CONTENTS

Module 01: Visual Studio and Visual Team Service

Lesson 1: Fleet Management Scenario

Simon, the systems developer for Contoso, is tasked with creating a new model for the fleet management system in Visual Studio. He is to ensure that it is ready for element creation and that the project builds. For the fleet management scenario, we will need to complete the following four tasks: First, we'll need to configure the Visual Studio environment for Dynamics 365 for Finance and Operations development. Second, we will need to create a new model, package and a project. Third, we will need to create an element within the project; and, finally, we'll need to build the project for testing and delivery.

Lesson 2: Setup and Configuration

There are three primary considerations for development environments. First, the environment can be hosted locally or in the cloud on Microsoft Azure. Second, development environments must have a minimum of 16 gigabytes of RAM and two CPU cores. Third, more RAM and CPU cores translates to faster compile times on your development machine. The first time you use Visual Studio to develop with Dynamics 365 for Finance and Operations, you will need to complete some basic configurations.

Lesson 3: Demonstration: Configure Visual Studio

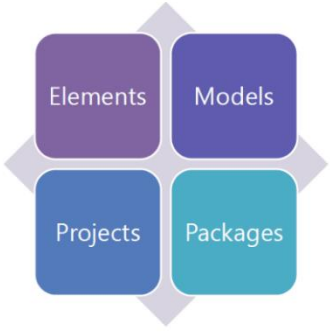
Open visual studio in administrator mode. → Navigate to the Dynamics 365 menu and select the *options* button → Dynamics 365 menu and the project's sub menu

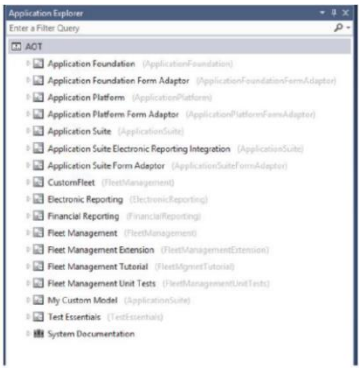
Settings

- Projects are set to organize elements by type - this will create a folder for each type of element in projects
- Projects are configured to automatically synchronize on build - this will automatically synchronize the table changes to the database every time we run a build
- Turn on line numbers in the code editor – this will display line numbers in the left margin of the code editor

Lesson 4: Common Terminology

Four main components in Visual studio





- Elements
  - Data types, tables, classes, forms, etc.
  - Objects that resides in the AOT and Application Explorer
  - These elements are what define the system and dictate what the user will see in the front end

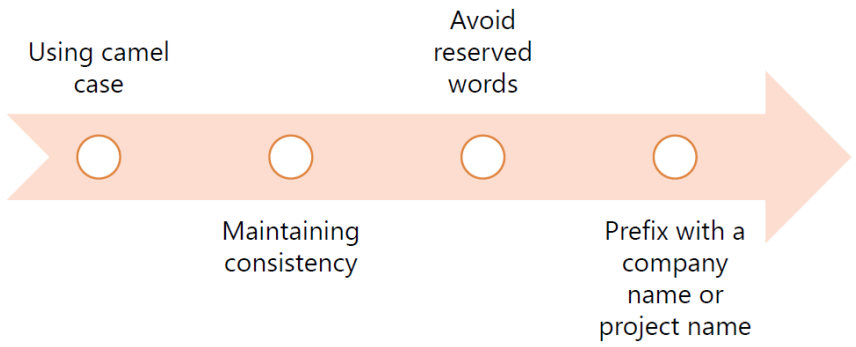
    - Can be customized once they are added to a specific project or model

- Models
    - Group or collection of elements that represents a distributable software solution
    - It is a design time concept
    - A particular model may contain multiple VS projects, however, a project may only belong to one model

- Projects
- Projects in AX2012 - smaller group

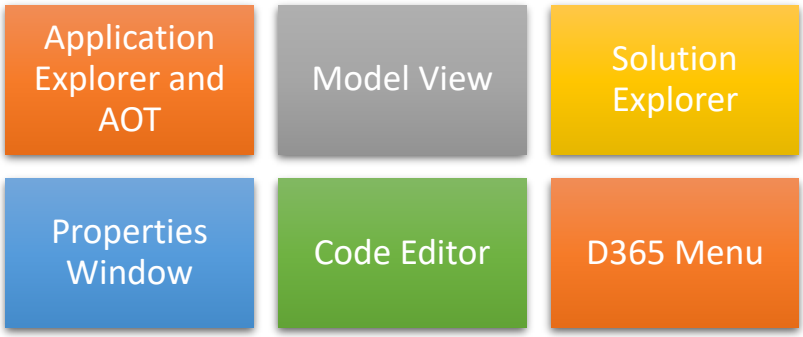
- Packages
    - Deployment unit that may contain one or more models.
    - It contains the elements, model metadata (description data that defines the properties and behavior of the model)
    - Can be exported to a file which can then be deployed into a staging and production environment

Lesson 5: Naming Conventions



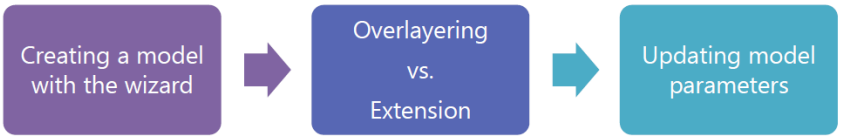
- Best practice to use CAML case, and include prefixes that relate to the originator and the type of element
- Projects - it is important to name projects specifically, so that their purpose can be easily identified
- Prefix certain module-specific elements with the module name (e.g. module elements are prefixed with CUST for customers, production module elements are prefixed with PROD for production)
- Companies may also prefix elements that they have customized with their company initials

Lesson 6: Navigation



- Application explorer and AOT –
- Model view - helpful approach to organizing the AOT
- Solution Explorer - organizes projects under one solution
- Properties window - where we can view and edit element properties
- Code editor & element designer window - where we can create and modify AOT elements in X++ code
- Dynamics 365 menu - where we can configure our environment, create models and perform builds.

Lesson 7: Projects, Models, and Packages



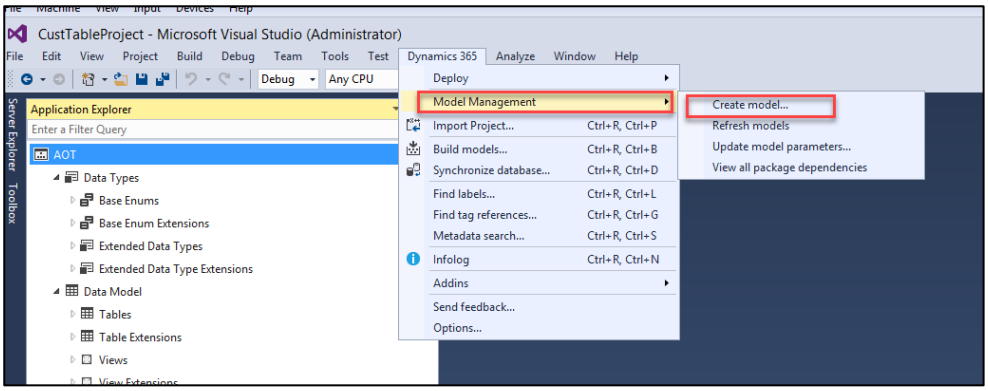
Models - a collection of elements that represent a distributable software solution.
Creating a model involves specifying metadata like the model's name and description

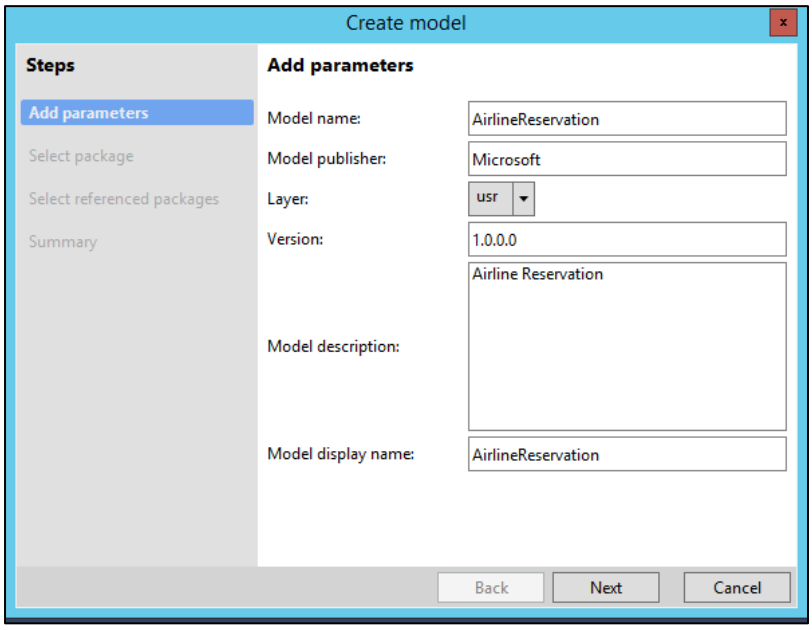
- [Model architecture] A model created in a stand-alone package is called an extension model. A new model added to an existing package is called an over layering model
- Update a model's parameters and set a model's dependencies which will define the other models it is dependent upon. These dependency models are also called referenced models

MODEL CREATION

Navigate to **Dynamics 365 > Model Management > Create Model.**

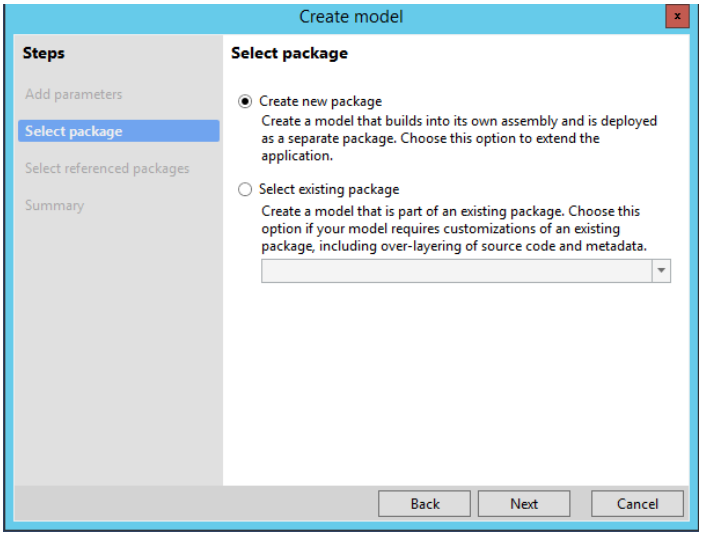
1. Upon clicking *Create model,* wizard for creating a model will be opened.





