNum
desc).AccountNum;
info('Max AccountNum: ' + accountNum);
```

Here is a simpler way to achieve the same result as the previous example.

```
X++
```

```
str accountNum = (select maxof(AccountNum) from CustTable).AccountNum;
info('Max AccountNum: ' + accountNum);
```

The following example returns the maximum **RecId** value of customers that aren't blocked. Here, the **maxof** aggregate function is used, and the **RecId** field is mentioned in the function. The field that is mentioned in the aggregate function must match the field name that is used to reference the data value after the closing parenthesis. Otherwise, empty data is returned.

```
X++
```

X++ transactional integrity

Article • 08/12/2022

This article describes transactional integrity in the X++ language.

If you don't take steps to ensure the integrity of transactions, data corruption can occur. At the very least, you might experience poor scalability with respect to concurrent users on the system. Two internal checking features help ensure the integrity of transactions: the **forUpdate** check and the **ttsLevel** check.

- A **forUpdate** check helps ensure that a record can be updated or deleted only if it has first been selected for update. You can select a record for update by using either the **forUpdate** keyword in the **select** statement or the **selectForUpdate** method on the table.
- A **ttsLevel** check helps ensure that a record can be updated or deleted only in the same transaction scope where it was selected for update.

The following statements are used to help ensure integrity:

- **ttsBegin –** This statement marks the beginning of a transaction. It helps ensure data integrity and also helps ensure that all updates that are done until the transaction ends (through **ttsCommit** or **ttsAbort**) are consistent.
- **ttsCommit –** This statement marks the successful end of a transaction. It ends and commits a transaction. The finance and operations app ensures that a transaction that has been committed will be performed according to intentions.
- **ttsAbort –** This statement lets you explicitly discard all changes in the current transaction. In this case, the database is rolled back to the original state, where nothing has been changed. Typically, you use this statement if you've detected that the user wants to break the current job. The **ttsAbort** statement helps ensure that the database is consistent.

Usually, it's a better idea to use exception handling instead of **ttsAbort**. The **throw** statement automatically aborts the current transaction. As the following example shows, statements between **ttsBegin** and **ttsCommit** can include one or more transaction blocks. In these cases, nothing is committed until a successful exit from the final **ttsCommit** statement occurs.

X++

```
ttsBegin;
    // Some statements.
    ttsBegin;
        // More statements.
```

Conversion of operations from set-based to record-by-record

Article • 08/12/2022

You can use the following statements and methods to help improve performance by reducing communication between the application and the database:

- delete_from
- update_recordset
- insert_recordset
- RecordSortedList.insertDatabase
- RecordInsertList.insertDatabase

In some situations, these record set–based operations can be converted to slower record-by-record operations. The following table identifies these situations.

Expand table

Situation	delete_from	update_recordset	insert_recordset	RecordSortedList, RecordInsertList	Used to override
Non-SQL tables	Yes	Yes	Yes	Yes	Not applicable
Delete actions	Yes	No	No	No	**skipDeleteActions**
The database log is enabled.	Yes	Yes	Yes	No	**skipDatabaseLog**
Overridden method	Yes	Yes	Yes	Yes	**skipDataMethods**
Alerts are set up for the table.	Yes	Yes	Yes	No	**skipEvents**
The **ValidTimeStateFieldType** property on a table is set to a value other than **None**.	Yes	Yes	Yes	Yes	Not applicable

You can use the **skip*** settings that are shown in the "Used to override" column to explicitly skip or ignore one or more factors that adversely affect performance. If one of the previously mentioned SQL operations is downgraded to a record-by-record operation, all the **skip*** settings are ignored. In the following example code, the **insert** method on the myTable table is run, even though it's explicitly stated that this method should be skipped if a container or memo field is defined for myTable.

X++

```
public void tutorialRecordInsertList()
{
    MyTable myTable;
    RecordInsertList insertList = new RecordInsertList(
        myTable.TableId,
```

Access data by using the SysDa classes

Article • 08/17/2023

This article explains how to create extensible queries by using the SysDa application programming interface (API).

The extensible SysDa API provides almost all the data access possibilities that are available in X++. In fact, the APIs are wrappers around the code that the X++ compiler would generate. Therefore, use of the SysDa classes carries no overhead, unlike use of the **QueryRun** object, for example. Additionally, the check that the X++ compiler does on data access statements is your responsibility. For example, you create a **where** clause that compares a globally unique identifier (GUID) to an integer. The X++ compiler would diagnose this clause as an error.

The SysDa APIs include an extensive set of APIs for creating custom queries. However, there is a smaller set of types that drives the primary query activities:

- Select: **SysDaQueryObject**, **SysDaSearchObject**, and **SysDaSearchStatement**
- Update: **SysDaUpdateObject** and **SysDaUpdateStatement**
- Insert: **SysDaInsertObject** and **SysDaInsertStatement**
- Delete: **SysDaQueryObject**, **SysDaDeleteObject**, and **SysDaDeleteStatement**

The following sections provide examples of each type of query and the customizations that it supports. The examples use a table that is named TestTable. This table has two fields: a string field that is named **stringField** and an integer field that is named **intField**.

Select query

To run a **select** query, follow these steps.

1. Create and configure a **SysDaQueryObject** object that specifies the table instance that will contain the designated records.
2. Create a **SysDaSearchObject** object, and pass the **SysDaQueryObject** object to the constructor.
3. Iterate over the results of the query by passing the **SysDaSearchObject** object to the **SysDaSearchStatement.findNext()** method.

The following example finds all rows in TestTable where **intField** <= **5**.

```
X++
```

Mitigate a SQL injection attack

Article • 06/13/2024

An SQL injection attack occurs when malicious data values are passed to Microsoft SQL Server in a query string. Those values can cause lots of damage in a database. SQL injection can occur if you aren't careful about how you use a query to pass data that comes from an uncontrolled source, such as user input, to SQL Server. SQL injection isn't usually an issue in finance and operations apps, because the built-in data access statements in X++ prevent it. However, if you use Direct-SQL, SQL injection can occur when raw SQL code is passed to the server.

A new API will help mitigate these attacks. The API is available starting with platform updates for version 10.0.17 of finance and operations apps (April 2021).

The issue

Consider a scenario where a developer writes the following code to look up the first name of customers, based on their last name.

X++

```
public str GetFirstName(str name)
{
    str sqlStatementText = "SELECT TOP(1) firstName FROM Customer WHERE
customer.Name = '" + name + "'";

    // Create a connection to the SQL Server
    var connection = new Connection();

    // Create a statement and submit the sql statement to the server:
    Statement statement = connection.createStatement();
    var results= statement.executeQuery(sqlStatementText);

    // Get the first record:
    results.next();

    // Harvest the results.
    return results.getString(1);

   // Close statement
    statement.close();

    return result;
}
```

Macros in X++

Article • 08/12/2022

This article describes how to create and use macros in X++.

Precompiler directives are processed before the code is compiled. The directives declare and handle macros and their values. The directives are removed by the precompiler so that the X++ compiler never encounters them. The X++ compiler only sees the sequence of characters written into the X++ code by the directives.

> ⚠ **Warning**
>
> Use of macros is not recommended. Macros are supported for backwards compatibility only.
>
> Instead of macros, use language constructs like these:
>
> - Constants
> - Sysda for queries.

#define and #if directives

All precompiler directives and symbols begin with the `#` character. A macro can be defined at any point in the code. The variable can have a value that is a sequence of characters, but it is not required to have a value. The `#define` directive tells the precompiler to create the macro variable, including an optional value. The `#if` directive tests whether the variable is defined, and optionally, whether it has a specific value. The X++ precompiler directives, the macro names that they define, and the `#if` directive value tests are all case-insensitive. However, it is a best practice to begin macro names with an uppercase letter.

#undef directive

You can use the `#undef` directive to remove a macro definition that exists from a previous `#define`. After a macro name has been created by `#define` and then removed by `#undef`, the macro can be created again by another `#define`. `#undef` has no effect on macros that are created by the `#localmacro` directive.

X++ attribute classes

Article • 06/13/2024

This article describes the use of attributes in X++.

An attribute is a non-abstract class that extends (inherits from) the **SysAttribute** class. Attributes represent or store metadata about types and methods. An attribute can be attached to a class, a class field, a class method, an interface, or a table.

Attributes are applied to the handlers of delegates and methods, to map the handlers to those targets.

Creating an attribute class

An attribute class can extend the **SysAttribute** class directly, or it can extend any descendant of the **SysAttribute** class. The **SysAttribute** class cannot be used as an attribute because it is declared **abstract**. The following example shows the declaration and design of an ordinary attribute class that you could create.

X++

```
public class PracticeAttribute extends SysAttribute
{
    // Fields in the classDeclaration.
    StartEnd startEndEnum;
    str reason;
    // Constructor.
    public void new(StartEnd _startEndEnum, str _reason)
    {
        startEndEnum  = _startEndEnum;
        reason = _reason;
    }
    // Other methods can go here.
}
```

Decorating a class with an attribute

The following example shows a class and a method that are decorated with the **PracticeAttribute** given in the previous example. If the constructor of the attribute takes no parameters, the parentheses for the parameters are optional. The attribute decoration could be `[AnotherAttribute]` without parentheses.

X++

X++ and C# comparison

Article • 03/09/2023

This article compares X++ and C# syntax and programming.

X++, C# Comparison: Hello World

This section compares the simplest X++ program to its counterpart in C#.

X++ to C# Comparisons

The following sections describe some basic similarities and differences between X++ and C#.

Similarities

The following X++ features are the same for C#:

- Single line (`//`) and multi-line (`/* */`) comments.
- `==` (equal) operator for determining whether two values are equal.
- `!=` (not equal to) operator for determining whether two values aren't equivalent.
- `+` (plus sign) operator for string concatenation.

Differences

The following table lists X++ features that are different in C#.

Expand table

Feature	X++	C#	Comments
`if` and `else` conditional statements	The `if` statement accepts any type of expression that it can automatically convert to a Boolean. Common examples include an `int` for which 0 means false, or an object for which null means false.	The `if` statement requires a Boolean expression.	The syntax structure regarding curly braces and parentheses is exactly the same between X++ and C#.

X++ syntax

Article • 10/03/2024

This article contains the syntax reference for X++.

X++ Keywords

The X++ keywords shown in the following table are reserved. These keywords can't be used for any other purpose.

Expand table

Reserved word	Description	Related information
!	Not.	Relational Operators
!=	Inequality operator (not equal to).	Relational Operators
#	Prefix on macro names.	How to: Use #define and #if to Test a Macro
&	Binary AND.	Arithmetic Operators
&&	Logical AND.	Relational Operators
(	Function call operator, which indicates the beginning of the function call.	
)	Function call operator, which indicates the end of the function call.	
*	Multiply. The asterisk (*) is also used in X++ SQL. One use is to signify all fields from the tables on a `select` statement. Another use is as a wildcard with the `like` operator, to signify 0 to many characters of any kind. The `like` operator also uses the ? character.	Arithmetic Operators
^	Binary XOR.	Arithmetic Operators
|	Binary OR.	Arithmetic Operators
||	Logical OR.	Relational Operators
~	Not.	Arithmetic Operators

API, class, and table resources

Article • 09/19/2022

This article describes where to find API documentation in Visual Studio and Microsoft Learn.

Application classes and tables

Application class and table documentation is in Visual Studio

You can find documentation for the Application classes in Microsoft Visual Studio. Search for the class name in Application Explorer and then display the code. You can find additional metadata about the class in the **Properties** window. You can download a list of all the tables in the Technical Reference Reports. For more information, see Find information about standard data entities.

Programming with application tables and classes

Application tables are similar to application classes, but with the following differences from classes:

- Tables are persistent.
- Table fields are always public.
- A table almost always corresponds to a real object.
- The definition of a table must sometimes be erased if you later want another table to extend it.

Design pattern of private new in application classes

All application classes are under Application Explorer > Classes. Every application class has the constructor method named `new`, even if the class has no **new** node in the Application Explorer. If the class has no explicit **new** node, the implicit **new** method is public. A design pattern that is sometimes used in the application classes is to declare the explicit **new** constructor method as **private**. Then a **public static** method is added to call the **new** method. The static method can restrict or control the call the **new** method based on various conditions, if necessary.

X++ compile-time functions

Article • 08/12/2022

This article lists the compile-time functions and describes their syntax, parameters, and return values.

Overview

Compile-time functions are executed early during compilation of X++ code. They should be used wherever possible in X++ code to make the code resilient to changes to the metadata stored in the Application Explorer. Compile-time functions have their input value verified by the compiler. If the input value is not found to match any existing object in the Application Explorer, the compiler issues an error. The inputs to these functions must be literals, because the compiler cannot determine the value that a variable contains at run time. A compile-time function is a metadata assertion function. It takes arguments that represents an entity in the Application Explorer and validates the arguments at compile time. It has no effect at run time. Attributes are classes that inherit from the **SysAttribute** class. To support the validation of form, report, query, and menu metadata, use the **AutoDeclaration** property on controls. Most of these functions retrieve metadata about items that are in the Application Explorer. Some common compile time functions are as follows:

- `classNum` – Retrieves the ID of a class.
- `classStr` – During compile time, verifies that a class of that name exists. This approach is better than discovering the error later during run time.
- `evalBuf` – Evaluates the input string of X++ code, and then returns the results as a string.
- `literalStr` – retrieves a label ID when given the string representation of a label, such as the string `"@SYS12345"`. For example, `myLabel.exists(literalStr("@SYS12345"));`.

> **Note**
>
> X++ compile time functions cannot be called from a .NET program.

Functions

X++ runtime function resources

Article • 08/12/2022

This article describes the X++ run-time functions.

The X++ language provides nearly 200 system functions that aren't part of any class and are executed at run time. Run-time functions are used for data type conversions, mathematical operations, and so on. Here are some common run-time functions:

- **str2Int –** Creates an int value from a str value.
- **abs –** Creates a positive real value from a real value that is either positive or negative.
- **conFind –** Retrieves the location of an element in a container.

Call run-time functions from .NET

The logic of the X++ run-time functions is also implemented in the following .NET assembly.

```
X++

Microsoft.Dynamics.AX.Xpp.Support.DLL
```

Inside this assembly, the X++ run-time functions are implemented as static methods of the following class.

```
X++

Microsoft.Dynamics.AX.Xpp.PredefinedFunctions
```

Categories and functions

The following table lists and describes only the categories of X++ functions. These categories are intended to help you understand the many functions. However, the categories don't represent any formal construct.

Expand table

X++ business runtime functions

Article • 08/12/2022

This article describes the business run-time functions.

These functions enter financial data and calculate formulas.

cTerm

Calculates the number of periods that are required for the current investment value to yield a target value.

Syntax

```
X++
```

```
real cTerm(real interest, real future_value, real current_value)
```

Parameters

Expand table

Parameter	Description
interest	The interest rate.
future_value	The target value.
current_value	The current investment value.

Return value

The number of periods that are required in order to reach *future_value*.

Remarks

The *current_value* and *future_value* parameters must have the same prefixed sign (plus or minus).

Example

X++ container runtime functions

Article • 08/12/2022

This article describes the container run-time functions.

These functions manipulate the contents of containers.

conDel

Removes the specified number of elements from a container.

Syntax

```
X++
```

```
container conDel(container container, int start, int number)
```

Parameters

Expand table

Parameter	Description
container	The container to remove elements from.
start	The one-based position at which to start removing elements.
number	The number of elements to delete.

Return value

A new container that doesn't include the removed elements.

Example

```
X++
```

```
static void conDelExample(Args _args)
{
    container c = ["Hello world", 1, 3.14];
        // Deletes the first two items from the container.
```

X++ conversion runtime functions

Article • 12/13/2024

This article describes the conversion run-time functions.

any2Date

Converts an **anytype** value to a **date** value.

```
X++
```

```
date any2Date(anytype object)
```

Parameters

Expand table

Parameter	Description
object	The value to convert to a date.

Return value

A **date** value.

Remarks

The *object* parameter can be of most data types, but useful output is obtained when it's of the **str** or **int** type. Inappropriate content generates a run-time error.

Example

```
X++
```

```
static void any2DateExample(Args _args)
{
    date myDate;
    str s;
    int i;
    s = "2010 6 17"; // A string object, of yyyy mm dd.
    myDate = any2Date(s);
```

X++ date runtime functions

Article • 08/12/2022

This article describes the date run-time functions.

dayName

Retrieves the name of the day of the week that is specified by a number.

X++

```
str dayName(int number)
```

Parameters

Expand table

Parameter	Description
number	The number of a day in a week.

Return value

The day of the week specified by the number parameter.

Remarks

The valid values for the number parameter are **1** through **7**. Monday is represented by **1**, Tuesday by **2**, and Sunday by **7**.

Example

X++

```
static void dayNameExample(Args _arg)
{
    str s;
    ;
    s = dayName(01);
    print "First day of the week's name is " + s;
```

X++ math runtime functions

Article • 08/12/2022

This article describes the math run-time functions.

These functions perform mathematical calculations.

abs

Retrieves the absolute value of a real number. Examples:

- **abs(-100.0)** returns the value **100.0**.
- **abs(30.56)** returns the value **30.56**.

Syntax

X++

```
real abs(real arg)
```

Parameters

Expand table

Parameter	Description
arg	The number to get the absolute value of.

Return value

The absolute value of *arg*.

Example

X++

```
static void absExample(Args _args)
{
    real r1;
    real r2;
    ;
```

X++ reflection runtime functions

Article • 08/12/2022

This article describes the reflection run-time functions.

classIdGet

Retrieves the numeric identifier (the class ID) of the class that the object that is initialized belongs to.

X++

```
int classIdGet(class object)
```

Parameters

Expand table

Parameter	Description
object	The object to get the class ID for.

Return value

The class ID of the specified object.

Example

X++

```
static void classIdGetExample(Args _args)
{
    int i;
    WorkTimeCheck w;

    i = classIdGet(w);
    print "Class ID for object is " + int2Str(i);
}
```

dimOf

X++ session runtime functions

Article • 08/12/2022

This article describes the session run-time functions.

curExt

Retrieves the extension that is used for the current company.

```
X++

str curExt()
```

Return value

The extension for the current company.

Example

```
X++

static void curExtExample(Args _arg)
{
    str s;
    // Sets s to the extension of the current company.
    s = curExt();
    print "Current extension is " + s;
}
```

curUserId

Retrieves the nonnumeric ID that represents the current user.

```
X++

str curUserId()
```

Return value

The nonnumeric ID that represents the current user.

X++ string runtime functions

Article • 12/13/2024

This article describes the string run-time functions.

match

Searches for a string or expression in another string.