Model name – will be specified by dev
Model publisher – person who created the model
Model description – describe the model to be created
Model display name – object name that will appear on AOT

2. If “Select existing package” is chosen, that means we’ll be using our over-layering or customization development approach. In this, we must select a model that contains the elements we wish to customize as a referenced model in the wizard. In this way, our new model has access to those elements.

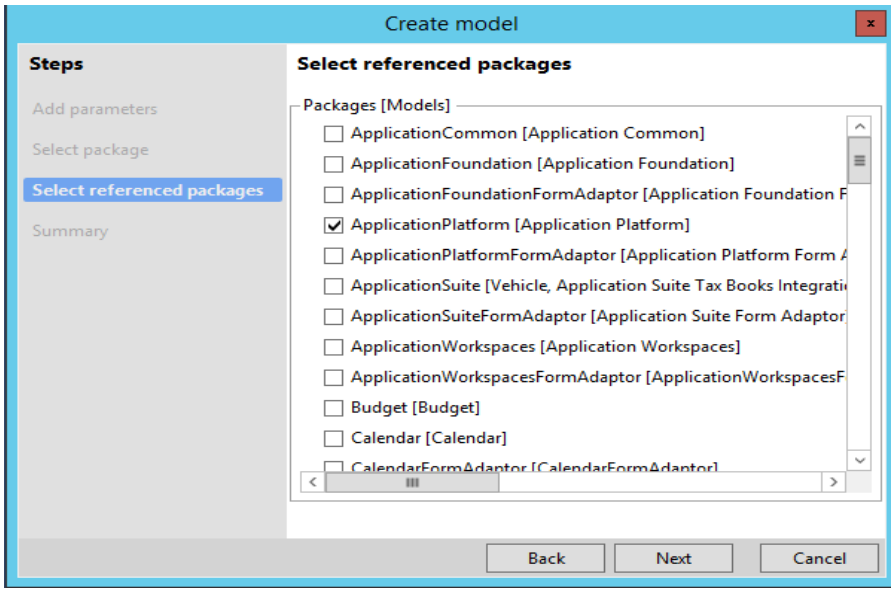


If "Create new package" is chosen, this implies we're going to use the extension model approach. Developing this way allows us to access the elements from any models we selected as referenced models. We can create new elements and add functionality to the existing code.

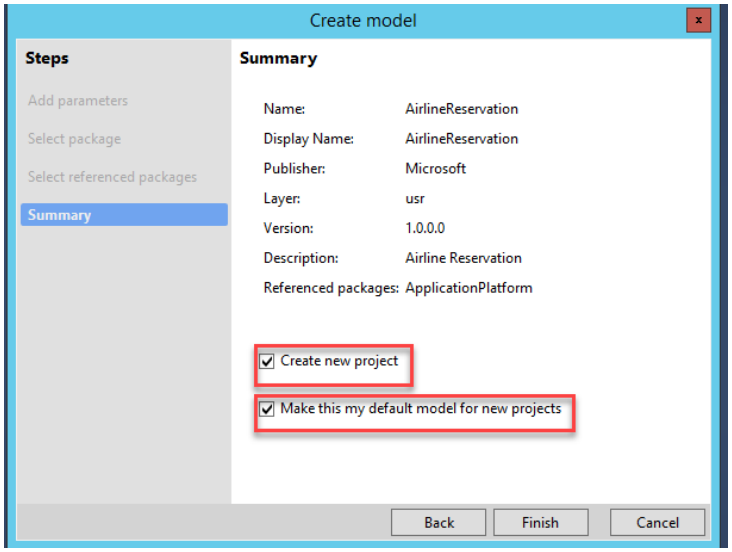
In the "AirlineReservation" model, we're going to create a new package (we're using the extension development approach). Select the "Create new package" option and click "Next."

Now we're going to select our referenced models. This means we're choosing the other models our "AirlineReservation" model can see, or which elements it'll have access to.

3. Select applicable packages to be use as reference



4. Summary window will display model name, display name, publisher and the referenced packages.



Over layering

- This allows you to directly customize metadata and source code in another layer. For example, you can change the design of a form in the VAR layer or CUST layer using an over layering model. However, because it is tightly coupled with the containing package, this method of customization yields slow compile times and could potentially increase the cost to upgrade to a new version of the product in the future.

Extension

- Development using extensions allows us to access the elements from any models that are specified as referenced models in the wizard. This allows us to add functionality to existing code. We can create new elements and create extensions of referenced elements in our projects. By creating extension elements, all of the customizations and code are stored in a separate assembly which contains only the additions or changes in the extension. Because this code resides in a separate assembly, it improves the performance of building and testing. This approach is also ideal for upgradable customizations as it eliminates metadata conflicts.

Lesson 8: Demo: Create a 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**.

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.
Company	The default company that will be used when the project is run.
Partition	The partition that will be used when the project is run.
Project File	The name of the file that contains information about the project.
Project Folder	The location of the project.
Model	The model that the project is associated with. All elements in the project must be in the selected model.
Model Publisher	A read-only value that indicates the publisher of the model.
Layer	A read-only value that indicates the application layer that the model is located in.
Synchronize database on build	A value that indicates whether the synchronize operation for tables will be performed when the build action is performed for the project.

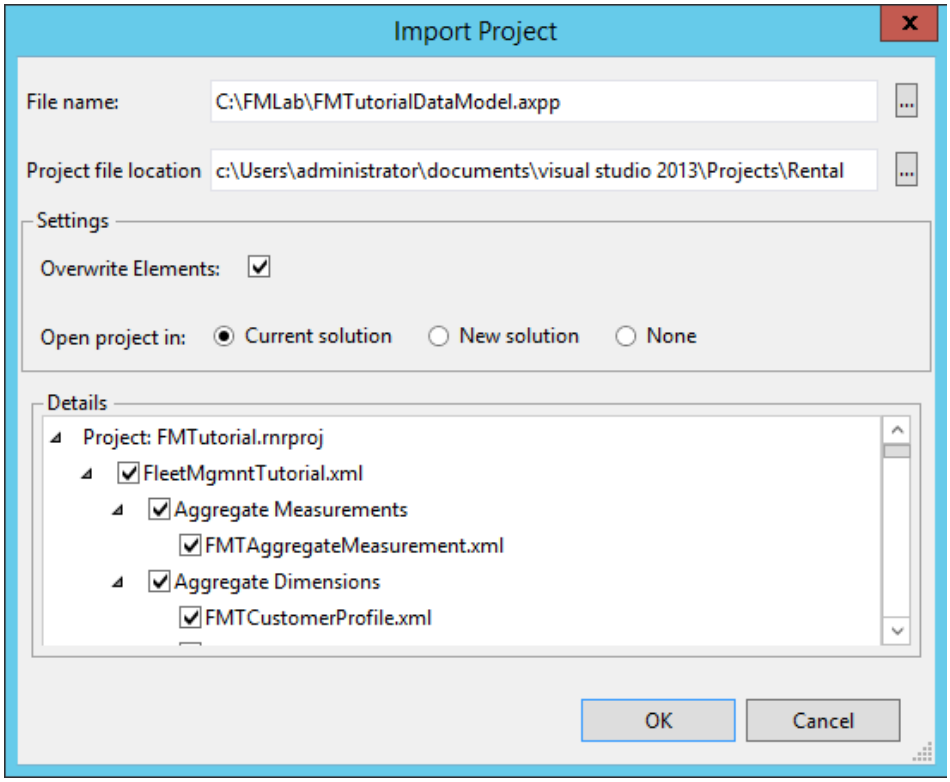
- Project package files
    - Contains all of the elements form a project
    - Used to import/export between different instances of AX
    - .AXPP file name extension
    - To export: Project menu> Export Project:
    - To import: Dynamics 365 Menu > Import Project

Lesson 9: Demo: Import an AXPP File and Add a New Element

IMPORT AXPP FILE

To use the contents of a project package file, you must import the .axpp file into an installation. The elements from the project package file will be imported into the same model that they were exported from. If that model doesn't exist in the installation, it will be created during the import process. To import a project package file, follow these steps.