```
X++

int match(str pattern, str text)
```

Parameters

Expand table

Parameter	Description
pattern	The string or expression to search for.
text	The string to search.

Return value

1 if the pattern is located in the string; otherwise, **0** (zero).

Remarks

The search is case-insensitive. The following special characters can be used to create the pattern for the *pattern* parameter.

- \: A backslash (\) nullifies, or escapes, the special treatment of special characters, so that a special character can be matched like a normal letter. A pair of backslashes is translated into one non-special backslash. Examples:
  - **match("ab$cd","ab$cd");** returns **0**.
  - **match("ab\$cd","ab$cd");** returns **0**. The backslash isn't escaped.
  - **match("ab\\$cd","ab$cd");** returns **1**. The backslash and dollar sign are escaped.
- **<** or **^**: A left angle bracket (<) or a circumflex (^) at the start of an expression is used to match the start of a line. Examples:
  - **match("<abc","abcdef");** returns **1**.
  - **match("<abc","defabc");** returns **0**.
  - **match("^abc","abcdef");** returns **1**.
  - **match("^abc","defabc");** returns **0**.

System tables

Article • 08/12/2022

This article contains the documentation available for the System classes.

> **Note**
>
> This article is not a complete list of the System table members. You can find a complete list of tables and their members in the Application Explorer.

Common

The Common table is the base class for all tables. It does not contain any data. It is primarily used in X++ code to refer to any table in a polymorphic way.

Methods

Expand table

Method	Description
aosValidateDelete	Validates on the server that the specified record can be deleted from a table.
aosValidateInsert	Validates on the server that the specified record can be inserted.
aosValidateRead	Validates on the server that the specified record can be read.
aosValidateUpdate	Validates on the server that the specified record can be updated.
buf2con	Packs the table buffers of an xRecord instance into an X++ container.
canSubmitToWorkflow	Indicates whether submission to workflow is possible.
caption	Gets and sets the caption property of a table.
checkInvalidFieldAccess	Gets and sets invalid field access.
checkRecord	Gets and sets the property that indicates whether to check mandatory fields.
checkRestrictedDeleteActions	Gets and sets the property that indicates whether a record can be deleted.
clear	Removes all rows from the table buffer.
company	Gets and sets the property that indicates a legal entity for the record.
con2buf	Unpacks a container into the table buffers.
concurrencyModel	Gets and sets the default concurrency model to use to update records.
context	Gets and sets the context property.
data	Retrieves a row from the table.
dataSource	Retrieves the data source of the table.
dbOpInTransaction	Makes sure that database operations are correctly closed if they fail.

Real async feature enhancements

Article • 11/08/2024

> **Important**
>
> Some or all of the functionality noted in this article is available as part of a preview release. The content and the functionality are subject to change. For more information about preview releases, see Service update availability.

This article describes the real async feature enhancements to SysOperations. These enhancements enable operations to be run asynchronously without blocking the client as the regular SysOperations do. Therefore, users can initiate multiple operations simultaneously. In this way, the enhancements help improve overall performance.

The status of async operations can be viewed on the same page.

To use real async operations, you must extend the `SysOperationServiceController` class.

For more information about the SysOperation framework, see SysOperation Framework Overview.

Enable the real async feature

1. In the **Feature management** workspace, select **Check for updates**.
2. Enable **SysRealAsyncOperationsFeature**.
3. After you've enabled the feature in Feature management, go to **Client performance options**, and select the **Enable real async operations** option.

Uptake the real async feature

To use real async operations, you must first extend the `SysOperationServiceController` class, as you normally do to implement SysOperations. Then override the following methods to enable real async operations for your service operation class.

Method 1: canRunAsRealAsync

By default, the `canRunAsRealAsync` method returns *false*.

1. To turn ON/OFF the real async execution of your operation at runtime, introduce a feature for your operation and confirm the feature is enabled inside

Extensibility home page

Article • 08/12/2022

Dynamics 365 Finance, Supply Chain, and Commerce are extensively customized by partners, value added resellers (VARs), and even some customers. The ability to customize the product is a strength that has historically been supported through overlayering of the application code. The move to the cloud, together with more agile servicing and frequent updates, requires a less intrusive customization model, so that updates are less likely to affect custom solutions. This new model is called *extensibility* and has replaced customization through overlayering.

Extensibility is the only customization framework in Finance, Supply Chain, and Commerce. Overlayering isn't supported.

Introduction

These introductory topics contain general information about customization. This information includes information about when the transition occurs from customization through overlayering to a purely extension-based model. These topics also explain how to log extensibility requests to Microsoft, and provide answers to frequently asked questions (FAQ).

- Application extensibility plans
- Extensibility requests
- Extensibility FAQ

What's new

Read What's new or changed for extensibility for extensibility-related updates that have been made since July 2017.

Getting started

The topics in this section will help you start to build extensions. They will also help you migrate solutions that are currently based on overlayered code to extension-based solutions. This section includes hands-on labs that walk you through simple customizations.

- Migrate from overlayering to extensions
- Customize model elements through extension

Application extensibility roadmap

Article • 08/12/2022

Reducing implementation and upgrade effort is a major initiative for the development team. The benefits of this initiative are to enable you to quickly take advantage of new innovations from Microsoft and your partners, reduce the total cost of ownership, and improve quality. A major part of this initiative is to change the customization approach for the product. In Dynamics AX 2012, several extension capabilities were added to the product. For example, the ability to do event-based customization using methods pre- and post-events was introduced. Extension capabilities have continued to grow in the evolution to the new application.

Extension-based customizations have several advantages over the legacy approach of overlayering-based customizations, especially when it comes to reducing implementation and upgrade effort.

- Overlayering-based customizations require code upgrade, recompile time, and extensive testing. This limits the ability to seamlessly apply hot fixes. These costs can be an inhibitor for customers to upgrade to newer versions containing innovations from Microsoft and partners.
- Extension-based customizations also improve the development experience. Models containing overlayered customizations must be in the same package as the base objects. This results in longer compile cycles and larger package distributions. Extensions are also much easier to unit test in isolation from the base object.
- Reducing upgrade costs through extension-based customizations reduces the support matrix for partners as fewer release combinations will need to be supported.

For these reasons, we have gradually been sealing the product models, so they only support extension-based customizations. **AppPlatform** and **AppFoundation** were the first. These models were sealed for overlayering in Platform update 3 (November 2016). Binary updates are now provided to these models on a monthly basis, achieving our goals of reducing upgrade cost and delivering innovation to our customers at a faster cadence.

With Microsoft Dynamics 365 for Finance and Operations release 8.0, we have sealed all product models. Now only extension based customizations are supported.

The following illustration shows the roadmap we followed as we moved to extensions, away from overlayering.

Extensibility requests

Article • 04/19/2024

Finance and Operations applications exclusively use extensions to customize the product. We're aware that this change impacts our entire partner ecosystem. We recommend that you read the resources listed on the Extensibility home page. These resources answer many questions and prepare you for building solutions using extensions.

You'll discover that some customizations, which were possible with overlayering, can't be done through extensions. To enable the same business requirements without overlayering, we have added many extension capabilities and expect to add more going forward. For some customizations that were done with overlayering, you need to log requests, to make us aware of what you need.

What we are doing

In future releases, we'll be adding even more extensibility capabilities to enable independent software vendors (ISVs) and value-added resellers (VARs) to deliver complete business solutions. We'll prioritize these on a customer-by-customer basis with frequent releases.

How do I log extensibility requests?

If you discover a customization that you can't implement as an extension, you must log a request to Microsoft to ensure appropriate extension support is added to the product for your scenario.

Before logging the request, there are a few things to consider:

- Could the requirement be met with existing extensibility features? Building solutions with extensions requires different design and implementation patterns.
- How important is the requirement to the customer and/or business analyst?
- Will the implementation be upgrade friendly for the long term?

For more information about extensibility, see Extensibility home page.

Extensibility requests are logged using a specific project in LCS. Logged requests are collected under that same project. We recommend that you log related requests under the same LCS project as this helps maintain a holistic view on all requests for a specific

Extensibility FAQ

Article • 08/12/2022

Will source code be available after the hard seal?

Yes, source code will be available after the hard seal. It's required for effective implementation and debugging.

How do I contact Microsoft if I have an extensibility request?

There is a special extensibility request form on the Lifecycle Services (LCS) site.

Where can I ask questions about extensibility patterns?

You can gain access to the Operations Extensibility group in Yammer. Operations Extensibility is an active group that has a significant amount of partner engagement. You get access via the Connect site by signing an NDA.

Where can I find documentation about extensibility patterns?

Documentation about extensibility patterns is available on the Extensibility home page.

Where can I get information about extensibility training?

We will announce training sessions in multiple ways. AppSource partners might receive direct invitations for some sessions. We will also announce workshops in the Operations Extensibility Yammer group and other forums.

What is the goal of sealing the application?

Migrate from overlayering to extensions

Article • 08/12/2022

Introduction

When the application was first released, we strongly recommended that extensions be used instead of overlayering for customization. Overlayering-based customizations have been migrated from release to release through code migrations, and many customizations of application code are still based on the overlayering of code. For most partners, at least some of their solution is still based on overlaying, and some partners will have lots of overlayering across their solutions.

The amount of work that is required to change an implementation from overlayered code to extensions depends on the code itself. Some overlayered code can be changed relatively seamlessly. However, for some changes, you must rethink the customization to find an appropriate way to accomplish it through extension. Therefore, it can be a major undertaking to change complete solutions where multiple places have overlayered code. Such an undertaking requires an investment in the solution. The upside of this investment is a more seamless upgrade process, because customization is now based on application programming interfaces (APIs) through extensions. Additionally, a lengthy code upgrade process is no longer required as it was for overlayered code. More importantly, daily servicing of a running environment offers many benefits. The core application and extensions no longer have to be compiled together, and patching can be done by deploying precompiled assemblies. Therefore, customers can apply patches to their system in a relatively seamless manner, and the amount of downtime is minimized. However, there is work that must be done before this result can be achieved.

Although there are multiple ways to approach this task, we have gained experience through our close work with independent software vendors (ISVs) and value-added resellers (VARs) that have already started to migrate from overlayering to extensions. In this article, we share some of this experience.

First things first

The task ahead is substantial, and we want to make sure that our shared investment pays dividends. Keep the goal in mind as you work through your customizations. When customization is done correctly, your solution has these qualities:

- It has no intrusive customizations.
- It supports side-by-side deployment with other ISV solutions.

Customize model elements through extension

Article • 08/12/2022

In this tutorial, you'll become familiar with the Fleet Management Extension model. This model contains elements that extend the functionality of the Fleet Management application. You can customize model elements by creating *extensions*. Unlike the overlayering capabilities of Microsoft Dynamics AX 2012, extensions don't overlay the baseline model elements. Instead, extensions are compiled as a separate assembly that adds to or customizes the model and the associated business logic. You can extend metadata, for example, by adding a field to a table or adding a control to a form, and also extend or customize business logic by defining event handlers and plug-in classes. You can now author event handlers on several pre-defined events on tables, forms, form data sources, form controls, and others. Plug-ins are also a new extensibility concept that enables replacing or extending the business logic of the application.

Prerequisites

This tutorial requires you to access the environment using Remote Desktop, and that you are provisioned as an administrator on the instance.

Understanding the Fleet Management model

The Fleet Management application provides a rental car company a system for managing vehicles, customers, and vehicle reservations. The application is designed for use by the Fleet Clerk and Fleet Manager personas.

Fleet Clerk

The Clerk is the front desk employee who handles the face-to-face and over-the-phone interactions with customers. The Clerk is primarily concerned with entering customer information into the application, creating vehicle reservations for customers, upselling the reservation by offering vehicle accessories, and processing vehicle returns upon completion of a vehicle rental. The Clerk spends the vast majority of their time using the **Fleet Management Workspace** to prepare for interactions with customers by anticipating their needs and providing a pleasant and memorable experience, while interacting with the customer.

Customize through extension and overlayering

Article • 08/12/2022

This article discusses the two methods of customizing source code and metadata of model elements - overlayering and extensions and details supported extension capabilities.

Overlayering

You can customize source code and metadata of model elements that are shipped by Microsoft or third-party Microsoft partners. In order to customize metadata and source code of a model, the developer must create a new model that overlays the model they want to customize. For example, solution developers can provide code in the SLN layer, independent software vendors can use the ISV layer, and value-added resellers can use the VAR layer. Functionality defined in higher layers (VAR layer in this example) can override the functionality of lower layers. The overlaying model must belong to the same **Package** as the source model and belong to a layer that is higher than the source model. Overlayering is a powerful tool to perform advanced customizations of metadata and source code, but may increase the cost of upgrading a solution to a new version.

Extensions

You can customize an application by using *extensions*. An extension enables you to add functionality to existing model elements and source code. Extensions provide the following capabilities:

- Creating new model elements.
- Extending existing model elements.
- Extending source code using class extensions.
- Customizing business logic. Ways to customize business logic include:
  - Creating event handlers to respond to framework events, such as data events.
  - Creating event handlers to respond to event delegates that are defined by the application.
  - Creating new plug-ins.

To get started, review or complete this tutorial: Customize model elements through extension.

What's new or changed for extensibility

Article • 08/12/2022

This article provides links to extensibility updates.

- Extensibility changes version 10.0.3
- Extensibility changes version 10.0.2
- Extensibility changes version 10.0.1
- Extensibility changes version 10.0
- Extensibility changes version 8.1.3
- Extensibility changes version 8.1.2
- Extensibility changes version 8.1.1
- Extensibility changes version 8.1
- Extensibility changes version 8.0.4
- Extensibility changes version 8.0.3
- Extensibility changes version 8.0.2
- Extensibility changes version 8.0.1
- Extensibility changes in version 8.0
- Extensibility changes in version 7.3
- Extensibility changes July 2017

Feedback

Was this page helpful? Yes No

Provide product feedback

Extensibility changes in Dynamics 365 for Finance and Operations version 10.0.3

Article • 08/12/2022

This article lists the extensibility features that were implemented in Microsoft Dynamics 365 for Finance and Operations version 10.0.3. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

The following enumerations have been made extensible in this update:

- LedgerJournalWFApprovalModule
- RetailReceiptTransaction

SQL operations made extensible

The following SQL operations have been made extensible in this update:

- CustVendTrans support for an extensible map pattern

Metadata changes

The following metadata changes have been made in this update:

- CostSheetAmount.NoOfDecimalsIsExtensible
- SalesLinePercent.NoOfDecimalsIsExtensible

Refactored methods

The following methods have been refactored to support extensibility:

- BankReconciliationDataInitializer.initDocumentOpenTmp
- BankReconciliationDataInitializer.initStatementOpenTmp
- Class\BomCalcJob_All.processSingleTask
- Class\KanbanEventQuantityMap.newStandard
- Class\MCRFullTextSearchRefresh.run
- Class\PdsRebateAgreementValidate.validate