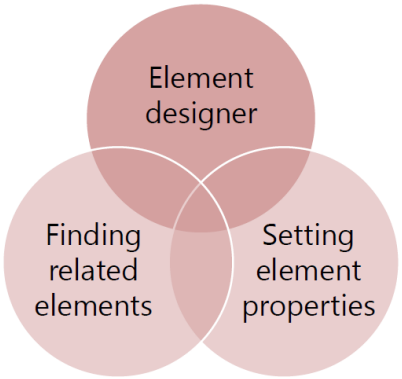
1. On the **Dynamics 365** menu, click **Import Project**.
2. In the **Import Project** dialog box, specify the location of the project package (.axpp) file to import.
3. If you want elements from the project package file to overwrite any existing elements, select **Overwrite Elements**.
4. Specify whether you want to open the project in the current selection, in a new solution, or not at all.
5. In the **Details** field, review the elements that will be imported. You can clear the check box next to any elements that you don't want to import.



6. Click **OK** to complete the import process.

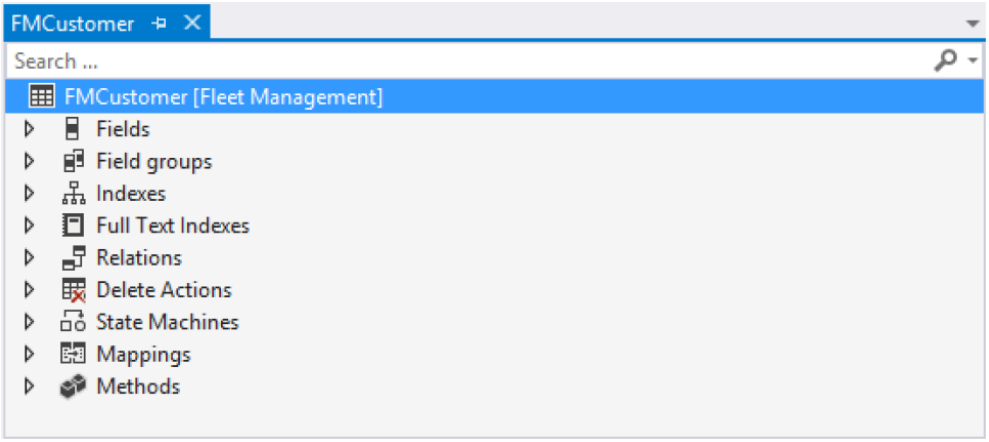
Lesson 10: Using Elements

ADD A NEW ELEMENT



The element designer is used to edit an element. For example, we can add fields to table using the element designer. We can then set properties for the element, such as the name or label, using the properties window. You can also navigate between related elements in Visual Studio.

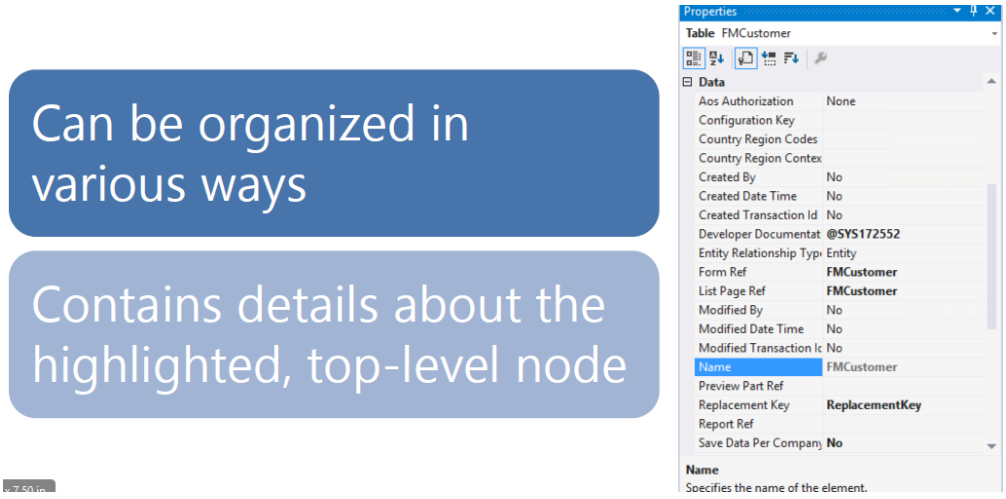
ELEMENT DESIGNER



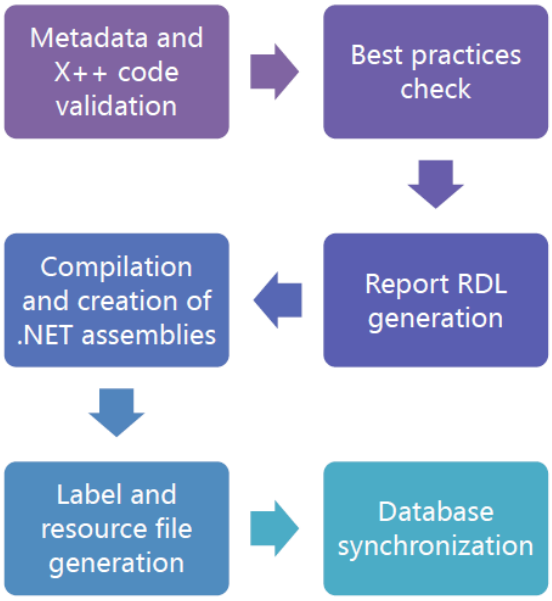
Element designer allows to customize an element
Contains various features of element

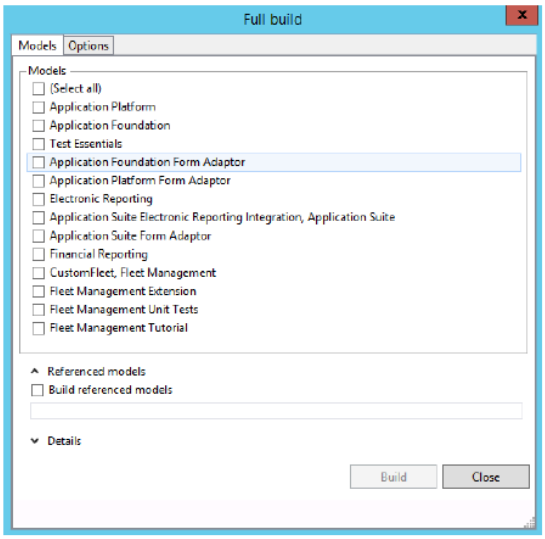
- All development is done against local xml files.
- These xml files are stored in a local disk in folders organized by package and model
- The actual xml file contains both the metadata for an element as well as any source code for an element
- This means when you customize a table, it stores the property of the table such as the name and the table type along with x++ code
- Keep in mind that the element properties pane only shows properties for the currently highlighted node in the element designer.

ELEMENT PROPERTIES



Lesson 11: Performing Builds





The purpose of build > so that it can be used by the application
A full build can be performed on an entity model or multiple models
It can also be performed on a project. Only build the elements that are in the current project
Build > Build [project]
Build > build solution
Build entire model
It only builds new or changed elements
When we delete an element, choose the rebuild options in order for that deletion to take effect
Rebuild forces all element in the project to build regardless of whether they have changed or not
Right click project > properties > Synchronize database on build [true/false]

Module 02: System Architecture

Lesson 1: Application Stack and Server Architecture

The application stack and server of Dynamics 365 Architecture aligns with three key pillars:

- New client
- Cloud readiness
- New development stack

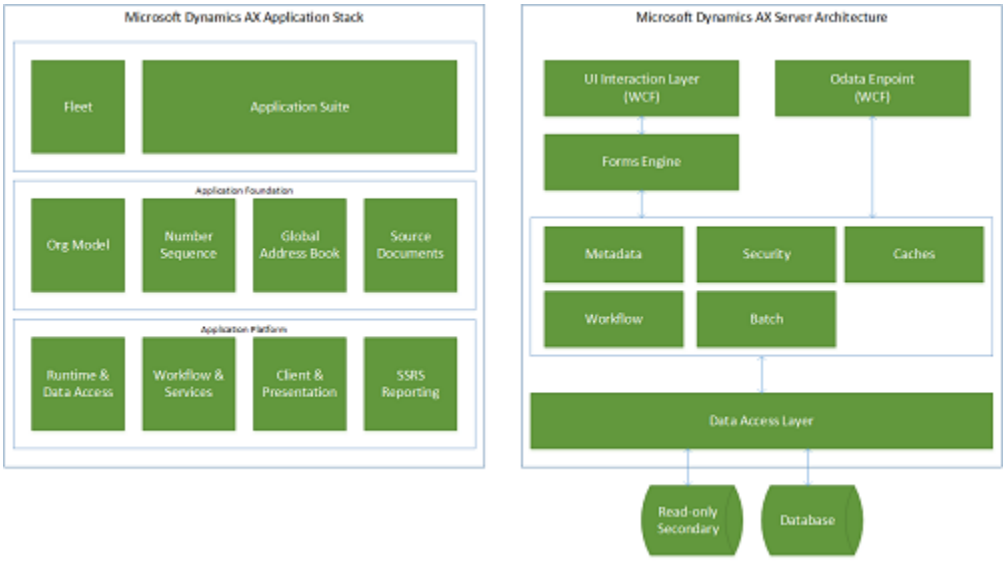
The application stack has been divided into 3 separate models: **Application Platform**, **Application Foundation**, and **Application Suite**. The separation enables the new application development on the base foundation models, just as the Fleet Management sample application has been developed. Note that the following important points about the changes in the server architecture:

- The services endpoint on a 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 a server. The form objects still run on the server, and rendering has been optimized for browsers and other clients through a server and client-side (browser) investments.
- The server, adding 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 the service (IaaS) virtual machines (VMs).
- It is hosted on Azure and is available for access through an Internet. A user can use a combination of clients and the credentials to access it. The recommended primary identity provider is OrgID, and store for the identity is Azure Active Directory (Azure AD). The security subsystem uses the same AuthZ semantics for users and their roles.
- Two types of clients must be considered for access in the cloud: active clients and passive clients.
  - Active clients can programmatically initiate actions based on responses from a server. An active client doesn't rely on HTTP redirects for the authentication.
  - Passive clients can't programmatically initiate the actions based on responses from the server. A passive client relies on HTTP redirects for authentication.