- Class\PurchRFQFormLetter.main
- Class\ReqTransPoMarkFirm.getPurchIdSingleThread
- Class\TAMVendRebateCorrectClaims.correctClaims
- Class\TAMVendRebateCorrectClaims.createClaimCorrection
- Class\TAMVendRebateCorrectClaims.rebateAmountPerUnit
- Class\WhsReleaseToWarehouseForm.buttonRelease_clicked

Extensibility changes in Dynamics 365 for Finance and Operations version 10.0.2

Article • 08/12/2022

This article lists the extensibility features that were implemented in Microsoft Dynamics 365 for Finance and Operations version 10.0.2. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

The following enumerations have been made extensible in this update:

- MarkupModuleType
- MCRCustPaymType
- PaymSchedBy

SQL operations made extensible

The following SQL operations have been made extensible in this update:

- JmgPayAdjustment.payAdjustLoop
- ProjPosting.ExtensionHash.New field
- WmsArrivalOverviewGeneration.buildPurch
- WmsArrivalOverviewGeneration.buildTransferOrder

Metadata changes

The following metadata changes have been made in this update:

- CostSheetPercent.NoOfDecimalsIsExtensible
- WHSCycleCountingWarehouseWorkLineEntity.IsPublic

Refactored methods

The following methods have been refactored to support extensibility:

- /Forms/ProjJournalTable/datasource/ProjJournalTable.initValue

Extensibility changes in Dynamics 365 for Finance and Operations version 10.0.1

Article • 08/12/2022

This article lists the extensibility features that were implemented in Microsoft Dynamics 365 for Finance and Operations version 10.0.1. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

The following enumerations have been made extensible in this update:

- ACOCostStatus_BR
- ACOCostType_BR
- ACOJournalType_BR
- BankModuloCheck_NO
- InventTransferOrderType_BR
- ProdJourType
- ProjTransStatus
- RetailLabelTypeBase
- RetailLedgerBank
- RetailTenderFunction
- SalesPurchTrntype_BR
- SMAGetPriceFrom
- SMASubscriptionIndexChange

SQL operations made extensible

The following SQL operations have been made extensible in this update:

- InventSumDelta.findInventSumDeltaInventSumFieldsAll.
- LedgerFiscalJournal was changed so that it uses QueryObject.
- TaxTransDP.

Metadata changes

The following metadata changes have been made in this update:

Extensibility changes in Dynamics 365 for Finance and Operations version 10.0

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations version 10.0. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

These enumerations have been made extensible in this update.

Expand table

Enumeration
AssetAccrualCalendar
AssetYear
BankReconciliationReportType
BudgetPlanColumnPeriodLength
BudgetPlanHCMReportGroupOption
CurrencyTypeBrief_RU
EInvoiceStatus_IT
EInvoiceStatus_IT
HRPAuthorityBasis
HuExchOutflowType
InventJournalTagStatus
InvoiceAssociationType
MCRClaimType
MCRMerchandisingEventCategory
PaymAttribute
PaymProposalReportedBy
PayrollCategory
projActualVsBudget
ProjListStateId

Extensibility changes in Dynamics 365 for Finance and Operations version 8.1.3

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations version 8.1.3. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

These enumerations have been made extensible in this update.

Expand table

Enumeration
AttributeDataType
BankDocumentBookType
BudgetPlanHCMReportGroupOption
CommitmentType
CommitmentType
CustVendNegInstStatus
CzAdvanceInvoiceStatus
DateTransactionDuedate
LedgerAllocationMethod
LedgerCovDocumentType
MCRFraudType
PercentHours
PerDayWeekMthQtYr
PrepaymentHandlingLayout_W
ProdStatusAll
TaxDirection

Extensibility changes in Dynamics 365 for Finance and Operations version 8.1.2

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations version 8.1.2. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

These enumerations have been made extensible in this update.

Expand table

Enumeration
DimensionHierarchyType
DirPartyType
DirPersonMaritalStatus
PrintPostCancel
INSAffiliate
LedgerJournalLinesDisplayOption
LedgerTransPerJournal
ProjDortValue
ProjPaymentStatus
RequisitionReleaseType
RetailPOSSeedDataType
SysDimension
TrvExpType
TSTimesheetEntryGridView
VendProspectiveVendorRegistrationWizardTab

Metadata changes

These metadata changes have been made in this update.

Expand table

Operation
DataEntities/LedgerJournalNameEntity/Fields/DeleteLinesAfterPosting.Allow Edit
DataEntities/LedgerJournalNameEntity/Fields/DeleteLinesAfterPosting.AllowEditOnCreate
Forms/AssetProposalDepreciation/Design/Tab/ParametersTabPage/ParametersGroup/SummarizedDepreciationControl.Value
Data manipulation method not raising event: PriceDiscAdmDeleteTradeAgreements.run
Data Types/Base Enums/WHSReverseWorkMode.Label
DataEntity smmProspectEntity is not public
DataEntityView/GeneralJournalAccountEntryEntity.PublicCollectionName, PublicEntityName and IsPublic
Enum/HcmPersonGender/EnumValue/NonSpecific.Label
LedgerJournalEngine.shouldOverwriteAmountWithSettledAmount

Extensibility changes in Dynamics 365 for Finance and Operations version 8.1.1

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations version 8.1.1. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Enumerations made extensible

These enumerations have been made extensible in this update.

Expand table

Enumeration
BankCodeType
CountryRegionType
MainAccountDimensionListProviderType
ProdSchedulingSortType
ProjAccountTypeSales
ProjBudgetBalancesGroupByOptions
ProjListStateType
ProjStatementType
SalesDeliveryDateControlType

Refactored methods

These methods have been refactored to support extensibility.

Expand table

Refactored methods
[Extensibility] Method signature change: WHSWorkExecuteDisplayListWork.displayListWorkStep
[Extensibility] Refactor WhsWorkExecuteDisplayAdjustOut to ProcessGuide framework
AssetJournal
AXSalesQuotationTable.setQuotationId
BankStatementBankAccountIdentify.searchBankAccountTable
BankStatementValidate.doValidate
BankStatementValidate.validateDate
BankStatementValidate.validatePeriodGap
BankStatementValidate.validatePeriodOverlap
BOM.validateWrite
BomCalcDialog.updateBomRoute
BOMCalcItem.createBomCalcItemAndAddToListBom
BOMCalcTable.transferToSalesLine
BOMCalcTable.transferToSalesQuotationLine
BomConsistOf.init

Extensibility changes in Dynamics 365 for Finance and Operations version 8.1

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations version 8.1. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Refactored methods to support extensibility

These methods have been refactored to support extensibility through chain of command, delegates, or by providing access to members.

Expand table

Method
AssetPost.createTrueUpDepreciation
AssetPost.createTrueUpDepreciation
AssetPostDisposal.post
AssetPostDisposal.postVoucherTransactions
AssetPostDisposal.postVoucherTransactions
AssetTable.buildComposedOf
AssetTable.createStruct
AxInventDim_SalesLine.setInventSiteId
BankChequePrint.printDocument
BankCodaProcessing.custSettlement
BankDeposit.InitFromLedgerJournalTrans
BankExchAdj_RU.calcAndPostCurrency
BankExchAdj_RU.calcAndPostCurrency
BankExchAdj_RU.calcBalance
BankPrintTestCheque.printCheque

Extensibility changes in Dynamics 365 for Finance and Operations update 8.0.4

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations update 8.0.4. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Refactored methods to support extensibility

These methods have been refactored to support extensibility through chain of command, delegates, or by providing access to members.

Expand table

Method
AgreementHeader.getModuleType
AssetSplit.construct
BankDepositSlipController.main
BankPositivePayExport.sendFileToUser
CaseDetailForm.getRecordsFromDataSource
CostSheetDesigner.DataSource:CostSheetCalculationFactor.validateWrite
CostSheetNodeCalculation.validate
CostSheetNodeCalculationRate.calcLowestLevel
CostSheetNodeCalculationSurcharge.equal
CustAccountStatementExtController.processParty
CustAccountStatementExtDP.insertNewRecords
CustAccountStatementExtDP.setSysDocuBrandDetails
CustBillOfExchangePost.postNextStep
CustBillOfExchangePostProtestHonored.postNextStep
CustCollectionJourController.runPrintMgmt
CustCollectionLetterCreate.createJournal
CustCollectionLetterCreate.run
CustCollectionLetterNote.CustCollectionLetterJour.active
CustCollectionLetterPost.processRow
CustCollectionLetterPost.validateCollectionLetter
CustFreeInvoiceCorrection.createInvoiceLines
CustInterestPost.main
CustInterestPost.validateInterestTrans
CustInvoiceJour.printJournal

Extensibility changes in Dynamics 365 for Finance and Operations update 8.0.3

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations update 8.0.3. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Refactored methods to support extensibility

These methods have been refactored to support extensibility through chain of command, delegates, or by providing access to members.

Expand table

Method
AccountingSourceExplorerProcessor.filterEntries
AgreementClassification.init
AgreementConfirm.createLineVolumeCommittmentHistory
AgreementConfirm.newAgreementConfirm
Agreementline.findLineForAutoMatch
Agreementline.getAgreementLinesForOrderLine
AgreementLine.getAgreementLinesForPurchReqLine
Agreementline.getAgreementLinesList
Bank_FR.checkControlText
Bank_IT.checkCIN
Bank_IT.checkRegistrationNum
BankAccountTrans.insert
BankAccountTrans.update
BankChequeCopy.fillTmpChequePrintout
BankChequePrint.printDocument

Extensibility changes in Dynamics 365 for Finance and Operations update 8.0.2

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations update 8.0.2. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Refactored methods to support extensibility

These methods have been refactored to support extensibility through chain of command, delegates, or by providing access to members.

Expand table

Method
AccDistProcessorProjectExtension.allocateExistingDistribution
AccDistProcessorProjectExtension.createDistributionLists
AccDistProcessorProjectExtension.ledgerDimensionAllocationList
AgreementConfirmationDP.processReport
Bank_FR.checkControlText
Bank_IT.checkCIN
Bank_IT.checkRegistrationNum
CustVendCheque.initTmpChequePrintout
FBSpedFileCreator_Contabil_BR.createRecordI052
HierarchyTemplateCopying_proj.createFromHierarchySource
HierarchyTreeLookup.datasource smmActivities.init
InventDim.validateFieldCombination
InventTransWMS_Register
InventUpd_Financial.updateFinancialIssue
InventUpd_Registered

Extensibility changes in Finance and Operations update 8.0.1

Article • 08/12/2022

This is a list of extensibility features that were implemented in Dynamics 365 for Finance and Operations update 8.0.1. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Refactored methods to support extensibility

These methods have been refactored to support extensibility through chain of command, delegates, or by providing access to members.

Expand table

Method
Class ProjControlPeriod::PeriodInsert
Class ProjInvoiceChoose::doSalesLine
Class CustInvoiceJour::printFreeTextJournal
Class ProjCostControl.createEmptyTransactionType
Class ProjEstimate::autoGenerateEstimateLinesFromTask
Class ProjEstimateDataContract.updateEstimates
Class ProjForecastBudget.Run
Class ProjHierarchyProvider.preDeleteHierarchy
Class ProjPlanVersionsManager::CopyTasks
Class ProjPlanVersionsManager.createDraftFromPublishedVersion
Class ProjPlanVersionsManager.PublishQuotationSubHierarchy
Class ProjPlanVersionsManager.CreateDraftVersion
Class ProjTask.addTask
Class ProjInvoiceProposalCreateLines.performTransTypeSelectionCtrlLookup
Class VendOpenTrans.editMarkTrans

Extensibility changes in the Finance and Operations version 8.0

Article • 08/12/2022

Hard-sealed application models

In Dynamics 365 for Finance and Operations version 8.0, all of Microsoft's application models have been hard-sealed. Overlayered code in these models will now produce compilation errors. The only supported customization model is through extensions. If you cannot customize these models through extension, then you will have to make a request to Microsoft to enable extensibility by changing the standard application.

The following table includes a list of models that are now hard-sealed with this release.

Expand table

Module	Model
ApplicationCommon	ApplicationCommon
ApplicationSuite	Electronic Reporting Application Suite Integration
ApplicationSuite	Foundation Upgrade
ApplicationSuite	Foundation
ApplicationSuite	SCMControls
ApplicationSuite	Tax Books Application Suite Integration
ApplicationSuite	Tax Engine Application Suite Integration
CaseManagement	CaseManagement
Currency	Currency
DataImpExpApplication	DataImpExpApplication
DataUpgrade	DataUpgrade
Directory	Directory
Directory	SecurityReports
GeneralLedger	GeneralLedger
Ledger	Ledger

Extensibility changes in Finance and Operations, Enterprise edition 7.3

Article • 08/12/2022

This article lists the extensibility features that were released in Dynamics 365 for Finance and Operations, Enterprise edition 7.3. For more information about the schedule of changes that support extensibility, see Application extensibility plans.

Soft-sealed application models

This release marks the last release before all models will become hard-sealed, and as a step toward this all application models are now soft-sealed. Soft-sealed models still allow for making overlayered code, but warnings will be generated when you compile the overlayered code.

> **Note**
>
> You can still overlayer code, but extension is the recommended approach.

The following table includes a list of the models that are soft-sealed with this release.

Expand table

Module	Model
ApplicationCommon	ApplicationCommon
ApplicationSuite	Electronic Reporting Application Suite Integration
ApplicationSuite	Foundation Upgrade
ApplicationSuite	Foundation
ApplicationSuite	SCMControls
ApplicationSuite	Tax Books Application Suite Integration
ApplicationSuite	Tax Engine Application Suite Integration
CaseManagement	CaseManagement
Currency	Currency

Extensibility changes in Finance and Operations, Enterprise edition (July 2017)

Article • 08/12/2022

This is a list of extensibility features that were implemented in the Dynamics 365 for Finance and Operations, Enterprise edition (July 2017). This version was released in July 2017 and has a build number of 7.2.11792.56024. For more information about the schedule of changes that support extensibility, see Application extensibility roadmap.

Soft-sealed application models

The following application middle-tier models were soft-sealed in this release. Overlayered code in these models will generate warnings on compilation.

Expand table

Category	Model
Application Frameworks	CaseManagement
Application Frameworks	Dimensions
Application Frameworks	Directory
Application Frameworks	Organization
Application Frameworks	Currency
Application Frameworks	ApplicationCommon
HCM Core Models	3817938
Tax Models	Tax
Tax Models	Tax Books
Tax Models	Tax Books Application Suite Integration
Tax Models	Tax Engine Application Suite Integration
Tax Models	Tax Engine Configuration
Tax Models	Tax Engine Interface

Intrusive customizations

Article • 08/12/2022

This article defines the characteristics of an intrusive customization. Intrusive customizations are the major obstacle to keeping continuous upgrade costs close to zero. Some types of intrusive customizations can be prevented by tooling, whereas other types remain the responsibility of the author of the extension. The X++ compiler and Microsoft Visual Studio designers will prevent some types of intrusive customizations. However, a subset of intrusive customizations can't be detected by tooling but might still prevent continuous upgrades. Ultimately, the developer is responsible for avoiding intrusive customizations.

A customization that violates any of the following principles is intrusive.

> **Note**
>
> When extending other solutions you are expected to be responsible. This includes:
>
> - Accepting the responsibility of your extension. You are introducing new behavior and therefore own the full responsibility of the change.
> - Allowing other extensions to co-exist. Recognize that you are not the only consumer of an extension point. For example, only respond when needed.
> - Only adding extensions that are coherent with the extension point. The longevity of a solution is defined by its resilience to change. Recognize that extension points may be exercised in more or fewer scenarios in future versions. For example, only add relevant validation logic to a validate() method.
> - Avoiding dependencies on implementation details. Implementation details are likely to change in future versions. Make your solution resilient by avoiding dependencies on local variables, call-stacks and, calling sequences and avoid using reflection.

Don't change type definitions

Types are referenced by their definition. A change to a type's definition is a breaking change and requires that all references be updated. It's impossible to ensure that future references will be implemented correctly (for example, in the model that hosts the type). There are several implications:

Class extension model in X++

Article • 02/03/2025

This article describes the class extension model in X++.

Extension is a term used for features that let you extend existing artifacts in a new model. There are rich ways to extend both the X++ code and metadata. This article describes how X++ code can be extended and add methods and state to artifacts defined in other models without recompiling those models.

A similar code extension mechanism already exists for X++ and is modeled after the corresponding feature in C#. This extension mechanism allows a class to be designated as an extension class through a naming convention and by hosting public static methods. These classes are a programming utility concept that allows you to add methods to existing types, namely the type of the first parameter to the extension method. In this way, the method is "extending" the new class with a method that it didn't originally have. This article describes the next step that offers a more capable and natural extension story. In object-oriented programming, the term *extend* has a well-defined meaning. When we say, "class B extends class A," we mean that B inherits from A, that A is B's parent class, and the usual object-oriented rules are implied. This term in X++ syntax is used in class declarations to express this relationship. The term *extension* can describe metadata that has contributions from several models. *Class augmentation* is used to designate the relationship between a class A in a base model and a class B in a model that depends on it. Where B provides additional functionality to class A in the context of that model. We'll continue to use the term *extension class*, because it's so prevalent.

The effective class concept

It's useful to have a term for a class that consists of the public members of the augmented artifact and all the public members of all the class extensions that augment that artifact. This class is called the effective class in a given model. The following illustration shows an artifact, **MyArtifact**, that is defined in a base model, **MyModel**, and two dependent models that have extension classes for **MyArtifact**.

Class extension - Method wrapping and Chain of Command

Article • 08/12/2022

The functionality for class extension, or class augmentation, has been improved. You can now wrap logic around methods that are defined in the base class that you're augmenting. You can extend the logic of public and protected methods without having to use event handlers. When you wrap a method, you can also access public and protected methods, and variables of the base class. In this way, you can start transactions and easily manage state variables that are associated with your class.

For example, a model contains the following code.

X++