  Currently, Access Control Service (ACS) doesn't help a mechanism for non-interactive authentication. Therefore, even when the active clients try to authenticate by using ACS, they must use passive client authentication, in which a browser dialog box prompts the user to enter his or her credentials.

- A completely revamped metadata subsystem incorporates a new compiler and Microsoft Visual Studio–based development model. The model store is represented as the set of folders and XML artifacts that are organized by model. The model elements, such as tables, forms, and the classes, are represented by an XML file that contains both metadata and source code.

The left side of the below diagram shows how the application stack has been split into distinct models. The right side shows how the key components are stacked in a server.



Lesson 2: Cloud Architecture

The cloud Dynamics 365 Architecture includes services that automate software deployment and provisioning, operational monitoring and reporting, and seamless application lifecycle management. The cloud architecture consists of 3 main conceptual areas:

- Lifecycle Services (LCS) – LCS is the multi-tenant shared service that enables a wide range of lifecycle-related capabilities. Capabilities that are specific to this release add software development, customer provisioning, service level agreement (SLA) monitoring, and reporting capabilities.
- Finance and Operations – The VM instances are deployed through the LCS to your Azure subscription. Various topologies are available: demo, development/test, and the high-availability production topologies.
- Shared Microsoft services – Finance and the Operations use several Microsoft services to enable a "One Microsoft" solution where customers can manage a single sign-in, subscription management, and billing relationship with Microsoft across Finance and the Operations, Microsoft Office 365, and other online services.

Many features of the Azure platform are used, such as a Microsoft Azure Storage, networking, monitoring, and SQL Azure, to name a few. Shared services put into the operation and orchestrate the application lifecycle of the environments for participants. Together, Azure functionality and the LCS will offer a robust cloud service.

Module 03: Labels and Resources

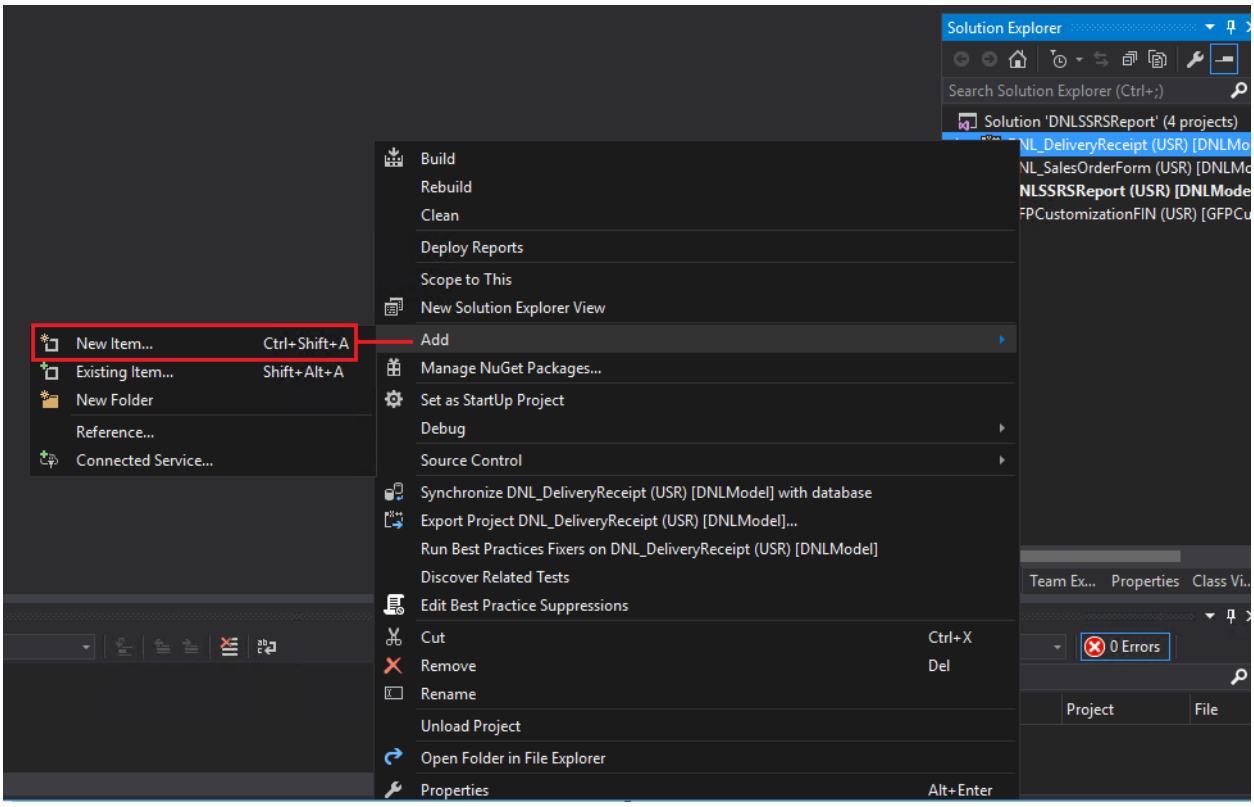
Lesson 1: Labels

Label files are small data structures that keep all related and new labels in a central storage

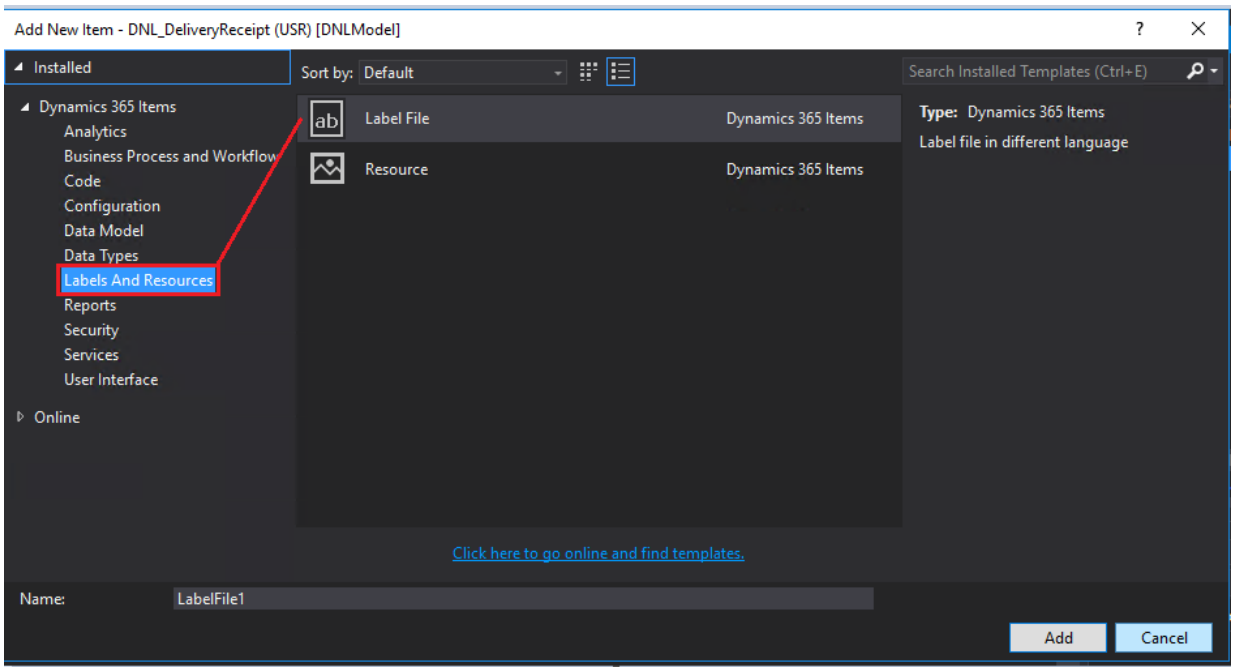
Labels themselves are a reference from the element or artifact that point back to the label file

Create Labels

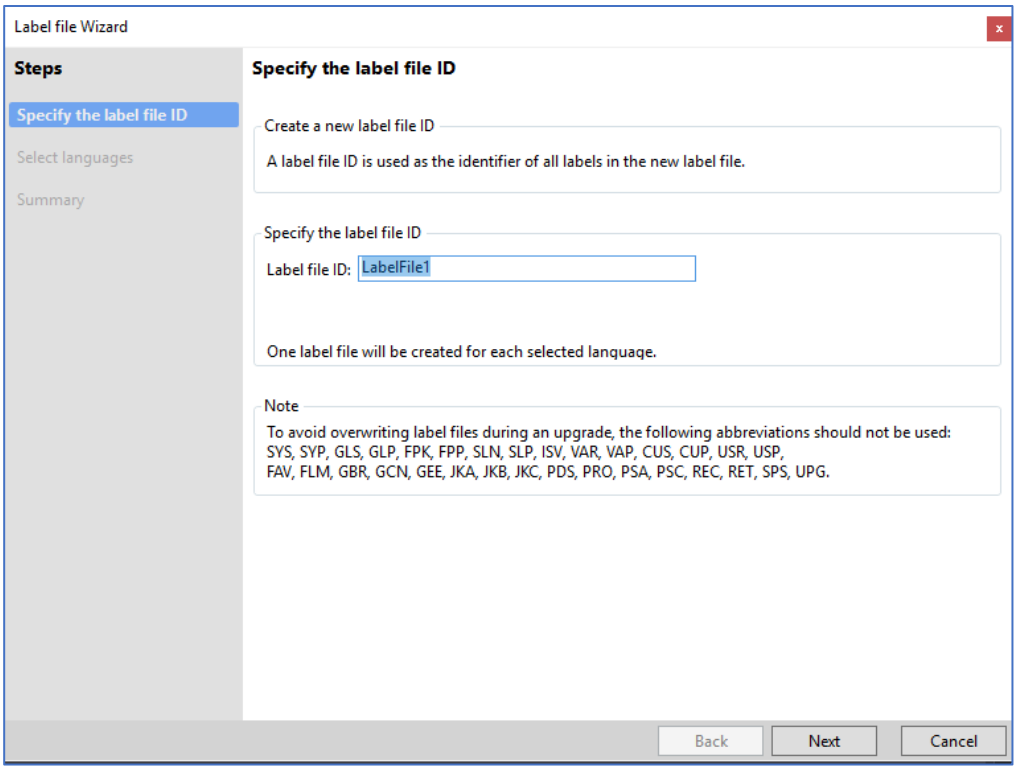
1. Right click on your project under solution → Click Add → Click New Item



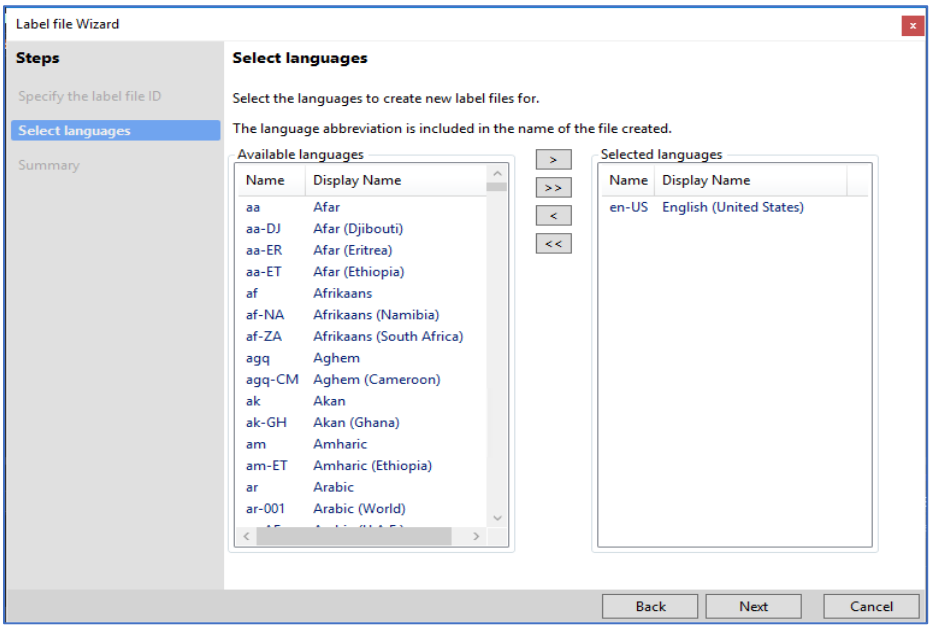
2. Select *Labels and resources* under Dynamics 365 Items tab → Then select *Label File* → Specify label file name → Click Add



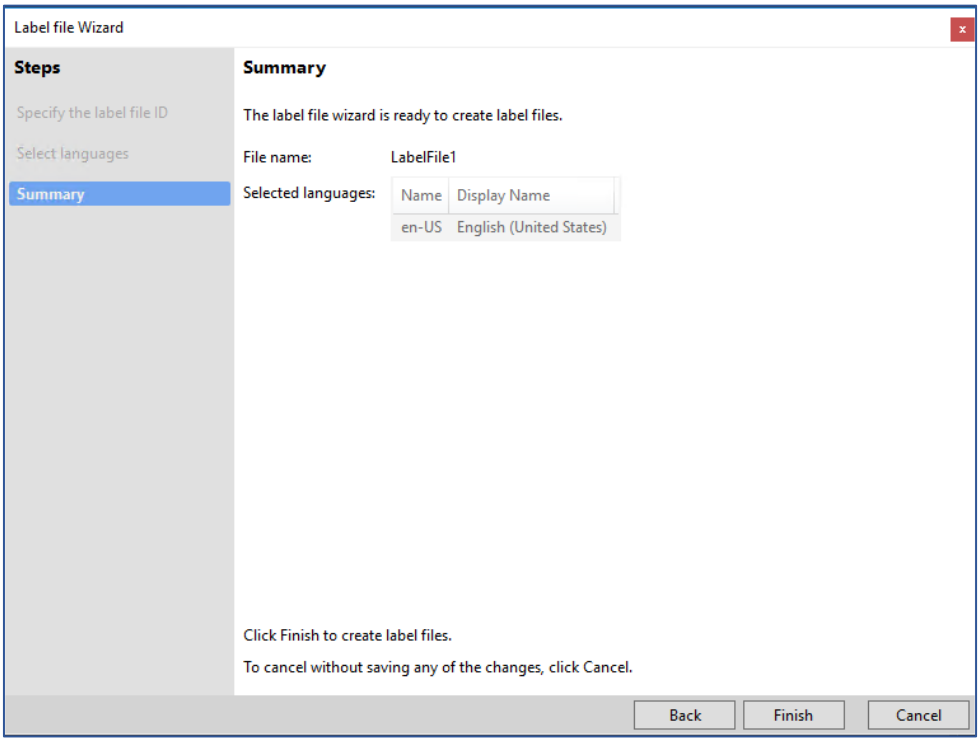
3. Specify label file ID → Click Next



4. Select appropriate language for newly created label file → Click Next



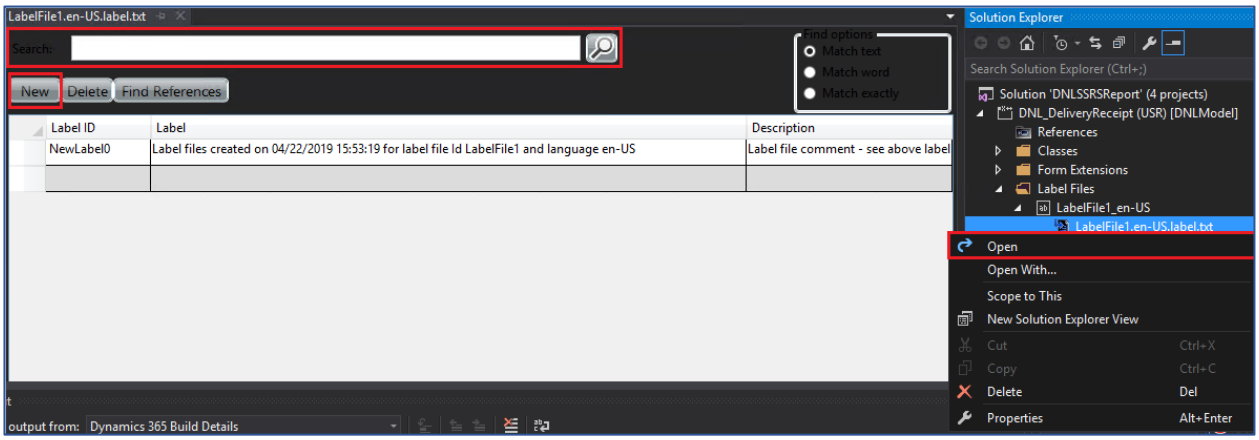
5. Summary window will display related information to the newly created label file. Then click finish



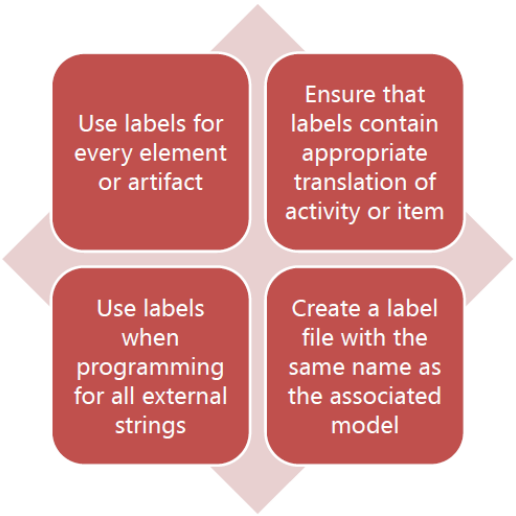
to add label file on project solution.