```
class BusinessLogic1
{
    str doSomething(int arg)
    {
        // ...
    }
}
```

You can now augment the functionality of the **doSomething** method inside an extension class by reusing the same method name. An extension class must belong to a package that references the model where the augmented class is defined.

X++

```
[ExtensionOf(classStr(BusinessLogic1))]
final class BusinessLogic1_Extension
{
    str doSomething(int arg)
    {
        // Part 1
        var s = next doSomething(arg + 4);
        // Part 2
        return s;
    }
}
```

In this example, the wrapper around **doSomething** and the required use of the **next** keyword create a Chain of Command (CoC) for the method. CoC is a design pattern

Naming guidelines for extensions

Article • 06/13/2024

High level guidance: use prefixes to reduce conflicts and improve identification

Naming model elements

Every element in a model must have a name that is unique across all models at installation time. However, at installation time, you don't know the names of all the models that your model might be installed together with. To accommodate this situation, every element name should include a prefix that is specific to your solution. By including this prefix when you name elements in your model, you significantly reduce the risk of naming conflicts.

- If a model contains multiple solutions, each solution in the model can be identified by a different prefix.
- You must carefully choose the prefix to minimize the risk that other models from other parties use the same prefix for their elements.

When you extend functionality in other models, elements that are being extended already contain a prefix. However, you should not add your prefix to the extension elements, so that the names include multiple successive prefixes. Instead, you should include your prefix or another term or abbreviation as an infix when you name extension elements.

Naming extensions

An extension element, such as a table extension, view extension, or form extension, must have a unique name that minimizes the risk of conflicts with extensions in other models. To minimize the risk of conflicts, the name should include a term, abbreviation, or infix that distinguishes the extension from other extensions to the same element in other models.

- Include either the name of the model where the extension element resides or the prefix that the extension is associated with. For example, a Warehousing module extends the HCMWorker table and uses the **WHS** prefix in the name of all other elements. In this case, the extension might be named **HCMWorker.WHSExtension**. Notice that the prefix that is used to name other elements in the module is inserted as an infix in the name. As another example, an extension of the ContactPerson table in the **ContosoCustomizations** model might be named

Relax model restrictions to refactor overlayering into extensions

Article • 08/12/2022

Development tools for finance and operations apps, starting with version 8.0, do not allow Microsoft code to be customized by using over-layering. Instead, extension capabilities should be used to modify and add behavior. The "no over-layering" restriction is a key part of the evolution of the product toward providing customers with a cloud service that is simple to update and always running the most recent version possible to allow all customers to receive the benefits of the latest features and fixes.

After you upgrade code from Dynamics AX 2012 or from Dynamics 365 for Finance and Operations version 7, any customizations that still use over-layering will cause errors when you compile your code. To refactor the code, the over-layering restriction can be temporarily relaxed in the model descriptor file of the model that is being over-layered. This temporary relaxation only works on development and demo environments and cannot be deployed on runtime environments like Standard Acceptance Test (or higher) sandbox or production environments. Relaxing the descriptor restriction will enable the code to be gradually refactored to extensions, compiled, run, and then tested.

Detailed process

Complete the following steps to relax model restrictions. This procedure can be completed on a cloud environment or a local virtual machine (VM).

1. Deploy the development environment for the finance and operations app.
2. Run the Lifecycle Services (LCS) code upgrade service to upgrade the solution.
3. Temporarily allow over-layering in Microsoft models as needed to enable compilation.

   a. Locate the desired model within the C:\AOSService\PackagesLocalDirectory folder.

   b. Navigate to the descriptor folder. For example, \Currency\Descriptor.

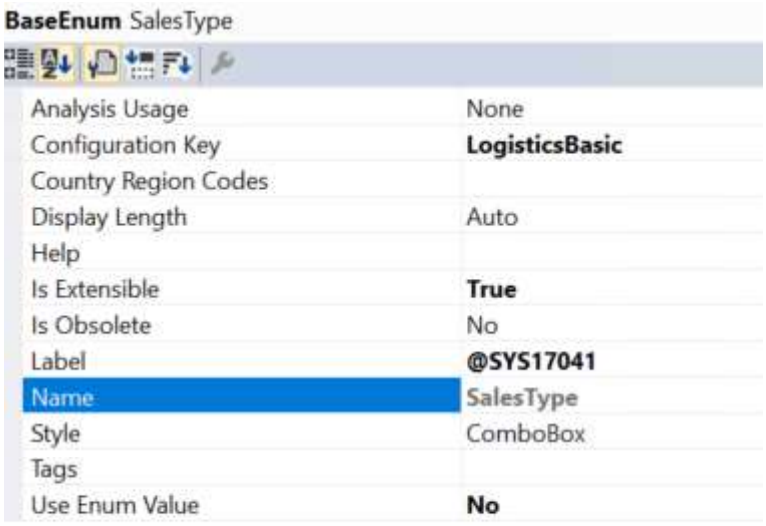
   c. Open the XML file. For example, Currency.xml.

   d. Add a **Customization** metadata element inside the **AxModelInfo** metadata element to indicate **AllowAndWarn** so that the start of the file now looks like this.

Add values to enums through extension

Article • 08/12/2022

To add new values to an enum, you should extend the enum. Any enum that is marked as **Extensible** (**IsExtensible** = **true**) can be extended. You can find the extensibility information in the **Properties** window in Microsoft Visual Studio, as shown in the following illustration.



When enum values of extensible enums are synchronized, the integer values of the baseline enum are deterministic, whereas the integer values of the extension enum values are non-deterministic. The values are generated during synchronization. Therefore, you can't have logic that depends on the integer value of the enum values. Here are some examples:

- Range comparisons, such as `<`, `>`, and `..`
- Modeled ranges in views and queries
- Query ranges that are created from code

Usually, an extended enum must have its own implementation wherever it's used. Look for all uses of the enum, and uptake the implementation where it's required. Here are some significant places to look for:

- Switch blocks:
  - If the switch block doesn't have a default case block or a default case block that doesn't throw an exception, handle the extended enum value by subscribing to a delegate, if a delegate is provided. Otherwise, add a post-event handler to the method.
  - If the enum is used in a switch that has a default case block that throws an exception, contact Microsoft to request a delegate.

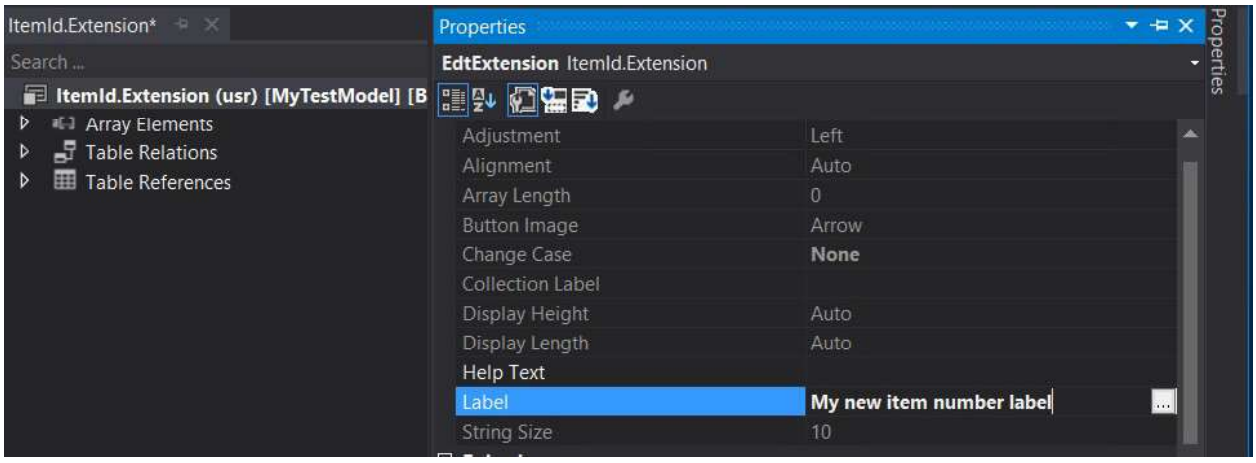
Modify extended data types (EDTs) through extension

Article • 08/12/2022

There are several properties that can be customized on existing extended data types (EDTs) through extension:

- Label
- Help text
- Form help
- Country region codes
- String size
  - You can only modify the value if the EDT does not extend from another EDT.
  - You can only set the new String size to a value equal to or larger than the base EDT value.
- Decimals (NoOfDecimals property)
  - For more information, see Extending decimal point precision for selected data types.

You modify the properties as you would for newly added elements, using the property sheet.



After compiling the code, you can see the changes in the application.

Register subclasses for factory methods

Article • 08/12/2022

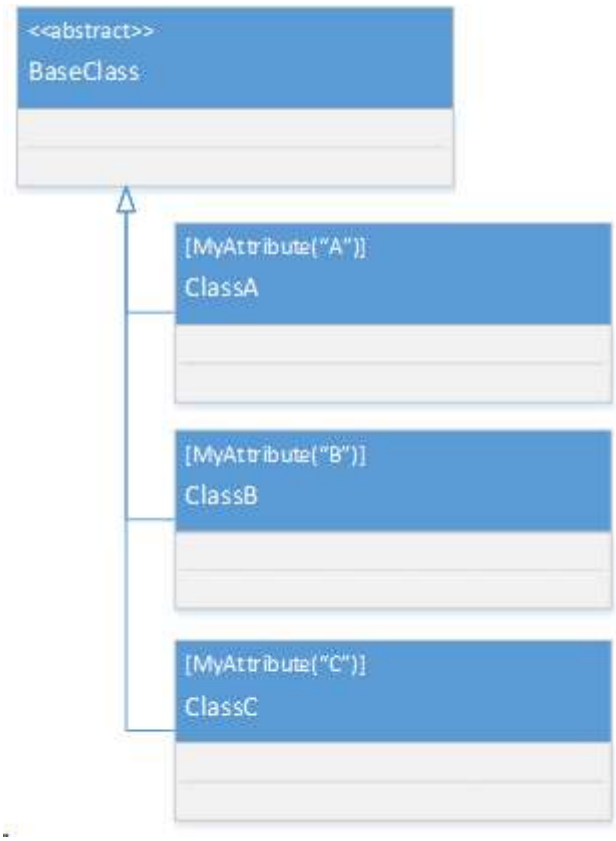
Class inheritance is a central concept in X++, as in other object-oriented languages. The object-oriented strategy pattern is used throughout the X++ business logic. In this pattern, variations in behavior can be encapsulated by subclasses, and the business process uses an abstract base class or interface. A factory method determines the variation that is used, by creating an instance of a specific subclass.

This article describes how to register your own variations for the factories.

In X++, the factories use reflection to perform the following tasks:

- Find the correct subclass. The factory uses an extension framework to search all subclasses in a hierarchy for a specific set of attributes. If the attributes that decorate a subclass match the parameters that were passed to the factory, that specific class is used.
- Create an instance. After the right type is identified, reflection is used to create an instance of the class.

The following illustrations shows a typical decorated hierarchy.



In X++, two extension frameworks serve the same purpose. The implementer of the factory method determines which extension framework should be used:

Respond by using EventHandlerResult

Article • 08/12/2022

Some delegate methods are implemented so that they can request a response from subscribing delegate handler methods. The delegate calling logic then uses the response from a potential subscriber when it continues execution after the response has been received. These delegate methods usually have a signature that has an **EventHandlerResult** parameter as the last parameter. However, because of the support for the **EventHandlerAcceptResult** and **EventHandlerRejectResult** types, the parameter can be of any type that implements the **IEventHandlerResult** interface.

- In general, the logic that is implemented in the delegate handler method should contain a condition that verifies that the subscribing logic is responsible for providing a response. It should also include logic to provide the response in the form of a result.
- When the delegate handler method must provide the response to an **EventHandlerResult** object parameter, the subscribing logic might also contain logic to calculate or retrieve the result.
- When the condition and the response logic are implemented, the calculation of the result must occur only when the condition is evaluated to **true**.
- All the subscribing delegate handler methods are run when a delegate is called. Therefore, you should make sure that the overhead of running your method is as low as possible when the method isn't responsible for providing a response. Therefore, make sure that the condition is evaluated to **false** as quickly as possible when your delegate handler method isn't responsible for providing a result.

Examples

The following example shows a delegate handler that has a condition in the form of a **switch** statement. The delegate handler also has logic to provide a response in the form of the result. The responding logic is run only when the condition is evaluated to **true**.

X++