6. Right click *LabelFile1.en-us.label.txt* → Click *Open*

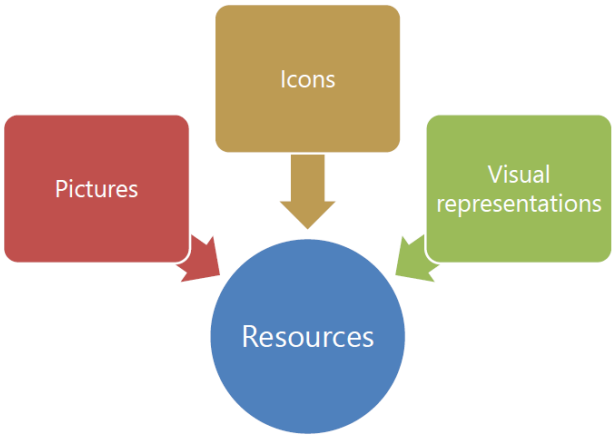
   To create new label, just click *New* button. And if you want to delete a label, click *Delete* button. To search for a specific label, type the keyword on the search bar. Select options if you want an exact match etc.



<u>Best practice for labels</u>



Lesson 2: Resources



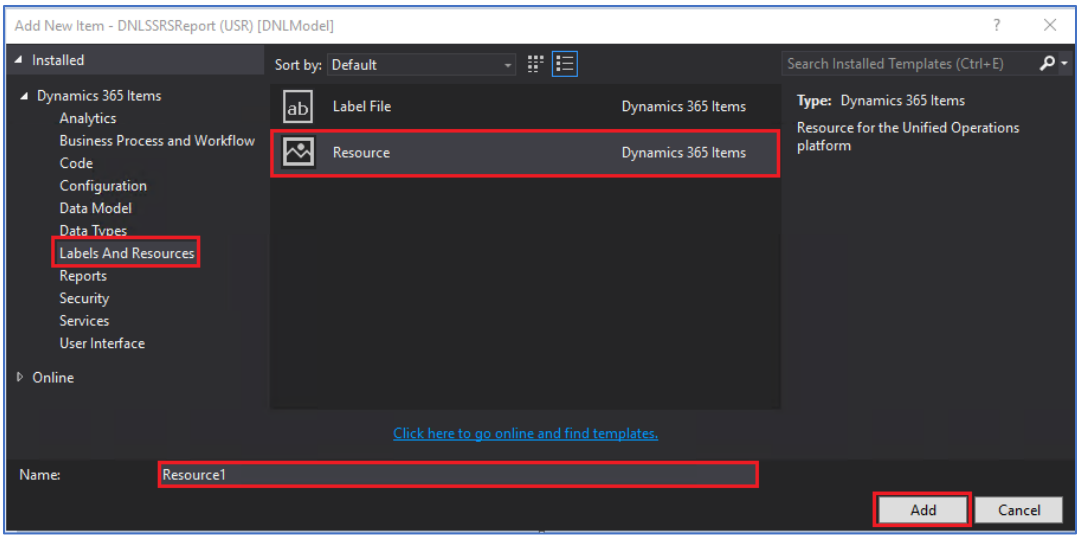
Resources are the way to manage the pictures, icons, and other visual representations that will be used within the user interface

This can be accessed through the resource library, adding additional resources or using resources that are at a URL location.

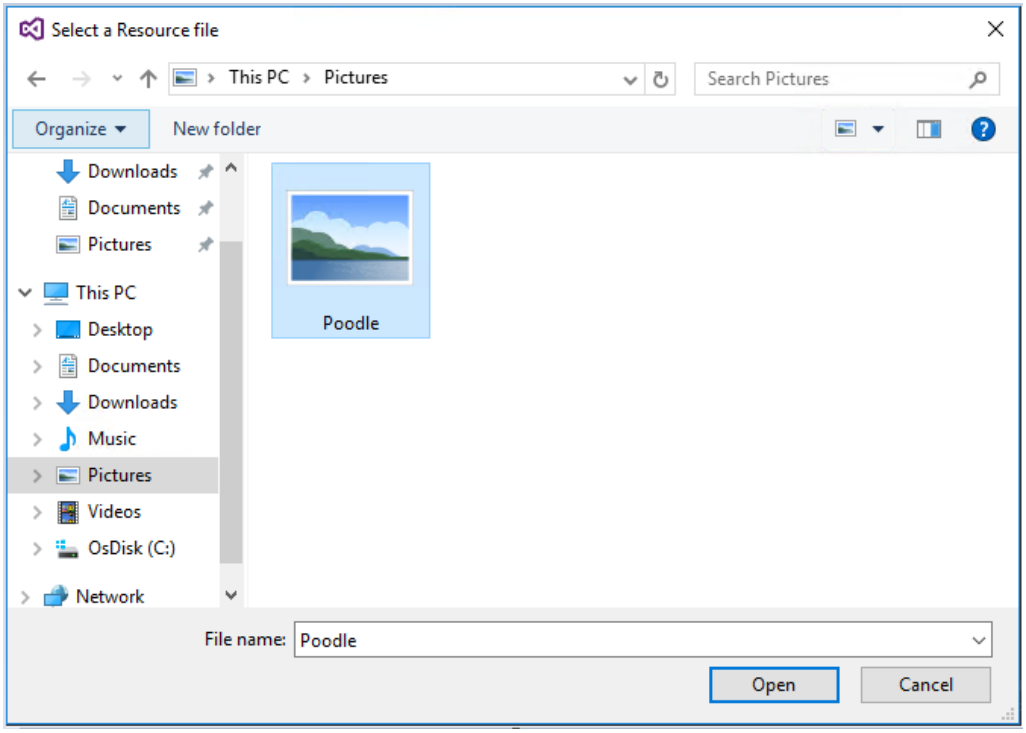
The URL location would be used in the case of an IIS server instance with an established product photo library.

Create resources

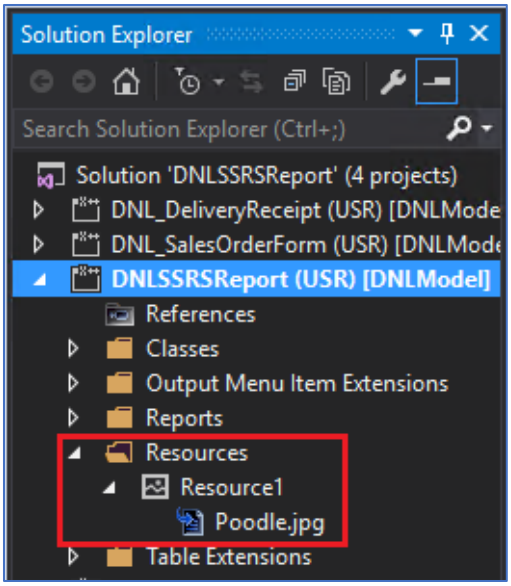
1. Right click project → Click Add → Select New Item
2. Select Labels and Resources → Select Resource → Specify resource name, then click Add button



3. New window will open, and you'll have to choose a file to be uploaded as your resource file → Click open.



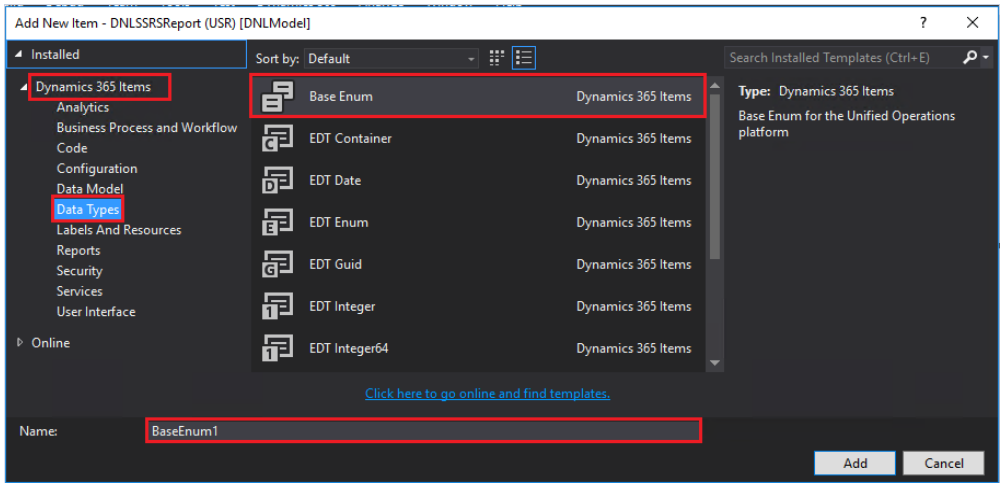
Resource file will be added on your project as seen on the screenshot below.