```
[SubscribesTo(tableStr(InventWarehouseEntity),
delegateStr(InventWarehouseEntity, validateWarehouseTypeDelegate))]
public static void
validateWarehouseTypeIsSupportedStandardDelegateHandler(InventLocationType
_inventLocationType, EventHandlerResult _result)
{
    switch (_inventLocationType)
    {
```

Extend the RunBase class

Article • 08/12/2022

When you extend functionality of the application suite, you will encounter classes that extend the **RunBase** class. This article shows how a **RunBase** class can be augmented end to end.

For example, you want to extend the SysUserLogCleanup class. Out of the box, this class can delete records from the SysUserLog table. However, you want to archive these records to a different table before they are deleted.

The SysUserLogCleanup class is a **RunBase** class. The **RunBase** class has a dialog box, where the user is prompted for parameters before the class is run. For this example, we will add a toggle button control to the dialog box, get the value of the control, act on the value in the run method, and make sure that the value is serialized via the pack and unpack methods. Serialization helps guarantee that the user's last selection is presented again if the dialog box is reopened. It also helps guarantee that the settings are applied if the class is run in the background.

To avoid collisions with other eventual extensions, we followed these best practices:

- **Prefix members and methods**. In the example, the prefix "my" is used. This practice is important, because it helps prevent name clashes with other extensions and future versions of the augmented class.
- **Use RunBase.packExtension() and RunBase.unpackExtension()**. These methods provide serialization in a nonintrusive manner. They enable serialization of multiple extensions of the same class. The methods are available starting in Platform Update 5.

The following example shows how to implement this scenario.

X++

```
[ExtensionOf(classStr(SysUserLogCleanup))]
final class MySysUserLogCleanup_Extension
{
    // static members
    static private SysUserLogCleanup myRunningInstance;

    // Extending class state...
    private boolean myArchive;
    private DialogField myDialogArchive;
    #define.CurrentVersion(1)
    #localmacro.CurrentList
```

Customize application startup by using delegates

Article • 08/12/2022

In Dynamics AX 2012, there were customization points that allowed you to subscribe to events (Application.Startup delegates) that were raised when the client was initializing. These events were deprecated because there is no concept of a rich client. On the server, only server sessions are considered, however because you can migrate logic from previous releases, new events have been added to the **ApplicationStartupEventManager** class.

The following sections highlight the new data sources that you can add to existing forms by using extensions.

static delegate void onSystemStartup()

- This event occurs when the system starts up.
- It is raised once per AOS upon startup.

static delegate void onFirstTimeUserInteractiveSessionCreated()

- This event occurs when the system is creating an interactive session for the first time for a user.
- It is raised once per user per AOS.

static delegate void onFirstTimeUserNonInteractiveSessionCreated()

- This event occurs when the system is creating a non-interactive session for the first time for a user.
- It is raised once per user per AOS.

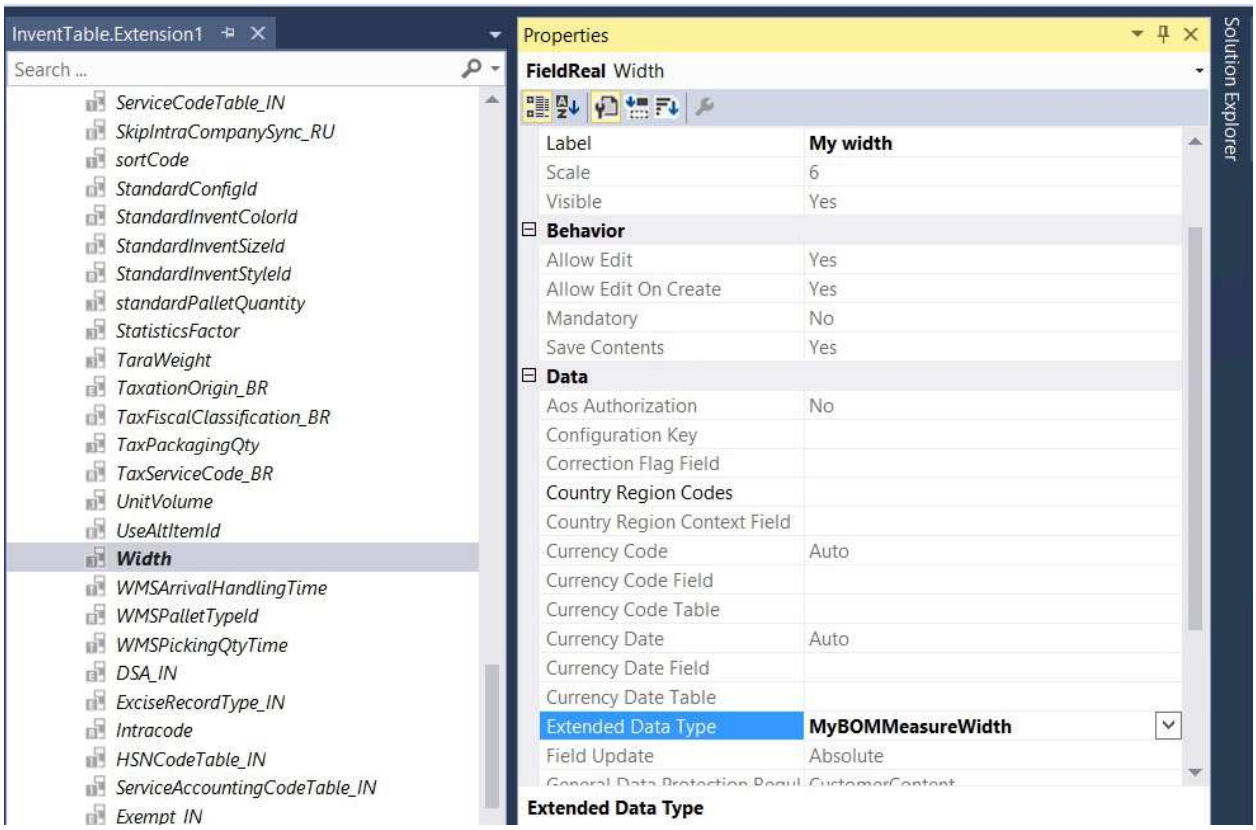
static delegate void onInteractiveSessionCreated()

Modify existing fields in a table through extension

Article • 08/12/2022

To modify properties on an existing field in a table, you must first create an extension for the table. You can modify the following properties:

- **Label**
- **Help text**
- **Country Region Codes**
- **Extended Data Type –** You can select only extended data types (EDTs) that are derived from the currently selected EDT. The lookup in the property sheet is filtered so that only those EDTs are shown. For example, to edit the EDT on the **Width** field in the InventTable table, you can create a derived EDT that is based on **BOMMeasureWidth**, and then modify the **Extended Data Type** property on the **Width** field in the **InventTable** extension. In this way, you can modify the look and feel of the **Width** field in the user interface when the new package is deployed.

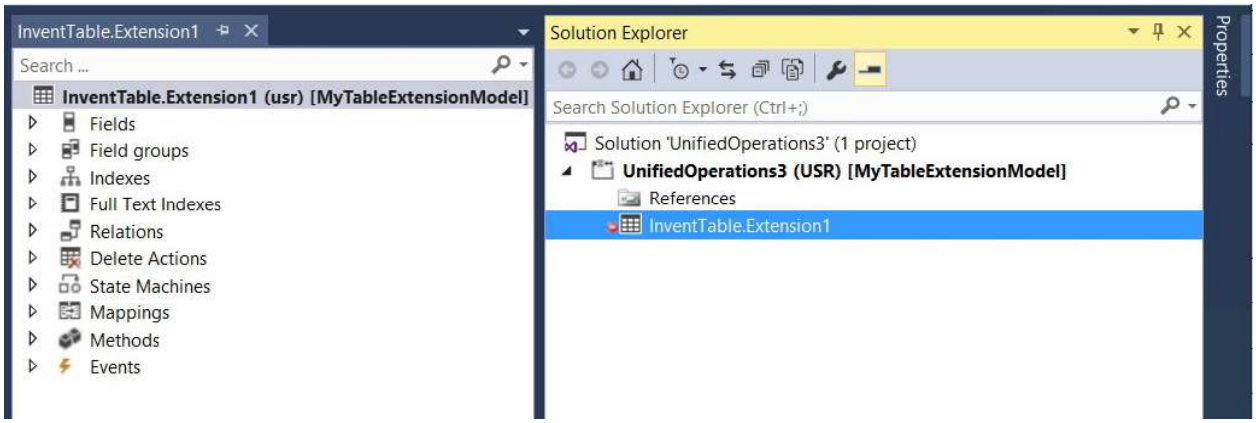


Feedback

Add fields to tables through extension

Article • 08/12/2022

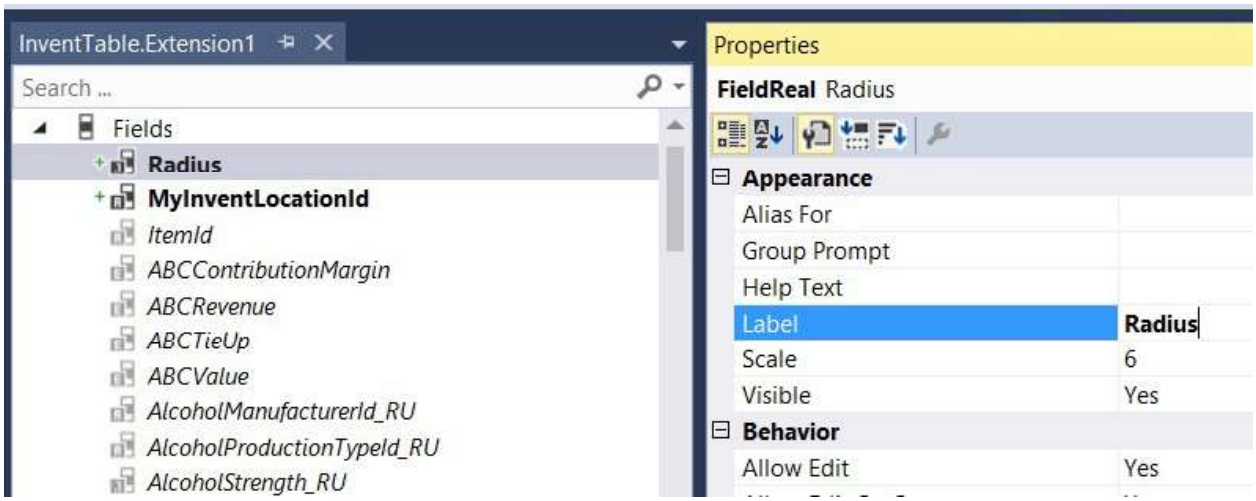
To add a new field to an existing table, you must first create a table extension. For example, to add a field that holds the radius of the released product, you must create an extension for the InventTable table in your model, as shown in the following illustration.



You can now add the field to the extension, just as you would add a field to a table in your model. You can use two methods:

- In the designer, right-click the **Fields** node, select **New**, and then select the type of field to add.
- Drag an existing Extended Data Type or Base Enumeration from your project onto the **Fields** node.

When you've finished, you can modify the properties of the new field. In the following illustration, only the **Label** property was modified.



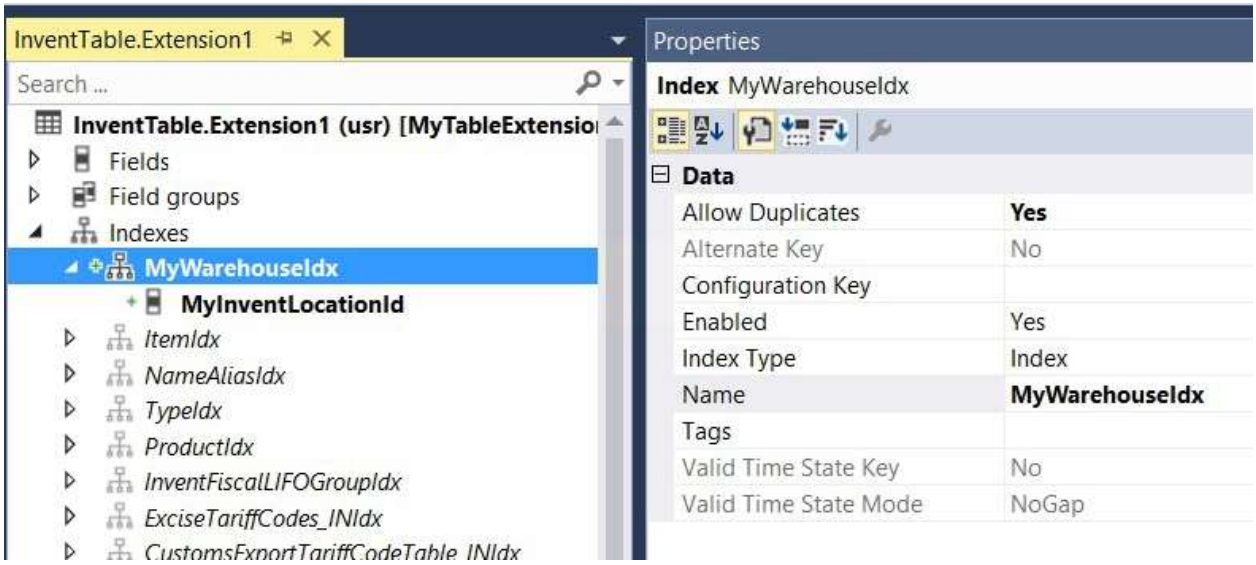
You can now optionally add the new field either to one of the existing field groups or to a new field group that you create. In the following illustration, the **Radius** field was added to the **PhysicalDimensions** field group.

Add indexes to tables through extension

Article • 08/12/2022

Often, you extend tables so that you can store additional data for later but also quickly access the data that is based on the new fields. Therefore, it's often beneficial to have a dedicated index that speeds up the database search. You can add a new index to an existing table through extension. To add an index to an existing table, you extend the selected table and then create an index just as you would create an index on a new table. You can add both new and existing fields so that they are part of the new index.

In the following illustration, an InventTable extension is used to define an index for a new field on the InventTable table.



> ⚠ **Warning**
>
> You should not use this approach to create unique indexes. This change is an intrusive change that might break the solutions of other independent software vendors (ISVs) if those solutions are deployed in the same environment. This capability will be removed in future platform releases.

Feedback

Was this page helpful? Yes No

Provide product feedback

Add relations to tables through extension

Article • 08/12/2022

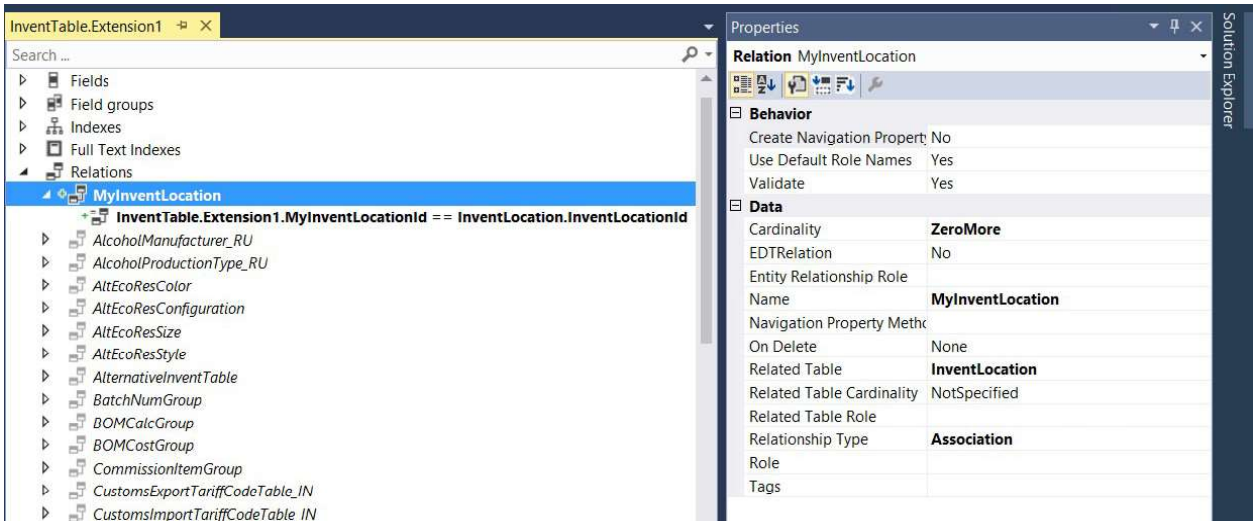
To enable rich and secure interactions with data in multiple tables, you must help guarantee referential integrity by defining relations that describe the link between two tables. By defining relations, you enable validation of the data that is entered and lookup capabilities for the related information.

You can add a new relation by extending a table.

In the following example, a new field, **MyInventLocationId**, is added to the InventTable table. This field is a reference to the InventLocation table that contains warehouses.

1. In the new extension model, create an extension of the InventTable table.
2. Create a new relation, just as you would create a relation on a regular table.
3. Specify the **Related Table**, **Relationship Type**, and **Cardinality** properties, and any other properties that apply to the relation.
4. Add the link by specifying the fields from the InventTable table and the InventLocation table that have the same value. In this case, the fields are **MyInventLocationId** in the InventTable table and **InventLocationId** in the InventLocation table.

The following illustration shows the new relation.



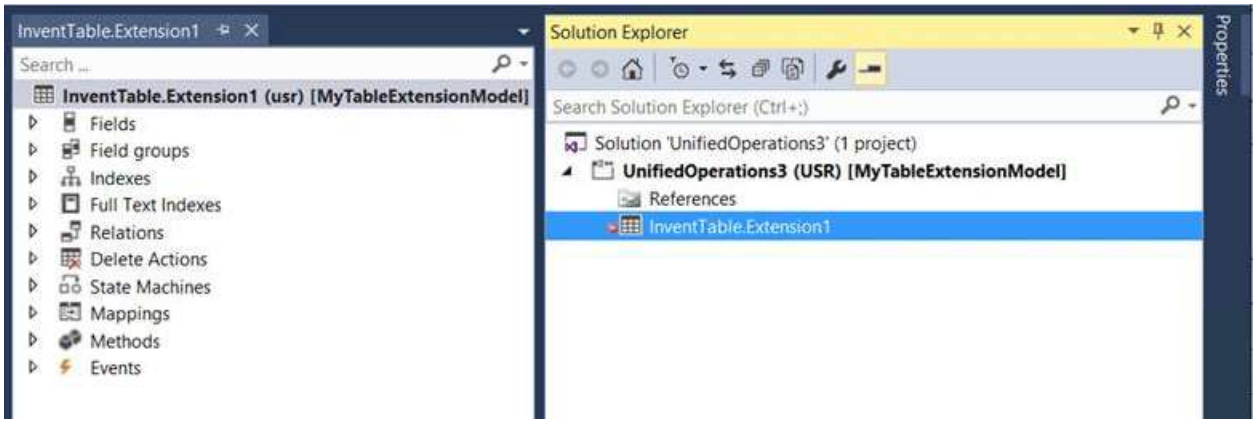
Troubleshooting

Navigation property methods not working

Modify table properties through extension

Article • 08/12/2022

To modify properties on a table, you must create an extension of that table. In Application Explorer, right-click the table, and then select **Create extension**. A new table extension is created in the selected project, as shown in the following illustration.



You can now modify the following properties through the property sheet:

- Country Region Codes
- Created By
- Created Date Time
- Form Ref
- Modified By
- Modified Date Time
- Preview Part Ref
- Tags
- Title Field1
- Title Field2

By setting the **Created By**, **Created Date Time**, **Modified By**, or **Modified Date Time** property to **Yes**, you help guarantee that a corresponding field is added to the table. Corresponding tracking information about the user is then stored in the table when records are created or updated. You can't set these properties to **No** if they are set to **Yes** on the base table.

By adding country or region codes to the list, you help guarantee that the corresponding table is also applicable when the system runs in the context of the specified country or region.

Add methods to tables through extension

Article • 08/12/2022

When you extend the business logic that is related to a table, the general coding principles that help keep your code clean still apply. Therefore, you must eventually encapsulate actions in separate methods on the table. In Microsoft Dynamics AX 2012, you completed that task by adding the method directly on the table through overlayering. To complete the same task through extension, you use a different approach. Specifically, you create an augmentation class.

For example, a new field that is named **MyInventLocationId** was added to the InventTable table through extension. A data event handler was also created for the **Inserting** event, and you must implement the logic of filling the new field there. To encapsulate that action, you will create a new method on InventTable and name that method **myDefaultInventLocationId**.

You first create a new class in the extension model. This class will augment the InventTable table, and enable access to the table's fields and methods in a manner that is easy to read and understand. It's important that you choose the correct name for your augmentation class. This name must be unique across all types in all models that are deployed. For more information, see Naming guidelines for extensions.

X++