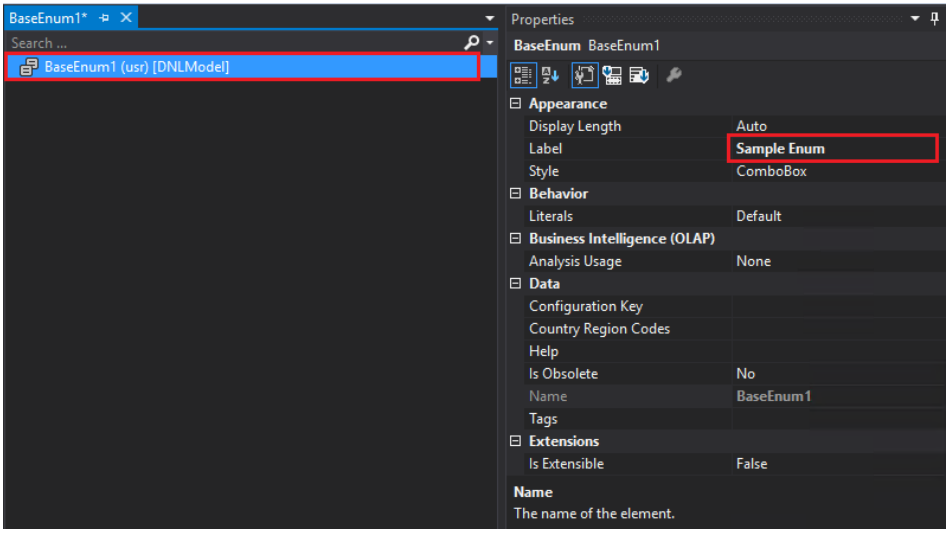
Module 04: Base Enumerations

Lesson 1: Create base enum

1. Right click on project → Click **Add** → **New Item**
2. Under **D365 Items,** click **Data Types** → **Base Enum** → Then enter base enum Name → Click **Add**



3. Set the label
    a. Right-click the newly **BaseEnum1** created in the middle pane and select **Properties**.
    b. Enter desired label in the **Label** field under the **Properties** section on the lower right side of the screen.
    c. Right-click the newly created **BaseEnum1 (usr)** in the middle pane and click **New Element**.



4. Build project / solution.

Lesson 2: Considerations and Best practice

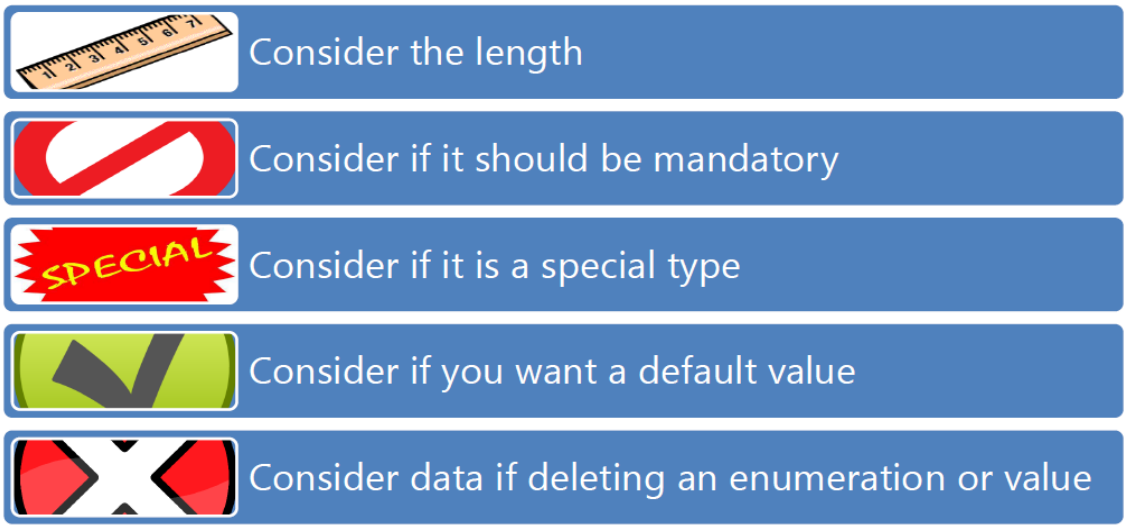


Figure 1 - Considerations

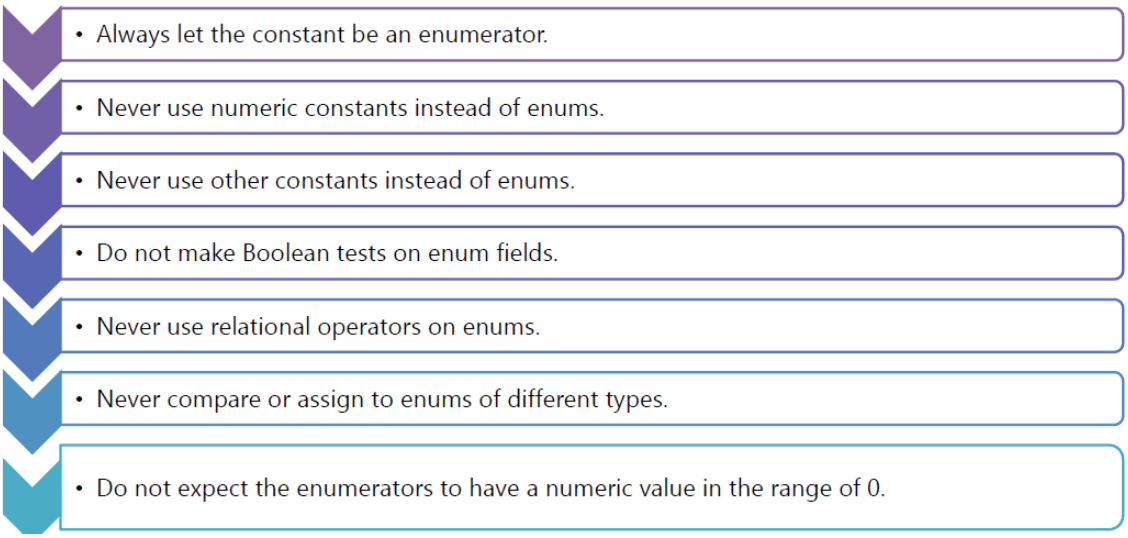


Figure 2 - Best practice

Module 05: Extended Data Types

Lesson 1: Primitive Data types

Type	Description
Anytype	A placeholder for any data type
Booleans	Can only contain the values false and true.
Dates	Contains day, month, and year.
Enums	An abbreviation for enumerated text—a set of literals.
GUIDs	A globally unique identifier.
Integers	A number without a decimal point. To declare an integer, use the keyword int.
Real	Numbers with a decimal point; also called decimals.
String	A number of characters. To declare a string, use the keyword str.
TimeOfDay	Contains hours, minutes, and seconds. To declare a time, use the system type timeOfDay.
UTCDateTime	Contains year, month, day, hour, minute and second.

Figure 3 - Primitive Data Types

An EDT is a primitive data type or container with a supplementary name and some additional properties. For example, you could create a new EDT called Name and base it on a string. Thereafter you can use the new EDT in variable and field declarations in the development environment.

Lesson 2: Extended data types

Extended data types (EDTs) are user-defined types, based on the primitive data types boolean, integer, real, string, and date, and the composite type container. You can also base EDTs on other EDTs.

This feature is not implemented as a language construct. EDTs are defined in the Application Object Tree (AOT).

Benefits of EDTs

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

Lesson 3: Best Practice

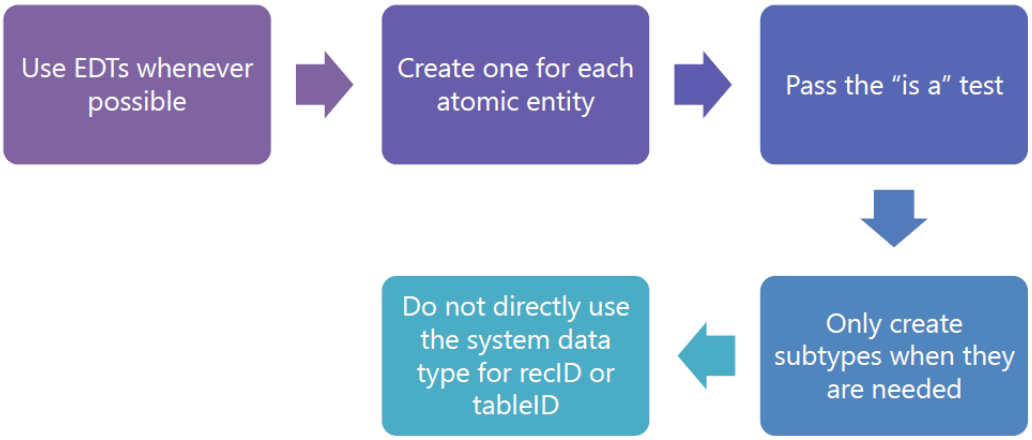


*Figure 4 - EDT hierarchy should be able **to pass the is a test**. This means an extended data type should be a member of the parent extended data type concept*

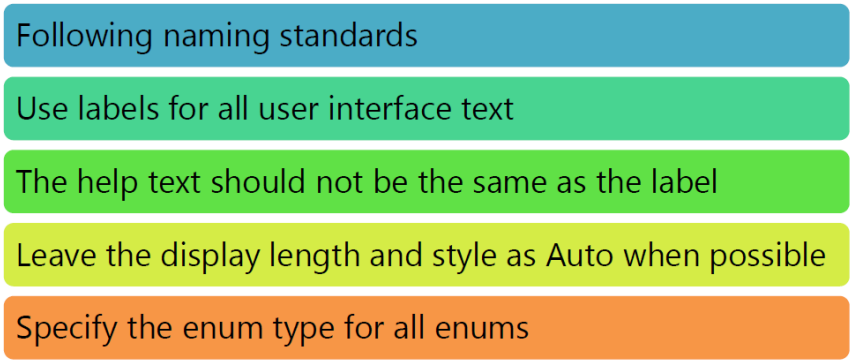


Figure 5 – Best Practice for EDT properties

Module 06: Tables

Tables are the foundation objects in Microsoft Dynamics AX and store data used by the system. A table is made up of records (or rows) that contain information about a single entry in the table. For example, a specific customer or product. A record consists of one or more fields (or columns) that contain a discrete piece of data of a specific data type.