```
[ExtensionOf(tableStr(InventTable))]
final class InventTableMy_Extension
{
    public void myDefaultInventLocationId()
    {
        // This would have partner specific logic to initialize the new
field.
        this.MyInventLocationId = this.inventLocationId();
    }
}
```

You can now add new methods to the augmentation class. These methods will then appear in IntelliSense for variables of the **InventTable** type, just as if they were defined directly on the table. This behavior applies to both static methods and instance methods.

There are a few rules for augmentation classes:

Perform business actions throughout the lifecycle of table records

Article • 08/12/2022

As part of your business workflows, records are regularly inserted, updated, and deleted. To customize the system behavior, you can hook into some of the record operations that are most often used. For example, you can fill additional fields on the record, perform additional data validation, or insert additional data into related tables. Several events that are available on the table let you achieve those customizations through extensions. Your code can subscribe to these events, and can insert your logic before or after the event is run.

For a list of the events that you can subscribe to, see the "Table extensions" section in Customize through extension and overlayering.

Feedback

Was this page helpful? Yes No

Provide product feedback

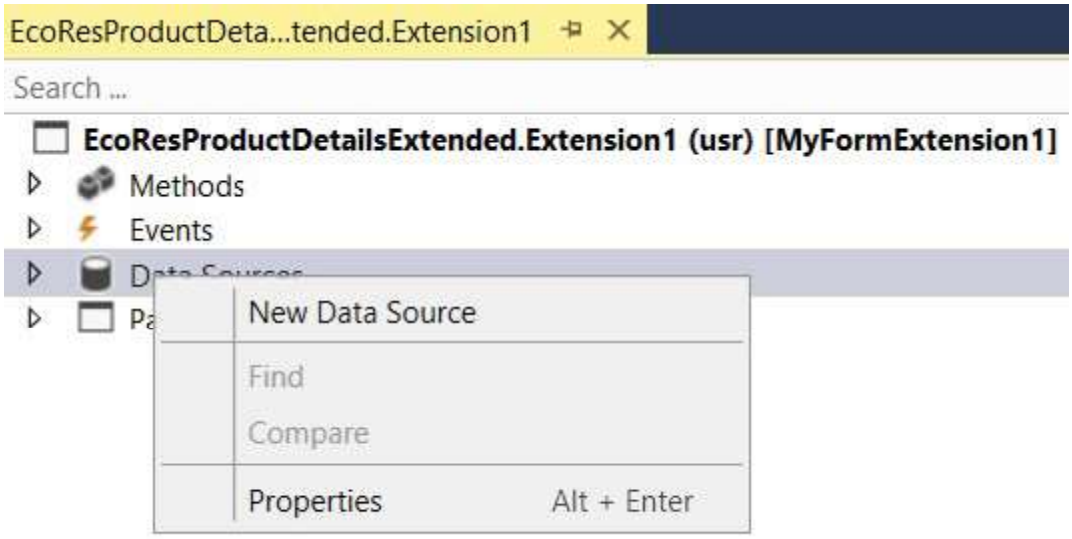
Add data sources to forms through extension

Article • 08/12/2022

Often, the information that is stored in existing tables doesn't satisfy customer requirements. Therefore, additional tables must be created, and data from those tables must be shown on pages.

You can add new data sources to existing forms through extension. Follow these steps.

1. In the extension model, create a form extension for the selected form.
2. Right-click the form extension, and then select **New Data Source**.

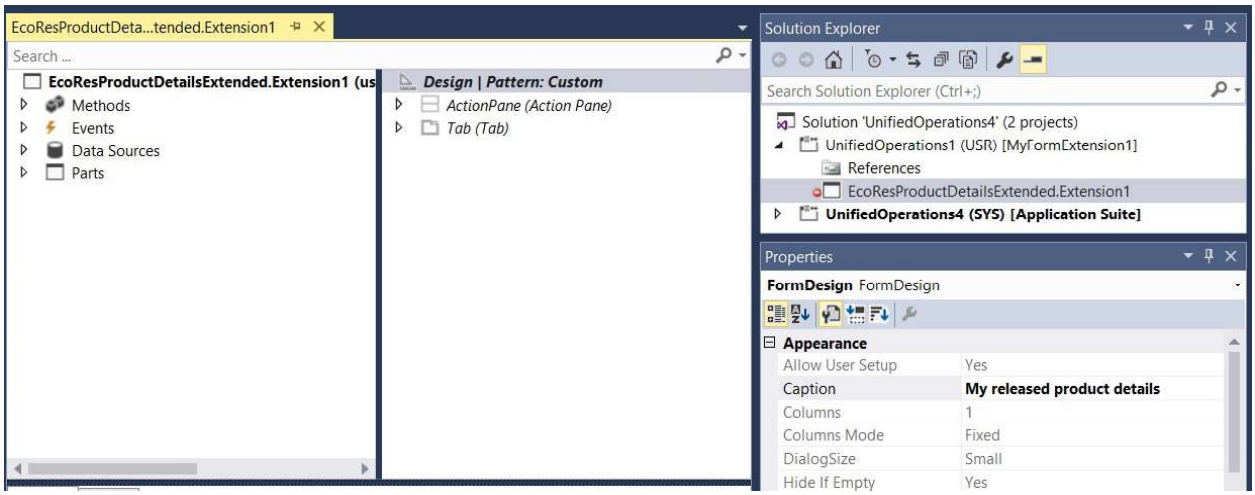


3. Specify the **Table** property and other required properties on the data source. For example, define how the data source should be linked with the other data sources for the form.
4. Drag fields from the new data source into the form design, as shown in the following illustration.

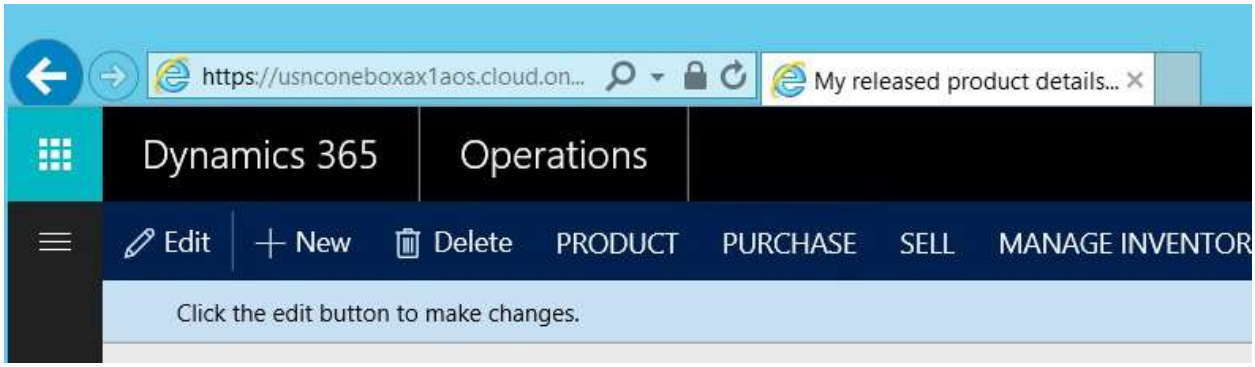
Change the captions of forms through extension

Article • 08/12/2022

The form caption appears in the page tab next to the web browser's Address bar and helps the user identify the page that is currently open. In metadata, the form caption is represented by a property on the form design. Therefore, to change the caption, you must modify the **Caption** property on the form design. You can make this change through extension. Create an extension of the selected form in the extension model, and then change the **Caption** property as usual.



The following illustration shows what the form caption looks like in a browser.



> **Note**
>
> None of the other properties on the form design can be changed.

Feedback

Modify the properties of form controls through extension

Article • 08/12/2022

Often, the way that users interact with the application requires modification. Typically, you hide or disable controls on the page, replace standard labels with labels that are more appropriate, or even add new controls that the user requires. You can also create a form extension.

> **Tip**
>
> You can achieve even more flexibility through event subscription on form control events. This approach is discussed in other topics. In this article, the focus is on metadata changes.

Example

The customer requires changes to the **Manage inventory** FastTab on the **Released product details** page. You must change the label of the FastTab, disable the field group that shows the catch weight configuration, and add new controls. (For this example, the new controls aren't bound to existing fields in the data source).

Follow these steps to make the required changes.

1. In the extension model, create an extension of the **EcoResProductDetailsExtended** form.
2. Navigate through the form design tree to the **TabPageInventory** tab page (**Design** > **Tab** > **Details** > **GroupDetails** > **TabHeader** > **TabPageInventory**), select it in the designer, and open the **Property** sheet.
3. Update the **Caption** property to the desired value.

Extend the scope of number sequences

Article • 08/12/2022

This article shows you how to extend the number sequence scope.

The scope of a number sequence defines which organization uses the number sequence. The scope can be **Shared**, **Company**, **Legal entity**, or **Operating unit**. **Company** and **Legal entity** scopes can be combined with fiscal calendar periods to create even more specific number sequences. New number sequence scopes can be added through extensions.

To create a new scope and have it show up in the client, complete the following steps:

1. Create an enum extension for **NumberSeqParameterType**. In the extension, add a new enum value for the new scope type.
2. Create an enum extension for **NumberSequenceType**. Add a new enum value for the new scope type. The **NumberSequenceType** enum is used in **NumberSequenceTableEntity** and **NumberSequencesReferenceEntity**.
3. Create a table extension for the **NumberSequenceScope** table. Add a new field for the new scope type.
4. Create an extension class for **NumberSeqScope** class.
    a. Create a post handler for the **NumberSeqScope::getValidScopeTypes** method. In the event handler method, add the new scope type to the valid scope types list.
    b. Create an event handler for the **onGetFormatSegmentShortName** delegate. In the event handler, return the short name for the new scope type.
    c. Create a post handler for the **NumberSeqScope::find** method and add logic for the new scope type so the corresponding **NumberSeqScope** instance can be found.
    d. Create a post handler for the **NumberSeqScope::getId** method and add logic for the new scope type, so the corresponding record can be found (or created if it does not exist) in the **NumberSequenceScope** table.
5. Create an extension class for the **NumberSequenceScopeFactory** class. Add a method that initializes the new **NumberSeqScope** that represents the new scope type.
6. Create a form extension for the **NumberSequenceDetails** form. Add controls that show the new scope type to the **Scope** tab.
7. Create an extension class for the **NumberSequenceDetails** form.
    a. Create a post handler for the **updateScopeControlVisibility** method to show the new scope type when the new scope type is selected in the **Scope** box.

Add new inventory dimensions through extension

Article • 08/12/2022

This article provides a high-level overview of how to add new inventory dimensions through extensions. It also includes information about how to access a sample application that contains an actual implementation.

Solution overview

The cornerstone in this solution is that multiple roles participate in the life cycle of adding new inventory dimensions through extensions. The following description simplifies and generalizes this solution, however, in real life there's overlap between the roles, and sometimes it might even be the same person filling several roles.

Microsoft's role

Microsoft provides a finite set of unused dimension fields.

In addition to the 15 existing dimensions, Microsoft supports 12 generic dimensions:

- 10 string-based
- 1 real-based
- 1 utcdatetime-based

This brings the total number of inventory dimensions in the standard application to 28:

- 5 product dimensions: Color, Size, Style, Config, and Version
- 5 tracking dimensions: Serial, Batch, Owner, Profile (Russia only), and GTD (Russia only)
- 6 storage dimensions: Site, Warehouse, Location, Status, License Plate, and Pallet (for upgrade and migration only)
- 12 unassigned generic dimensions: InventDimension1 to InventDimension12

Microsoft provides the physical schema.

ISV role

The ISV adds new inventory dimensions. The ISV solution provides all the specific functionality for the dimension - it must be strong-typed, maintainable, testable, and

Price and discount extensibility

Article • 08/12/2022

In Microsoft Dynamics 365 for Finance and Operations, Enterprise edition 7.3 and later, the pricing area is extensible. Some common customizations for price and discounts include:

- Adding new price group types and the corresponding price types (enum values for **PriceType** and **PriceGroupType**), in addition to adding search mechanisms for the new price types.
- Modifying the price and discount search, including passing in any additional parameters to the **PriceDisc** class.

PriceType and PriceGroupType enums

Typically, adding a new type of price discount search starts with adding a new enum value in the two enums: **PriceType** and **PriceGroupType**. To support extensibility, **PriceType** and **PriceGroupType** enum values are now encapsulated in the class hierarchies **PriceGroupTypeTradeAgreementMapping** and **PriceTypeTradeAgreementMapping**, respectively. These can be extended for any new **PriceType** and **PriceGroupType** extended enum values.

The mapping of fields on the **Customer**, **Vendor**, and **InventTable** tables that correspond to the price types is defined in **PriceTypeTradeAgreementMapping**.

The following diagram highlights the implementation. Note that the methods show only one of the sub-classes. The implementation needs to be on each sub-class.

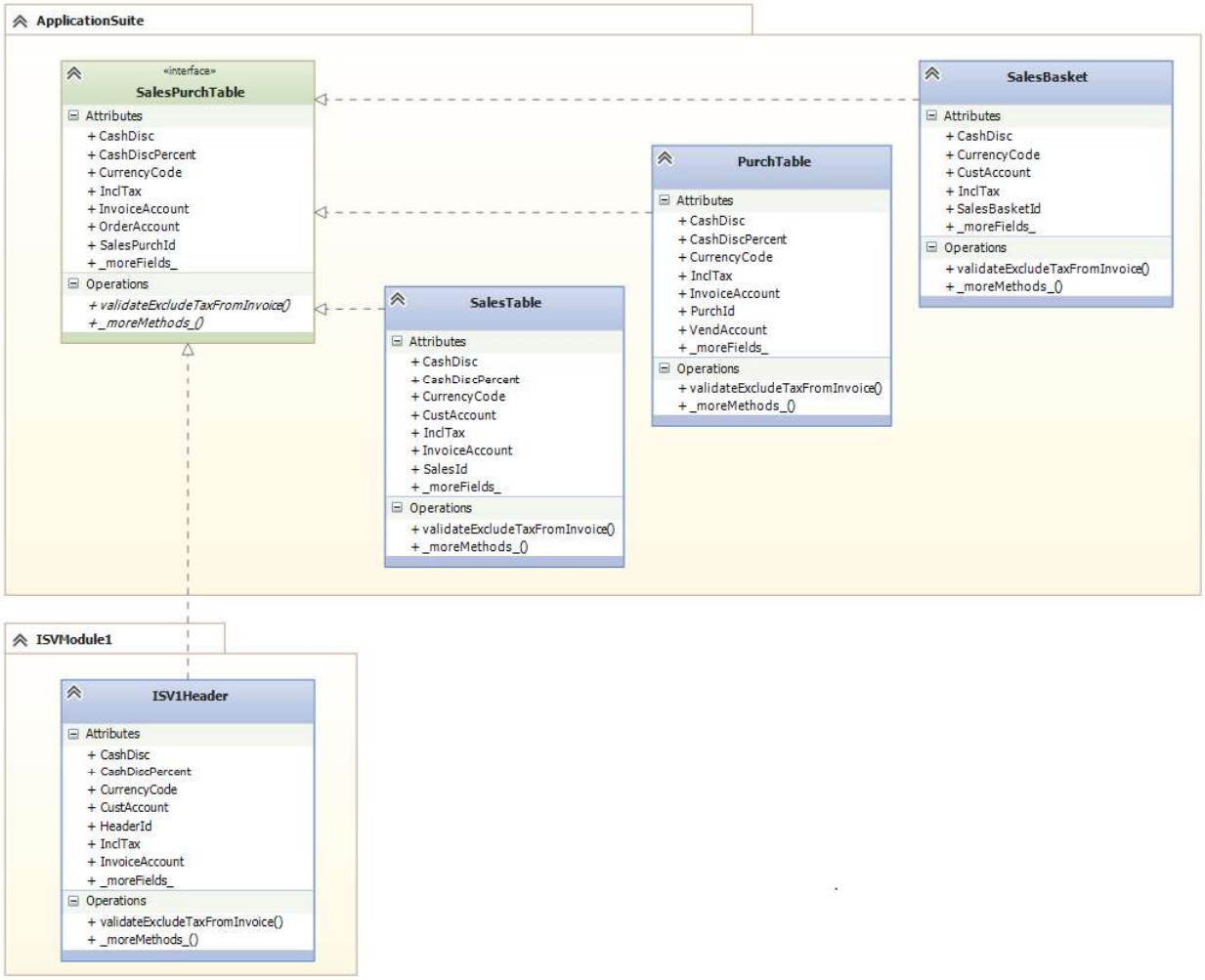
Table map extension

Article • 08/12/2022

To extend table maps, we have refactored table maps into a model, which allows you to extend a solution with additional fields and methods. This article discusses why you need a model to extend a table map.

Adding a field to an existing table map through extension can present some challenges. If these issues are not addressed during the implementation, there can be runtime errors. The errors occur because the developer cannot modify all the tables that are involved in implementing the table map. The same is true for adding a method to a table map if the method is called directly as an instance method on the table map. There is no way to enforce how fields on table maps must be mapped to fields on all tables that implement the table map. Similarly, there is no way to enforce how methods on table maps must also be methods on all tables that implement the table map.

The following diagram shows the **SalesPurchTable** table map, which is implemented by the **SalesTable**, **PurchTable**, and **SalesBasket** tables in the **ApplicationSuite** model. In addition, the **ISV1Header** table implements the **SalesPurchTable** table map, but **ISV1Header** is part of an **ISVModule1** model.



Extend table maps that are used as interfaces

Article • 08/12/2022

The **SalesPurchLine** and **SalesPurchTable** table maps expose a set of common fields and methods that are used by a variety of product features. The mapping of fields and the implementation of methods have been refactored into a class hierarchy. Some of these changes include:

- Methods on the table maps have been moved to the class hierarchy.
- Fields are exposed through parm-methods on the class hierarchy.
- The table map still exists and tables still implement the mapping to the table map, but the fields on the table map have been made obsolete, and field mapping has been removed.
- The methods on the table map have been made obsolete.

SalesPurchTableInterface hierarchy

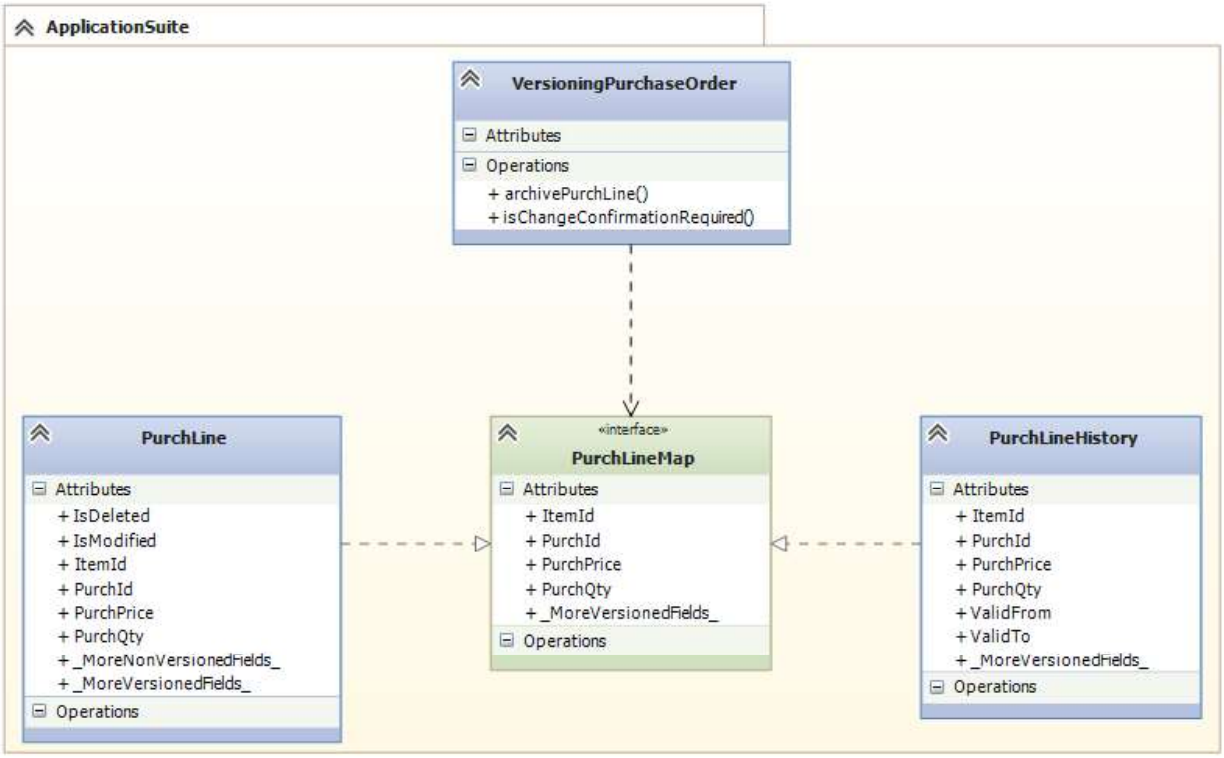
The new **SalesPurchTableInterface** class is the abstract base class for the ApplicationSuite functionality that has been introduced by the refactoring of the **SalesPurchTable** table map. The class contains abstract methods for fields and methods, which must exist for each table implementing the interface. It also contains the default logic in methods, which is common across all implementations. Each table that implements the **SalesPurchTable** table map must be represented as a derived class in the **SalesPurchTableInterface** class hierarchy. Each derived class must be decorated with a **SalesPurchTableInterfaceFactory** attribute class. The attribute is used to associate the derived class with the table, so that it is possible to instantiate a class of the correct type depending on a **SalesPurchTable** record.

Extend table maps that are used for versioning

Article • 08/12/2022

PurchLineMap table map logic

When new fields are added to the **PurchLine** and **PurchLineHistory** tables using table extensions, the new fields must be copied between the tables when a purchase order is versioned. The **PurchLineMap** table map specifies the fields that must be copied between the **PurchLine** table and the **PurchLineHistory** table when a new purchase order version is created or edited. To accomplish this, extend the **PurchLineMap** map table to include the additional fields. Additionally, the **PurchLineMap** is used by the **VersioningPurchaseOrder** class when archiving purchase order lines. The model is shown in the following diagram.



To be able to specify new fields to be copied, the **PurchLineMap** table map logic and its usage have been refactored. The copy logic has been moved to the **PurchLineVersioning** class, so the **VersioningPurchaseOrder** class references the **PurchLineVersioning** class instead of the **PurchLineMap** table map. The **PurchLineVersioning** class delegates the logic to copy the fields and the logic to determine whether a confirmation is required from the classes that implement the

Extending number of decimals for selected data types

Article • 08/12/2022

This article describes how to extend the number of decimals for selected data types. You can create extensions of specific extended data types of the type Real, to change the number of decimals for certain scenarios. To change the number of decimals, change the **NoOfDecimals** property as needed.

Extended data types are hierarchical and inherit behavior from the data type they extend. When changing the number of decimals for one extended data type, the number of decimals on all derived extended data types will follow. In other words, if you find an extended data type where **NoOfDecimalsIsExtensible** is false, then check the parent extended data type, as the number of decimals might be extensible in this wider scope.

> **Important**
>
> Due to database constraints, each of the data types described in this article can have a maximum precision of six decimals.

Weight

Weight data can be maintained with a maximum of two decimals by default.

If you require the ability to enter, maintain, and view weight data with a maximum of six decimals, you must extend the number of decimals for the **WeightBase** extended data type.

Product width, height and depth

These physical dimensions can be maintained with a maximum of two decimals by default.

If you require the ability to enter, maintain, and view this data with a maximum of six decimals, you must extend the number of decimals for the **InventWidth**, **InventHeight**, and **InventDepth** extended data types respectively.

Write extensible code

Article • 08/12/2022

X++ and the metadata model provide a powerful foundation for building business solutions. One of the design pillars is to automate as many technical concerns as possible, so that the engineer can focus on the business domain. For example, if you put all the text resources in label files, one technical concern that you don't have to worry about is the localization of text resources.

Extensibility is another technical concern. You want other people to be able to extend your solution in a safe, robust, and maintainable way. By default, your solution is highly extensible. However, there are a few guidelines that you should follow to help guarantee completeness.

Responsibilities

Any finance and operations environment runs a business solution that includes components from many sources. Typically, each solution has code from Microsoft, independent software vendors (ISVs) and partners, and also internally developed code. Each contributor is responsible for its own contribution to the solution and for the way that its contribution interacts with other contributions.

When you write extensible code, you invite other people to interact with your solution. Your responsibility is to enable other people to be good guests. Here is how you meet this responsibility:

- **Make robust extension points –** Extension points are the foundation of extenders' solutions. They must be well-defined and robust from release to release.
- **Invite side-by-side customizations –** Recognize that multiple extenders might use the same extension point. Enable 1:n (one-to-many) interactions instead of 1:1 (one-to-one) interactions.
- **Trust that extenders will be well-behaved –** All responsible parties share the same goal: to create great, lasting solutions for the customer. When you create extension points, you give up control and share the responsibility with other people. Assume that extenders will be cautious and use your extension points as they were intended.

Proven principles

Write extensible classes

Article • 08/12/2022

A class and its methods should have a single responsibility. Keep the following in mind in order to design classes that are resilient to changes in the long run. Class should have:

- A clear purpose
- Good names (class name and method names)
- Only methods that should be extended are exposed for extensibility, i.e. the key rule is allow extending as little as necessary.
  - Every public and protected method is an extensibility point in X++. Every time that a new method is introduced, downstream consumers get a new way to inject additional logic into the method.

Example

Non-extensible code

X++

```
    void calculatePrice(SalesLine _saleLine, AmountMST _amount)
    {
        // cannot add extra condition if needed
        if(_saleLine.QtyOrdered > 0 && _saleLine.SalesType ==
SalesType::Sales)
        {
            ttsbegin;
            // calculation of SalesPrice is locked and cannot be extended
            _saleLine.SalesPrice = _saleLine.QtyOrdered * _amount;
            _saleLine.update();
            ttscommit;
        }
    }
```

Extensible code

X++

```
 protected boolean canUpdateSalesPrice(SalesLine _saleLine)
    {
        return (_saleLine.QtyOrdered > 0 &&
    _saleLine.SalesType == SalesType::Sales);
    }

    protected SalesPrice calculateSalesPrice(
    SalesLine _saleLine, AmountMST _amount)
```

Write extensible methods

Article • 08/12/2022

Before you make a method extensible, you should assess the exposed functionality of the method and the impact that the extensions might have on the scenario where the method is used. For example, depending on the business scenario, there is low risk if you enable extensions to initialize a table record but high risk if you enable extensions to skip a specific validation. You might also want to consider the impact if the method is extended in parallel with other extensions.

After you've made a method extensible, future modifications to the method are restricted because of the potential user impact if the method signature or logic is changed.

Here are some guidelines to follow when you write extensible code:

- **Write short and concise methods –** A method should have only one responsibility. This approach enables easy extensions of the method, where the extensions can act only on the specific responsibility of the method. As a simple example, keep the construction and initialization of a class object in two separate methods.
- **Expose only what is necessary –** For any new class members or methods that are added, 'Keep any new class members or methods that you add private, to allow minimal access to them.
- **Use private, protected, public, and final explicitly –** For methods and class fields, this approach will guide any extenders of your code to your extension points but still let you keep full control of the parts that the extenders should not care about or depend on.
- **Method parameters**
  - The method is most likely long and should be refactored. Consider whether you should refactor the whole method into a class or split the method into smaller methods that require fewer parameters.
  - In other cases, when several parameters are required, the parameters often have a coherence that can be expressed by a class. By encapsulating these parameters in a class, you make it easy for extenders to add additional parameters to the base method, without breaking application programming interfaces (APIs) later.
- **Switch blocks**

Write extensible forms

Article • 08/12/2022

Methods on forms

- In general, the guidelines for writing extensible methods also apply to form methods.
- Chain of Command (CoC) gives access to the form's non-private members, which are the same as the non-private members for classes.
- CoC is enabled for nested classes. Therefore, methods that are defined in various levels on the form are extensible.

> **Note**
>
> One limitation of methods on forms, that for form data source methods only methods that are defined in the kernel are enabled for extensions, i.e. methods defined on the form data source are not extensible. This will be available in an upcoming Platform Update.

Field groups

Consider using field groups whenever possible. In this way, independent software vendors (ISVs) can add their fields for free when they extend a field group.

Form controls

Moving around form controls could potentially cause a break if the controls are made non-extensible by moving.

Feedback

Extended data types

Article • 08/12/2022

Extended data types (EDTs) have a rich extension model that lets extenders change specific behaviors.

To provide an extensible solution, keep the following guidelines in mind when you work with EDTs.

Label/Help text

Labels and Help text properties can be changed by an extension, but only one value can remain. If multiple solutions change the label of the same EDT, the various labels are, in functional terms, mutually exclusive. Therefore, those labels can't all be installed on the same system.

String size

String size can be defined only on root EDTs. The system will use the largest value that is defined across the EDT and its extensions.

For derived EDTs, string size can't be changed by an extension, because the IS-A relationship between the EDTs will be broken.

Assignments to string EDTs will truncate the string to match the defined string size.

Extends

The **extends** property can't be changed by an extension. Any change that is made to this property after release will cause a breaking change. Therefore, you must make sure that the property is set correctly before release.

If you set this property, neither you nor extenders will be able to make changes to the string size later.

Avoid unnecessary dependencies. For example, don't extend generic EDTs such as Name and Description.

Number of decimals

Write extensible enums

Article • 08/12/2022

An enumeration (enum) is made extensible by setting the following enum properties:

- Is Extensible = True
- UseEnumValue = No

If you set these properties, downstream implementors can extend the enum with more elements. The values of the elements are determined at deployment time and won't be identical across systems. However, the following behavior is ensured:

- Data upgrade scripts aren't required. Enum values are persisted in the database, regardless of the enum that is extended. Therefore, when an enum is made extensible, the enum values that are used on any system will prevail.
- The first element in the enum gets a value of 0 (zero). Therefore, an extensible enum can still be used with the **not** operator. The only exception is when the first element of the enum had a non-zero value before the enum was made extensible.

Using extensible enums in code

Because enum values are no longer controlled by the developer, there is no certainty about the enum values. When you use extensible enums in code, remember that extensible enums can't be used in comparisons. For example, **MyEnum::Value1 > MyEnum::Value2**.

Also, look for any conversions between integers and enums. For example, modeled ranges in views and queries, and queries that are created from code by using comparisons, such as < and > or by using hardcoded integer values in comparisons.

When the model and all dependent models are compiled, the comparisons and conversions to integers will be detected by the compiler as errors.

Make sure that logic where the enum values are used is extracted in **smaller methods**. In that way, an extension that uses Chain of Command (CoC) can handle the enum values that are added.

For **construct** methods where the instantiation is based on enum values, replace switch blocks with **SysExtension** wherever such a replacement is possible. In other cases, make sure that the default block is extensible. For an example, see the **PurchRFQCaseCopying** class.

Delegates

Article • 08/12/2022

Although you can subscribe to existing delegates, don't create new delegates. The Chain of Command (CoC) provides a richer, more robust, and more concise extension mechanism that supersedes delegates.

Instead of creating new delegates, structure your code in small methods that have good names, as described in the guidelines for writing extensible methods.

If you decide to use delegates, consider ensuring no more than one response where applicable. For more information, see EventHandlerResult classes in request or response scenarios.

Feedback

Was this page helpful? Yes No

Provide product feedback

Write extensible tables

Article • 08/12/2022

Tables have a rich extension model that lets extenders add fields, field groups, indexes, relations, methods, and more.

Unique indexes

Unique indexes can't be changed by an extension. Unique indexes define a table constraint, and they often also define the key of the rows in the tables. You aren't allowed to change unique indexes, because such changes change the nature of the table. Therefore, there is a high risk that the changes will cause logical conflicts with future versions of the solution that defines the table, or with other solutions that consume the table.

Avoid unique indexes that are likely to be changed either now or in the future. For example, don't create a unique index on product dimensions such as color, size, style, and configuration. Instead, create a unique index on a distinct product variant, so that the index doesn't have to be changed if new product dimensions are added.

If you require an extensible uniqueness constraint on multiple columns, consider creating a hash of the column's values. For an example, see the NumberSequenceScope table.

Data events

Tables have many predefined data events that are automatically raised.

Avoid calling **doInsert()**, **doUpdate()**, and **doDelete()**. These methods prevent the data events from being raised and make your table harder to extend. Instead, call **insert()**, **update()**, and **delete()**.

Field groups

Always use field groups to group related fields, and to build forms and reports. By consistently using this approach, you enable the extension to surface additional fields in forms and on reports by extending the field group.

Attributes that make methods extensible

Article • 08/12/2022

This article describes the various attributes that can be used to control extensibility capabilities for methods.

The following table provides an overview of the default support for extensibility and accessibility on methods. The table also provides guidance on the method signature changes.

Expand table

Attribute	Hookable	Wrappable	Replaceable	Accessibility	Signature
private	No	N/A	N/A	Accessible from within class it is defined in.	Signature can be changed
protected internal	No	Yes, from Platform update 25 onward	No	Accessible from with the class it is defined, from derived classes, and from classes in the same model.	Signature must remain compatible
internal	No	No	No	Accessible in the same model.	Signature can be changed
protected	No	Yes, unless marked final	No	Accessible from with the class it is defined and from derived classes.	Signature must remain compatible
public	Yes	Yes, unless marked final	No	Accessible from within the class it is defined, derived classes, and other classes that have access to the defining class.	Signature must remain compatible

Hookable

If a method is hookable, extenders can subscribe to pre-events and post-events.

Breaking changes

Article • 08/12/2022

When you make your solution extensible, you also help guarantee that you won't break the extension points later. A breaking change is any change that can break a consumer of your code.

This article lists some of the types of changes that can break your code.

> **Important**
>
> This list isn't exhaustive. Other types of changes that aren't listed here could also be breaking changes.

Data model changes

General

If any data model change requires that a data upgrade script be run, consumers might no longer be able to synchronize their data model, or they might lose access to data.

Data types

- **Changing an enumeration (enum) from extensible to non-extensible –** Consumers might have extensions to the enum.
- **Changing an enum from non-extensible to extensible –** Consumers might be using the enums in comparisons. For more details, see Write extensible enums.
- **Decreasing the decimal precision of an extended data type (EDT) of the real type –** Consumers might have dependencies on the ability to enter data that uses the precision.
- **Decreasing the size of an EDT of the string type –** Consumers might have dependencies on the size of the string.
- **Specializing the EDT by making it extend another EDT –** Consumers might have string length or decimal precision extensions to the EDT.
- **Changing the enum type of an EDT of the enum type when the enum is extensible –** Consumers might have extensions to the enum.

Data model

Compatibility checker tool

Article • 08/12/2022

The compatibility checker tool can detect metadata breaking changes against a specified baseline release or update. In this way, it helps ensure backward compatibility. Microsoft uses the tool to help ensure metadata compatibility.

The compatibility checker tool is available as one of the dev tools in Platform update 34. You can use it to ensure that your solutions are backward-compatible with earlier releases before you install or push updates to customers.

What the tool detects

The tool compares metadata of the current version with metadata of a baseline version. It detects and reports metadata changes that Microsoft has identified as breaking and added to the tool.

For a list of breaking changes that the tool detects, see the List of breaking changes detected by the tool section later in this article.

> **Note**
>
> - The list in this article doesn't include all the breaking changes that the tool can detect.
> - The tool doesn't detect all breaking changes.

What the tool doesn't detect

The tool detects only breaking changes that can be identified by comparing data. For example, it doesn't detect the following breaking changes that often occur:

- The reference to a protected or public method is removed.
- The responsibility of a method is changed.

Using the tool

You can use the tool to detect metadata compatibility issues that a new version has against the version that it's replacing. Microsoft uses the tool to detect any breaking changes that a new monthly update has against the previous monthly update.

Capture a trace

Article • 12/01/2023

This tutorial provides guidelines for taking traces in Microsoft Dynamics 365 finance and operations apps.

In this tutorial, you take a tour of the process of collecting and downloading traces. The Trace parser tool works much like the Microsoft Dynamics AX 2012 version, but it isn't backward compatible and can't be used to analyze AX 2012 traces. You can find the tool in your development environments. On the Windows **Start** menu, select **Microsoft Dynamics Trace Parser**. Alternatively, open **C:\Program Files (x86)\Microsoft Dynamics Trace Parser\Microsoft.Dynamics.AX.Tracing.TraceParser.exe**.

Prerequisites

This tutorial requires that you access the environment as an administrator on the instance. The administrator can also grant rights to other users to take a trace. In this way, you can trace scenarios that can't be reproduced with administrative rights.

Capture a trace - Client

1. On the navigation bar, select **Help**, and then select **Trace**.
2. In the **Trace name** field, enter a name for the trace that you're about to capture.
3. As required, set the **Include SQL parameter value** option to **Yes**.
4. Select **Start trace**.
5. Perform actions that must be analyzed, such as navigation to **Accounts payable** > **Vendors** > **All vendors**.
6. When you've finished, select **Stop trace**.
7. After the trace has stopped, select one of the following options. (For this tutorial, select the second option.)

    - **Download trace –** Store the captured trace on a local machine. You can analyze a downloaded trace by using the desktop version of Trace parser.

      > ⓘ **Note**

Single-user testing using Performance SDK and Task recorder

Article • 08/12/2022

Use the information in this article to do single-user testing by using Visual Studio and the Performance software development kit (SDK) together with a performance test script that is generated by using Task recorder.

> **Important**
>
> Visual Studio 2019 will be the last version of Visual Studio with web performance and load test features.
>
> - If you are using the Visual Studio and Test Controller/Test Agent for on-premises load testing, Visual Studio 2019 will be the last version. You can continue using it until the end of the support cycle.
> - For more information, see Cloud-based load testing service end of life.

Use Task recorder to define and record an end-to-end business scenario

Before you run a single-user test, you must work with your business team to define your end-to-end scenarios and then use Task recorder to create a recording of the steps in each scenario. For more information about how to create a task recording, see Task recorder resources. The scenarios that you should test depend on your customer's business requirements. In this article, you will use the "Create and confirm a sales order" sample scenario.

1. Sign in as a Sales persona.
2. Turn on Task recorder, and create and confirm a sales order that includes the following information:
    - Customer account
    - Item number
    - Sales quantity
    - Site
    - Warehouse

Run multi-user testing by using the Performance SDK

Article • 03/27/2024

This article explains how to run multi-user testing by using Microsoft Visual Studio, the Performance software development kit (SDK), and the Task recorder test scripts.

> **Important**
>
> Visual Studio 2019 is the last version of Visual Studio that includes web performance and load testing features and will be deprecated in the future. We suggest using Apache JMeter for performance testing. For more information, see JMeter.
>
> JMeter is owned by Apache, and therefore isn't supported by Microsoft. However, the following series of blog posts provides some helpful tips for using JMeter with Dynamics 365 finance and operations apps.
>
> - Part 1 - Dynamics 365 finance and operations apps performance testing with JMeter - Introduction
> - Part 2 - Dynamics 365 finance and operations apps performance testing with JMeter - Execution
> - Part 3 – Dynamics 365 finance and operations apps performance testing with JMeter - Result Analysis
> - Part 4 – Dynamics 365 finance and operations apps performance testing with JMeter - Tips and Troubleshooting

Prerequisites

Before you complete the steps in this article, verify that the following prerequisites are met:

- You have **Visual Studio Enterprise edition** in a development environment. Enterprise edition is required to create load tests. If you're deploying your development box as a cloud-hosted environment through Microsoft Dynamics Lifecycle Services (LCS), be sure to select the appropriate Visual Studio version to deploy.
- The Visual Studio web performance and load testing tools are installed as described in Install the load testing component.
- You have a tier-2 or higher sandbox environment that has the same release (application version and platform update) as your development environment.
- You've configured your development environment by following the steps in Single-user testing with Task recorder and the Performance SDK.
- C# performance testing classes have been generated for your end-to-end (E2E) scenarios, and you can run a single-user test by following the steps in Single-user testing with Task recorder and the Performance SDK.

Troubleshooting guide for testing with the Performance SDK

Article • 08/12/2022

No client was opened in the time-out period

This issue affects only single-user tests. When the test is running, a web client is opened, but a website is never loaded. Instead, there is an empty web client that has a white background. The following message appears at the top of the page, "This is the initial start page for the WebDriver server." The test eventually times out and fails, and an error message is shown.

Error - No client was opened in the time-out period

> Initialization method <Test class name>.TestSetup threw an exception. System.TimeoutException: System.TimeoutException: No client was opened in the timeout period.

Solution

See Multi-user testing using the Performance SDK. That article explains how to create a correct certificate for this type of test. It also explains how to add the thumbprint of the certificate to the wif.config file.

Zoom factor

This issue affects only single-user tests.

Error - Zoom factor

> Initialization method <Test class name>.TestSetup threw exception. System.InvalidOperationException: System.InvalidOperationException: Unexpected error launching Internet Explorer. Browser zoom level was set to 200%. It should be set to 100% (NoSuchDriver).

Solution - Zoom factor

In Internet Explorer, you can change the zoom factor to 100 percent by changing the following registry keys:

- Computer\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Zoom\ResetZoomOnStartup = 0
- Computer\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Zoom\ResetZoomOnStartup2 = 0
- Computer\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Zoom\Zoomfactor = 80000

Depending on the version of the local machine that is used, before you start the Remote Desktop Protocol (RDP) session, you might have to select **Change the size of text, apps and other items**. This field is available in **Display settings** in Microsoft Windows.

If those steps don't work, change the size of your remote desktop before you start the RDP session, so that the default zoom level in Internet Explorer is 100 percent.

Certificate thumbprint errors

Error example - Certificate thumbprint errors

> Initialization method MS.Dynamics.Performance.Application.TaskRecorder.TestRecord1Base.TestSetup** threw an exception. System.TypeInitializationException: System.TypeInitializationException: The type initializer for 'MS.Dynamics.TestTools.CloudCommonTestUtilities.Authentication.UserManagement' threw an exception. --> MS.Dynamics.TestTools.CloudCommonTestUtilities.Exceptions.WebAuthenticationException: Failed finding the certificate for minting tokens by thumbprint: b4f01d2fc42718198852cd23957fc60a3e4bca2e.

Diagnose issues and analyze performance by using Trace parser

Article • 08/12/2022

This article explains how you can use the Trace parser to consume traces and analyze performance in your deployment. You can use the Trace Parser to find and diagnose various types of errors. You can also use the tool to visualize execution of X++ methods, as well as the execution call tree.

> ⓘ **Note**
>
> There are features in the Trace parser that are similar to Microsoft Dynamics AX 2012. See the Dynamics Ax Performance Team Blog for more information.

Finding the Trace parser

Trace parser should be preinstalled with your developer deployment or VHD. The install location is here: **C:\Program Files (x86)\Microsoft Dynamics Trace Parser**. If it's not installed, you can run the installer from **C:\PerfSDK\PerfTools\traceparser.msi**.

Capturing events

There are two ways that you can obtain the data that you will analyze in the Trace parser. They include:

- Capture events from the local installation.
  - If the **Select Trace** window isn't already open, go to the **File** menu and click **Open trace**. In the **Select Trace** window, click **Capture Events**. After selecting your providers, click **Start**. The Trace Parser tool will start listening to all the providers and capturing the events. Capturing stops when you click **Stop and Import**.
- Open an existing ETL (Windows Event) file that was captured using tools such as Logman.

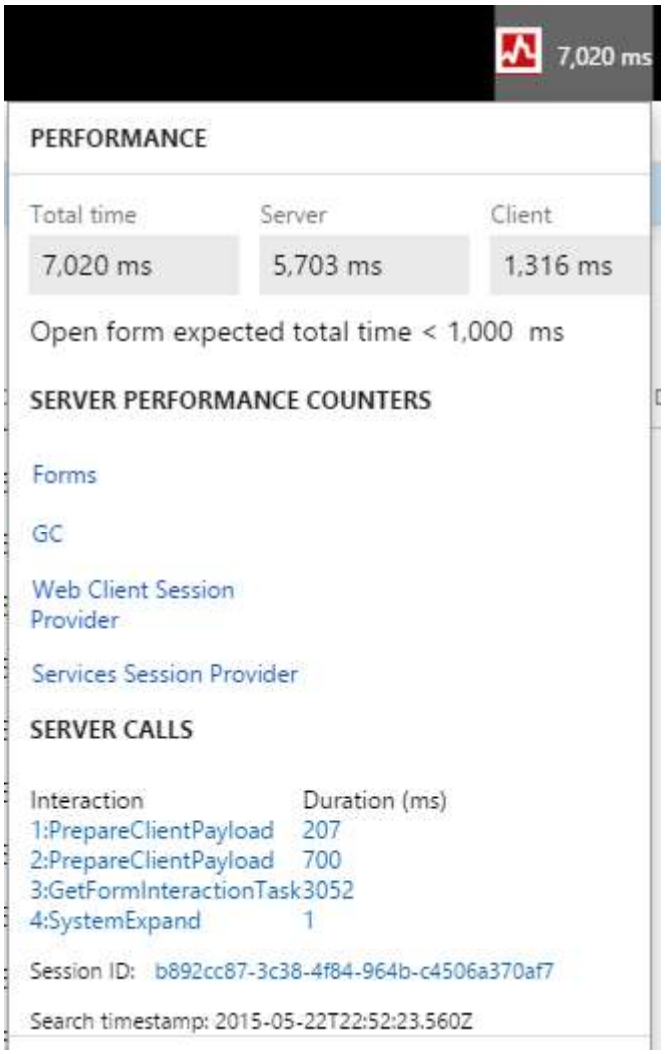
Performance timer

Article • 08/12/2022

This article provides an overview of the Performance timer, which is a tool that helps you to determine why your system's performance might be slow.

To open the Performance timer, open your webpage with the added parameter debug=develop: https://*yoursite*.cloud.test.dynamics.com/en/?cmp=USMF&debug=develop Note: When you run in debug mode you will notice slower performance. You can quickly get an overview of most performance issues by pressing F12 and working with the debugging tools that are available in your browser. The timer will show up here.

To open a list page, for example, such as the purchase order list page, click the Performance timer. The following screenshot shows the separation between client time and server time, and the total time. Additionally, you can see a set of performance counters and expensive server calls.



Batch parallelism and multi-threading in Dynamics 365 finance and operations apps

Article • 06/10/2024

This article describes the different approaches that are available for breaking large batch jobs into small fragments in Microsoft Dynamics 365 finance and operations apps.

Three approaches can be used to break up large batch jobs:

- Individual task modeling
- Batch bundling
- Top picking pattern

Each approach has pros and cons, as described in the following table.

Expand table

Approach	Pros	Cons
Individual task modeling	• It works well with a few work items of any type. • It's simple to write. • It's the best fit for creating dependency among the work items.	• When there's a large number of tasks, the extra overhead impedes performance because of the batch throttling mechanism and delays between batch task runs. • It negatively affects other batch jobs.
Batch bundling	• It works well with a simple, even workload. • There's no need for a staging table and no extra maintenance by the application code. • It doesn't over-pollute the batch table. • It bypasses the batch throttling mechanism when the appropriate bundle size is selected.	• When the work isn't evenly distributed, performance degrades because batch tasks finish processing at different times. • In some cases, it's possible to distribute the workload evenly.
Top picking pattern	• It works well with an uneven workload.	• An extra staging table is needed to track the progress.

Testing and validations

Article • 08/12/2022

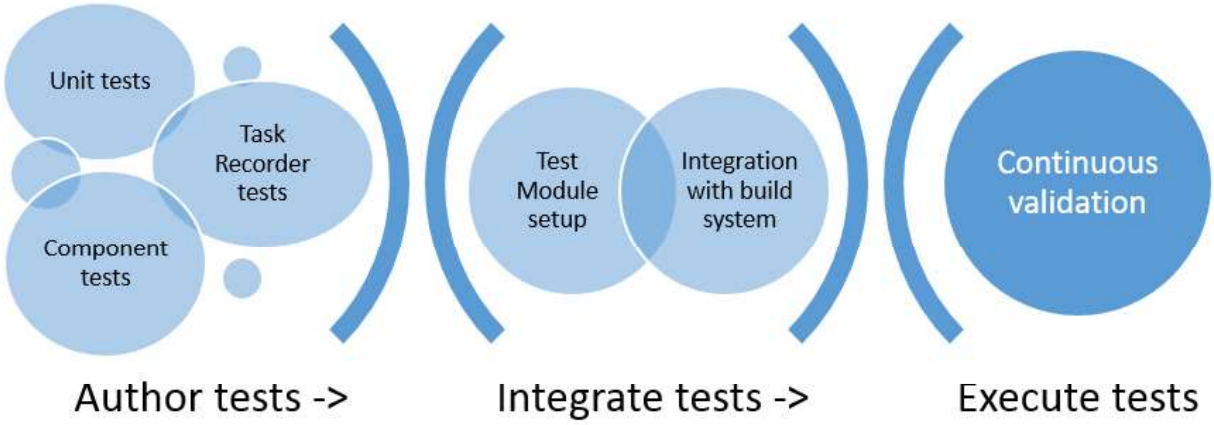
This tutorial shows you how to create and run test cases.

Prerequisites

You need to deploy Developer Topology with Developer and Build VM.

Key concepts

- Use SysTest Framework to author unit/component test code.
- Test isolation
- Test module creation to manage test code and FormAdaptors.
- Import Task Recorder recordings into Visual Studio to generate test code.
- Integrate a Test module with a build machine.



Use SysTest Framework to author unit/component test code

You can create new test cases to test the functionality in an application.

1. Open Visual Studio as an administrator.

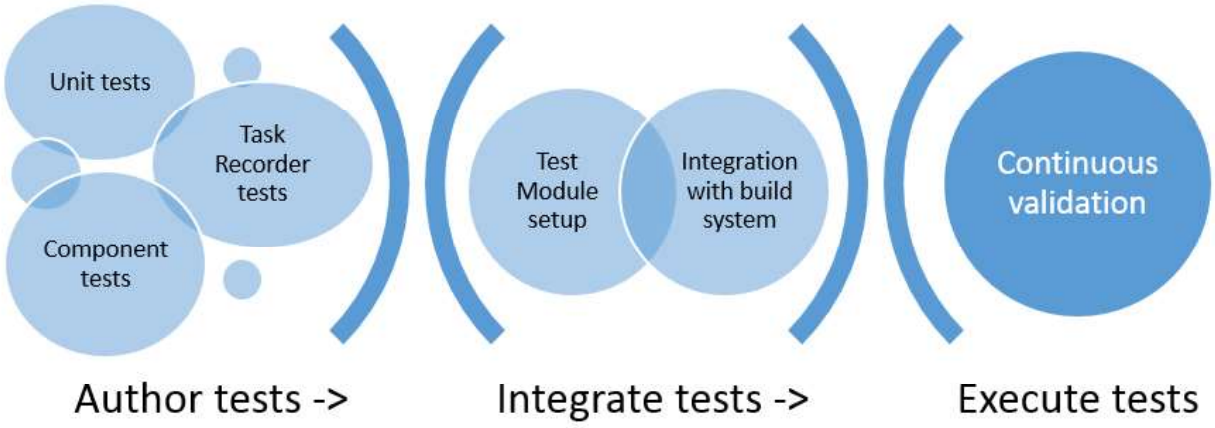
2. On the **File** menu, click **Open** > **Project/Solution**, and then select **FleetManagement solution** from the desktop folder. If the solution file is not on your computer, the steps to create it are listed in End-to-end scenario for the Fleet Management sample application.

Test projects in Visual Studio

Article • 08/12/2022

This article describes the options for testing in Visual Studio.

A custom unit test adapter is available in Visual Studio. This adapter lets test authors use the standard **Test Explorer** window in Visual Studio to schedule X++ tests and analyze test results. Developers can author tests by using **SysTestAdaptor**. They can also generate test code from Task Recorder recordings. These test cases can then be added to build systems for validations.



Author unit/component test code by using the SysTest Framework

When you create a project in Visual Studio, you can add an X++ unit test. You extend the class with **SysTestCase**, and then either add the **SysTestMethodAttribute** attribute or prefix the case with "test" in the method name.

X++

```
class FMUnitTestSample extends SysTestCase
{
    [SysTestMethod]
    public void testTotalsEngineConfig()
    {
    }
}
```

After you save the class, each test appears in Test Explorer, just as a C# test would appear.

Deploy and use a continuous build and test automation environment

Article • 10/31/2022

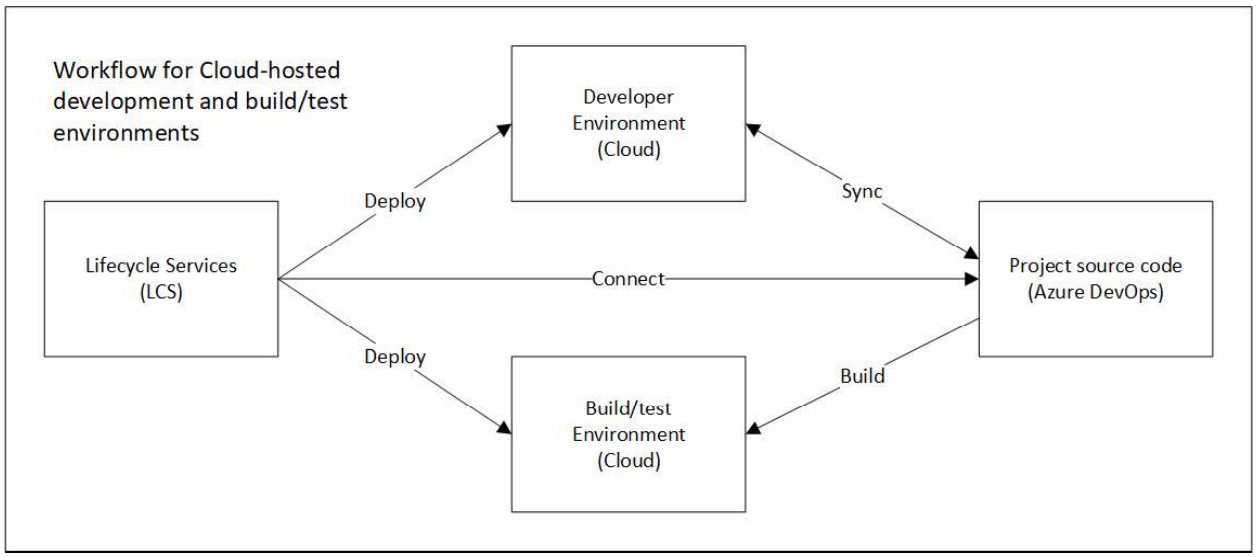
This article describes how to deploy and use an environment that supports continuous build and test automation.

Prerequisites

Cloud deployment of virtual machines (VMs) requires a Microsoft Azure DevOps subscription.

Workflow

After you configure an Azure DevOps subscription in Microsoft Dynamics Lifecycle Services (LCS), you can use LCS to deploy developer VMs or build/test VMs. LCS configures a developer VM that can be mapped to an Azure DevOps project. LCS also configures a build VM that is automatically mapped to an Azure DevOps project and has a build agent/controller that builds modules from the Azure DevOps project and runs automated tests that have an external endpoint for validation. The following illustration shows a typical workflow.



This workflow includes an LCS deployment of a developer VM and a build/test VM in Azure.

- LCS creates developer and the build/test environments in Azure. To create a build/test environment, LCS must be able to determine where the source code for

SysTest Filtering using class and method attributes

Article • 08/12/2022

Starting with Platform update 12, the SysTest framework contains improvements to the SysTest class and method attributes in X++. These improvements also change how these attributes work in the Visual Studio test window as well as the Visual Studio Test Console, which is the tool used in the automated build process. The SysTest framework now supports the major test attributes in the adaptor to be on par with the MSTest framework adaptor. This includes attributes like **Category**, **Owner**, **Priority**, and **Test Property**.

TestCategory

The **Category** attribute, **SysTestCategory**, was already available in previous platform updates using the **SysTestCategory** attribute. Starting with Platform update 12, you can specify multiple categories on both the class level and the individual method level. Additionally, **TestCategory** is enabled for filtering in the Visual Studio Test Console. This means that you can create build pipelines with test filters on specific categories. You can use the **TestFilter** variable in the build pipeline. For example, to run tests only with a category **Nightly**, set the variable to **TestCategory=Nightly**.

Priority

The **Priority** attribute **SysTestPriority**, which requires an integer value, is now available. A priority can only be specified once, but is supported on both the class and method level, with method level taking precedence over class level. The priority is also exposed as a test filter. This means that you can use the **TestFilter** variable in the build pipeline. For example, to only run tests with a priority of **1**, set the variable to **Priority=1**.

Owner

The **Owner** attribute, **SysTestOwner**, has also been added. This attribute was technically already supported for filtering in the **Test Toolbox** window, but the attribute itself was missing in X++. Similar to **Priority**, an owner can only be specified once and is supported on both the class and method level, with the method level taking precedence. **Owner** is not available as a test filter for the console, which aligns with the

SysTestRow attribute for testing multiple values

Article • 08/12/2022

Sometimes, tests must test multiple input values for the same feature. To make troubleshooting and reporting easier, you should avoid testing multiple things in the same test method. Instead of creating multiple methods, you can use the **SysTestRow** attribute that the SysTest framework now supports. This attribute works like the **DataRow** attribute in C#.

> ⓘ **Note**
>
> The **SysTestRow** attribute is available in version 10.0.25 and later.

SysTestRow example

In this example, you have a method that squares a number.

X++

```
public class MyMath
{
    public static int square(int _i)
    {
        return _i * _i;
    }
}
```

You want to test this method by using a few different numbers. Among these numbers, you want to include special cases that should work. Therefore, you write the following test method that uses the **SysTestRow** attribute to test multiple values in the same method.

X++

```
public class MathTests extends SysTestCase
{
    [SysTestMethod,
    SysTestRow(0,0),
    SysTestRow(1,1),
    SysTestRow(-1, 1),
    SysTestRow(4, 16),
```

Acceptance test library resources

Article • 08/12/2022

The Acceptance test library (ATL) is an X++ test library that offers the following benefits:

- It lets you create consistent test data.
- It increases the readability of test code.
- It provides improved discoverability of the methods that are used to create test data.
- It hides the complexity of setting up prerequisites.
- It supports high performance of test cases.

Example of a test that is written in ATL

X++

```
// Create the data root node
var data = AtlDataRootNode::construct();

// Get a reference to a well-known warehouse
var warehouse = data.invent().warehouses().default();

// Create a new item with the "default" setup using the item creator class.
Adjust the default warehouse before saving the item.
var item = items.defaultBuilder().setDefaultWarehouse(warehouse).create();

// Add on-hand (information about availability of the item in the warehouse)
by using the on-hand adjustment command.
onHand.adjust().forItem(item).forInventDims([warehouse]).setQty(100).execute
();

// Create a sales order with one line using the sales order entity
var salesOrder = data.sales().salesOrders().createDefault();
var salesLine = salesOrder.addLine().setItem(item).setQuantity(10).save();

// Reserve 3 units of the item using the reserve() command that is exposed
directly on the sales line entity
salesLine.reserve().setQty(3).execute();

// Verify inventory transactions that are associated with the sales line
using the inventoryTransactions query and specifications
salesLine.inventoryTransactions().assertExpectedLines(

invent.trans().spec().withStatusIssue(StatusIssue::OnOrder).withInventDims([
warehouse]).withQty(-7),
```

Navigation concepts for test data

Article • 08/12/2022

To simplify the discoverability of generation methods for test data, a set of navigation objects is introduced. For more information about the generation methods, see Test data methods.

Navigation should start from the root object, the module must be specified, and then the entity must be specified together with the test data methods.

```
X++

data.module().entity().testDataMethod();
```

Examples

```
X++

modelGroup = data.invent().modelGroups().fifo();

itemBuilder = data.products().items().whsBuilder();

data.sales().salesOrders().ensureCanCreate();

data.invent().parameters().enableQualityManagement();
```

Advantages

- Discoverability of data creation application programming interfaces (APIs). IntelliSense helps you get to the helper methods that are relevant for the entity.
- Local references to frequently used nodes (for example, `items.whsBuilder()`). Therefore, long names aren't required.

Root navigation object

The root navigation object is the starting point for finding the test data methods that are required. The root navigation object exposes modules where test data methods are defined.

Variable naming

Test data methods

Article • 08/12/2022

Entity and helper navigation objects expose test methods that let you set up test data. This article provides information about the most common types of test data methods.

Factory methods

Factory methods focus on creating data that doesn't yet exist in the database. There are two types of entity factory methods, `init` methods and `create` methods. An `init` method initializes the entity but doesn't save it to the database. A `create` method initializes the entity and saves it to the database.

Naming convention

`init<EntitySpecification>`

`create<EntitySpecification>`

In this naming convention, `<EntitySpecification>` is the description of the key characteristics of the object that must be created.

Examples

X++

```
salesOrder = data.sales().salesOrders().initDefault();

purchaseOrder = data.purch().purchaseOrders().createDefault();
```

Best practices

The `create` method should always call the `init` method that has the same entity specification.

Example

X++

Entities in the Acceptance test library

Article • 08/12/2022

A test entity class represents data and behavior that are perceived as a single concept. Test entity classes are based on pages such as **Sales order**, **Transfer order**, and **Released product**. The test entity classes expose the properties that are most often used in test scenarios, and the behavior that is most important from the perspective of test data setup and scenario tests.

An entity in the Acceptance test library (ATL) **must** have the following methods:

- Property methods that are used to get and set entity properties.
- Fluent setter methods that enable entity properties to be set in a fluent manner.
- A method that saves the entity to the database.

An entity in ATL *can* have the following methods:

- Action methods that are used to expose business operations that are relevant to the entity.
- Query methods that are used to enable navigation to components and related entities.

Naming convention

`[AtlEntity]<ModuleName><EntityName>`

In this naming convention:

- `<ModuleName>` is based on the names of the modules in main menu. You should use a short version or an abbreviation to support brevity of test code.
- `<EntityName>` is based on the user interface (UI) names instead of the table names. For example, use `SalesOrder`, not `SalesTable`.

If an entity has two UI names, it's OK to use the shorter name. For example, you can use `Item` instead of `ReleasedProduct`, because these names are used interchangeably.

Examples

```
X++
```

```
AtlEntitySalesOrder
```

Acceptance test library commands

Article • 08/12/2022

Command classes are responsible for running business operations. They let you use fluent application programming interfaces (APIs) to set the parameters of these operations.

Naming convention

`AtlCommand<ModuleName><EntityName><ExecuteBusinessOperation>`

In this naming convention:

- `<ModuleName>` is based on the names of the modules on the main menu. You should use a short version or an abbreviation to support brevity of test code.
- `<EntityName>` is optional and is used when the command applies to different types of entities.
- `<ExecuteBusinessOperation>` is a verb that represents the name of the business operation.

Examples

```
X++
```

```
AtlCommandInventMark

AtlCommandSalesReturnOrderLineRegister
```

Implementation

Command objects that are returned should implement the `AtlICommand` interface and should inherit from the `AtlCommand` class.

Command objects should provide fluent setter methods that are used to set the parameters of the command.

Example

```
X++
```

Creators in the Acceptance test library

Article • 08/12/2022

Creator classes provide fluent application programming interfaces (APIs) that are used to create test data.

Naming convention

`AtlCreator<ModuleName><EntityName>`

In this naming convention:

- `<ModuleName>` is optional and is based on the names of the modules on the main menu. You should use a short version or an abbreviation to support brevity of test code.
- `<EntityName>` is optional and is used when the command applies to different types of entities.

Examples

```
X++

AtlCreatorCostGroup

AtlCreatorCustomer
```

Fluent setter methods

Creator classes should provide fluent setter methods that are used to set the properties of the entity that is being constructed.

Naming convention

`set<EntityPropertyName>`

In this naming convention, `<EntityPropertyName>` is the name of the property that is being set for the entity by using the fluent method.

Example

Queries in the Acceptance test library

Article • 08/12/2022

A query class provides fluent application programming interfaces (APIs) that are used to find an instance of the corresponding entity, based on various criteria. Query classes are often used in validation scenarios. They are usually used together with specifications.

Naming convention

`AtlQuery<ModuleName><EntityNamePlural>`

In this naming convention:

- `<ModuleName>` is optional and is based on the names of the modules on the main menu. However, a short version or an abbreviation should be used to support brevity of test code.
- `<EntityNamePlural>` is the plural version of the entity name.

Examples

```
X++
```

```
AtlQueryWHSLoadLines

AtlQueryInventTransferOrderLines
```

Implementation

Query classes inherit from the `AtlQuery` class that is common to all queries.

Fluent setters

Query classes should provide fluent setter methods to specify ranges for the query.

Example

```
X++
```

Public application tests

Article • 04/24/2023

This article describes a collection of tests that cover the standard functionality of finance and operations apps. The following table lists the tests that are available.

Expand table

Name	Description
Acceptance Test Library - Test Case Common (`AtlTestCaseCommon`)	A collection of abstract classes that make it easier to create tests that are focused on a specific area.
Warehouse Public Tests (`WHSTests`)	A collection of scenario and performance tests for Warehouse Management.

The tests serve the following purposes:

- Enable engineers to detect eventual regressions early. For example, they can identify an extension that breaks a standard scenario.
- Serve as inspiration for the creation of test coverage for your own scenarios. For example, you can find a test that covers a similar scenario, copy it, and modify it to cover your variant of the scenario.

Run the tests

You'll typically use one of the following methods to run tests:

- **Run related tests in Visual Studio.** If you select and hold (or right-click) a project in Microsoft Visual Studio and then select **Discover tests**, the system automatically finds any test that's related to the other files in your project, and adds it to your project. This approach makes it easy to quickly run the most important tests by using Test Explorer.
- **Run all tests.** The easiest approach is to add all tests from a model to a new project and then use Test Explorer to run the tests. It can take a long time to run the tests.
- **Integrate test automation in your build pipeline.** By running tests as often as possible, you help minimizes the time that's required to detect a regression. Ideally, a pull request that contains a regression will be rejected.

For more information about how to configure build pipelines, see Deploy and use a continuous build and test automation environment.

Specification classes

Article • 08/12/2022

A specification class provides fluent application programming interfaces (APIs) that are used to define the set of criteria that an entity should meet. Specifications are often used in validation scenarios. They are usually used together with query classes.

An advantage of specification classes is that the validation code becomes very concise and expressive. Basically, you can do multiple validations in a single line of code.

Naming convention

`AtlSpec<ModuleName><#EntityName>`

In this naming convention:

- `<ModuleName>` is optional and is based on the names of the modules on the main menu. However, a short version or an abbreviation should be used to support brevity of test code.
- `<#EntityName>` represents the name of the entity that is used throughout the Acceptance test library (ATL).

Examples

Console

```
AtlSpecWHSLoadLine

AtlSpecWHSWorkLine
```

Implementation

Specification classes should provide fluent setter methods to specify various criteria of the specification.

Example

The following code verifies that the work contains six lines that meet the specified criteria. For example, the first line should have **1** as the line number of **1**, **Pick** as the work

Acceptance test library Code generation wizard

Article • 08/12/2022

The Acceptance test library (ATL) code generator quickly generates and updates new ATL entities, queries, and specifications, based on tables and data entities.

Create the AtlEntity class by using the wizard

Follow these steps to create the `AtlEntity` class by using the **Code generation** wizard.

1. In Microsoft Visual Studio, open the table in the designer window.
2. Right-click the name of the table, and then, on the **Add-ins** menu, select **Generate ATL Entity**.
3. Select the fields that should be included in the `AtlEntity` class, and then select **Add**.
4. Rename the entity and the fields as you require.
5. Select **Generate** to create the class.

Additional optional steps

When you create the `AtlEntity` class, you can also complete these tasks:

- Add required actions for the scenario.
- Add a `default` method to `AtlData` classes.
- Override the `setMainRecordField` method to call the `modifiedField(_fieldId)` method on the table.

```
X++
```

```
protected void setMainRecordField(FieldId _fieldId, anytype _value)
{
    super(_fieldId, _value);
    common.modifiedField(_fieldId);
}
```

Create the AtlQuery class by using the wizard

Best practices for the Acceptance test library

Article • 08/12/2022

Use var and declare variables inline

- Use the `var` keyword (type inference).
- Declare variables inline instead of in a separate statement.

Do this

```
X++
```

```
var item = items.default();
var salesOrder = data.sales().salesOrders().createDefault();
var salesLine =
salesOrder.addLine().setItem(item).setInventDims([warehouse]).setQuantity(10
).save();
```

Don't do this

```
X++
```

```
InventTable item;
AtlEntitySalesOrder salesOrder;
AtlEntitySaleOrderLine salesLine;
…
item = items.default();
salesOrder = data.sales().salesOrders().createDefault();
salesLine =
salesOrder.addLine().setItem(item).setInventDims([warehouse]).setQuantity(10
).save();
```

Justification

The advantages of using `var` are that you write less code, you don't have to remember exact type names, and the test logic isn't cluttered with unimportant information. Overall, the test code easier to read.

Acceptance test library FAQ

Article • 08/12/2022

Which fluent prefix should I use: set, for, or with?

Depending on the class that you want to add a fluent method to, different rules might apply:

- Entities
- Creators
- Commands
- Queries
- Specifications

Should I implement an entity or a creator class?

For most entities, the effort of creating an entity class is the same as the effort of creating a creator class. Therefore, the entity class should be created. However, in some cases, the process of creating an entity might not be straightforward. A good example is the Item entity. Because more than ten different tables make up the Item entity, it's hard to create the entity class. Because you will almost never have to update existing items in your test cases, it's OK if you just have a creator class, which is much easier to implement.

Does the order of the chained fluent setters matter?

In most of the cases, the order of the chained fluent setters doesn't matter. However, be aware that defaulting occurs at the time of the call to the setter method. Therefore, a change in the order of the methods can produce different results. Here is an example for the sales line.

Option 1

```
X++
```

Date effectivity

Article • 08/12/2022

This article provides information about date-effective data entities and data sources, and shows how to create a date-effective entity. It also explains how date effectivity applies to read and write activities.

There are different design patterns for date-effective features that involve data entities. The patterns are classified into two main categories:

- **Date-effective entities –** The entity has at least one date-effective data source, and the entity itself is also date effective.
- **Non-date-effective entities –** The entity itself is not date effective, but it does contain date-effective data sources.

The next sections describe the small list of properties and methods that control the date-effective behavior of entities and their date-effective data sources.

Date-effective entities

The following table describes the properties that control the date-effective behavior of a data entity.

Expand table

Property name of the entity	Node of the property	Value	Description
ValidTimeStateEnabled	Data entity node in the designer	Yes (or No)	The value **Yes** makes the entity date effective. The entity must have **ValidFrom** and **ValidTo** fields. These fields are mapped to the **ValidFrom** and **ValidTo** fields of a date-effective data source. The value **No** does *not* disable the enforcement of date effectivity on any date-effective tables that are data sources of the entity.
ValidTimeStateKey	Under the data entity node, **Keys** > **EntityKey**	Yes (or No)	The value **Yes** identifies the key that is required to enforce the date-effective values on this particular entity.

Read activities

Independent software vendor (ISV) development home page

Article • 08/12/2022

This article provides links to topics about development by independent software vendors (ISVs).

- Link X++ modules to packages by using ISV Studio
- ISV licensing
- ISV licensing on-premises

Feedback

Was this page helpful? Yes No

Provide product feedback

Link X++ modules from ISV packages by using ISV Studio

Article • 08/12/2022

Independent software vendors (ISVs) can link their X++ modules to their registered products and solutions by using Microsoft Power Platform ISV Studio. Linking enables ISV's to monitor the success and usage of their applications in finance and operations apps.

> **Note**
>
> For the link from X++ into ISV Studio to work correctly, customers need to have deployed ISV packages with the correct solution ID in all the ISV models. The customer's environment also has to be version 10.0.16 or higher.

Find the product ID in Microsoft Partner Center

Sign in to Partner Center and open the **Offer overview** page for your product. From the browser's URL bar, locate the product ID globally unique identifier (GUID), as shown in the following example.

HTTP

```
https://partner.microsoft.com/dashboard/commercial-
marketplace/offers/<product-ID-GUID>/overview
```

> **Note**
>
> The product ID does not necessarily match the offer code of your product, although they may be similar. Using the offer code in your descriptors will not correctly identify your X++ modules to ISV Studio.

Update your X++ model descriptors

For all models that make up your solution, locate the descriptor XML files. For every descriptor that belongs to a solution, update the `SolutionId` tag with the product ID

Independent software vendor (ISV) licensing

Article • 02/20/2025

This article describes the independent software vendor (ISV) licensing feature. It includes information about benefits and capabilities of the ISV licensing feature, and explains how to enable licensing for an ISV solution, create a package and generate a customer-specific license, and create self-signed certificates for test purposes.

The Microsoft Dynamics ecosystem provides tools and frameworks that let independent software vendors (ISVs) build, deploy, sell, and therefore monetize vertical industry solutions that can be repackaged. The ISV licensing feature provides the following benefits:

- It provides a safer licensing mechanism for ISV solutions for customers and partners. ISV solutions are enabled only if the customer has purchased a valid license key from the ISV.
- It aligns how customers handle licenses for ISV solutions from different ISVs, and therefore lowers the total cost of ownership (TCO).
- ISVs can independently generate, manage, and distribute ISV licenses by using industry standard frameworks.

This feature doesn't enable ISV competitor copycat protection (that is, source-based protection).

> **Important**
>
> Starting with Dynamics 365 Finance version 10.0.43, all licenses are issued at the tenant level. This means a single license can potentially be used across multiple environments that share the same tenant ID.

Capabilities

This section describes various capabilities of the ISV licensing feature.

ISVs can generate their own licenses

ISVs can independently generate their own licenses, apply them to solutions, and deliver those solutions to partners and customers. Each ISV license enables run-time features

Independent software vendor (ISV) licensing (on-premises)

Article • 06/19/2024

This article explains how to import independent software vendor (ISV) licenses into an on-premises deployment.

> **Important**
>
> The process that is described in this article is available only for customers who have on-premises environments that are deployed with Platform update 12 or later.

For general information about the benefits of ISV licensing, information about how to enable licensing for your solution, and other information that is related to self-signed certificates, see Independent software vendor (ISV) licensing.

Generate the license file

1. Collect the tenant name and ID for the customer to issue the license to:
   a. Connect to the instance of Service Fabric Explorer where the environment is hosted.
   b. Go to **Clusters** > **Applications** > **AXSFType** > **fabric:\AXSF**, and then, on the right page, select the **Details** tab.
   c. In the **Parameters** table, find the values for the **License_TenantDomainGuid** and **Licence_TenantId** keys.

2. Generate a license for the customer (tenant ID and name), and sign the license by using the certificate's private key. The following parameters must be passed to the AXUtil **genlicense** command to create the license file. The command will generate an XML file.

Expand table

Parameter name	Description
file	The name of the license file.
licensecode	The name of the license code from Microsoft Visual Studio.