In Microsoft Dynamics AX, tables are located in the Application Object Tree (AOT) under the **Data Dictionary\Tables** node. Each table contains the following primary elements:

- Fields
- Field Groups
- Indexes
- Relations
- DeleteActions
- Methods

A table name can contain letters and numbers but must begin with a letter. Spaces and special characters are not allowed.

Lesson 1: Table components

Full Text Indexes are similar to regular Indexes, but they are optimized specifically for substring text search.

The **Delete Actions** node will not be used for new development. Starting with this version of the product, delete actions are specified on the properties of a table relation. However, the node remains for upgrades from previous versions. We will look at these pieces more in depth in the next modules.

The **State Machine** feature enables workflow events to be bound to state transitions on the underlying entity's state machine. This enables you to perform state transitions within a workflow without writing any additional code.

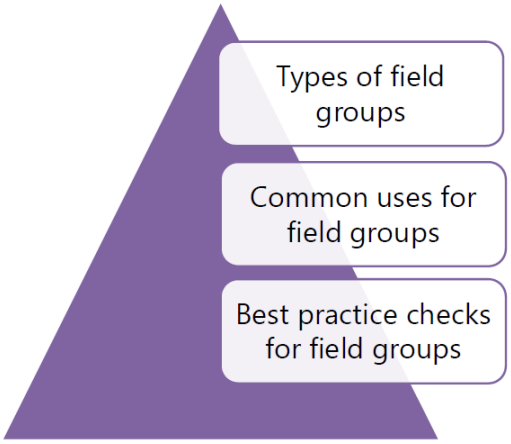
The **Mappings** feature enables you to associate table fields with fields defined in a separate map. This means you can use the same field name to access fields with different names in different tables. For example, in two separate address tables, one may refer to the Zip Code field as Zip Code and the other table might call it Postal Code, but a common map referenced by both tables may unify the field reference as just Zip.

Events are triggers that occur in conjunction with certain activities on the table. They initiate any event handlers which are listening for certain events. For example, if we want to activate a method when a table record is being updated, we could register an event handler against the on updating event, which exists by default on every table.

Lesson 2: Create Tables

1. In the Solution Explorer, right-click **Project.**
2. Click **Add > New Item**.
3. In **Dynamics 365 Items**, select **Data Model.**
4. Click **Table.**
5. Enter **tableName** in the **Name** field.
6. Click **Add**.
7. Add fields / EDTs
8. Set labels and properties of fields
9. Build

Field groups



Lesson 3: Temporary Tables

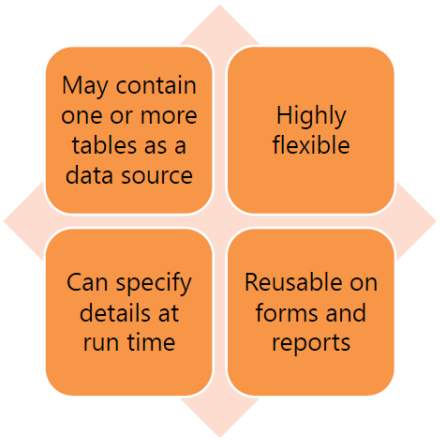
A temporary table contains data only when it is in use or instantiated by a particular process. The data is automatically deleted at some point after all references to the table go out of scope.

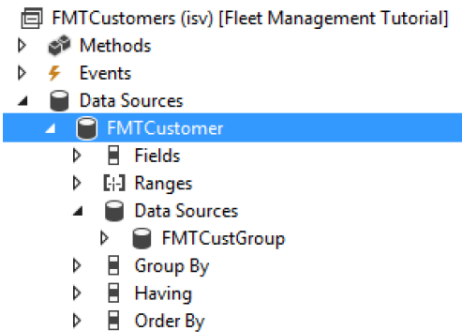
There are two main types of temporary tables: **InMemory** and **TempDB**. TempDB tables have a persistent schema in the database, but they are configured to delete table data when references to the table go out of scope. In contrast, an InMemory table does not exist in the database by default and is instead held in memory until the size reaches 128 KB.

InMemory tables size reaches 128kb

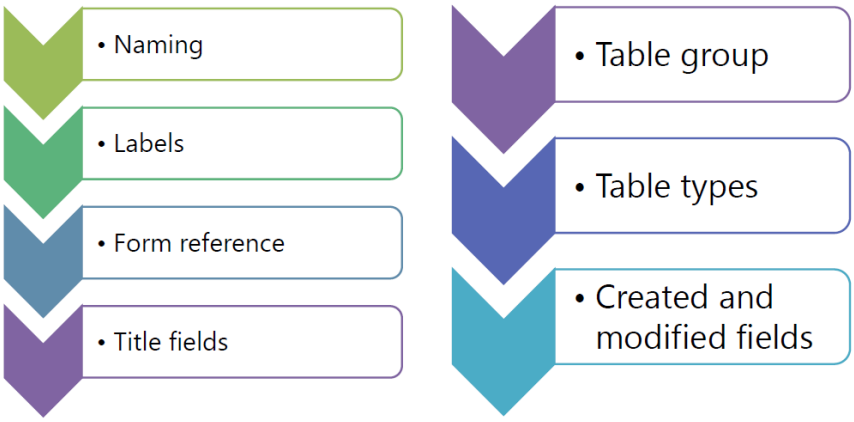
Rule of thumb: use **TempDB** table if your data set or report will exceed 5000 rows or records

Lesson 4: Queries





Lesson 5: Best Practices for Tables



The **Name** of a table should contain several segments: first, the module the table is associated with; second, a logical description of the table contents; third, the type of contents, which can include such strings as Trans, Jour, Line, Table, Group, Parameters, or Setup; and fourth, you can optionally include a postfix of T-M-P for temporary tables, or the string 'Table' for primary tables in each module.

The **Label** property is mandatory and enforced by the compiler unless the table has the Visible property set to No. The text value of the label must be unique across all tables, views, and maps, in all languages. Temporary tables are excluded from this system-wide validation.

For tables where the **FormRef** property is set to a display menu item, the display menu item must exist.

The title field properties encompass **TitleField1** and **TitleField2**. **TitleField1** should be a key field for the records in the table. **TitleField2** should be a descriptive field for the records in the table. The **InventItemGroup** table is a good example of how to use the title fields. Its **TitleField1** property is set to **ItemGroupId**, which is a key field on the table, and **TitleField2** is set to **Name**, which is a descriptive field for the records in the table. If the value of **TitleField1** and **TitleField2** are the same, an error will occur.

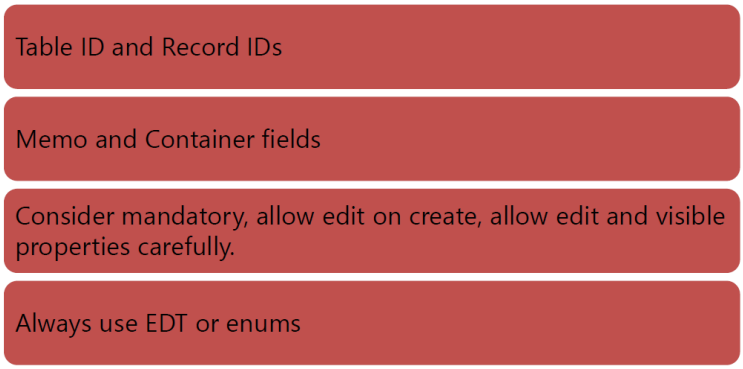
The **TableGroup** property is Mandatory. It is used to categorize tables by the type of data they contain, and should be set to the closest matching group for all tables.

The **TableType** property is used to determine whether a table is **Regular** or **Temporary**. You can use the setTmp table method to make a non-temporary table temporary rather than creating a copy of the table and changing its **TableType**.

Every table supports adding the **Created** and **Modified** data fields. Carefully consider if you need to track the created or modified by, date, and transaction ID. Only set these properties to Yes if the information provided by the field is needed.

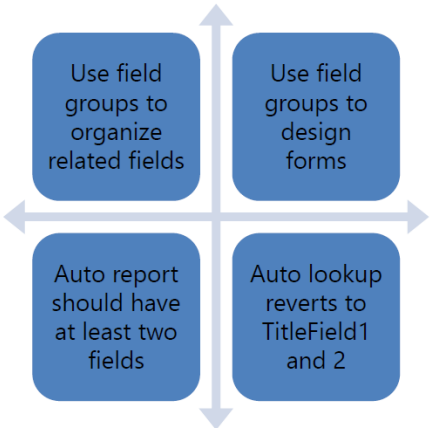
Lesson 6: Best Practices for Table fields

- For table IDs and Rec IDs, do not directly use the system data types recId or tableId. Instead, use the the RefRecId EDT for fields that refer to record IDs, and the RefTableId EDT for fields that refer to table ID.
- Consider your use of memo and container fields in application tables. Adding these field types to database tables will negatively impact application and database fetch times. They also inhibit array fetching, and they cannot be used in WHERE expressions. Consider using String fields with a specific number of characters instead of memo fields. Also, when selecting from a table that



  contains memo or container fields you aren't using, you should use a field list to exclude these and any other fields that you aren't using.
- Exercise caution when setting the mandatory, allow edit on create, allow edit, and visible properties of the field. These properties control user and developer access to fields, so you need to consider the possible side-effects of constraining when a field can or should be edited or viewed.
- Always used extended data types or enums for your table fields. By setting properties on an enum or EDT instead of directly on a table field, you are increasing reusability and decreasing potentially redundant development work.

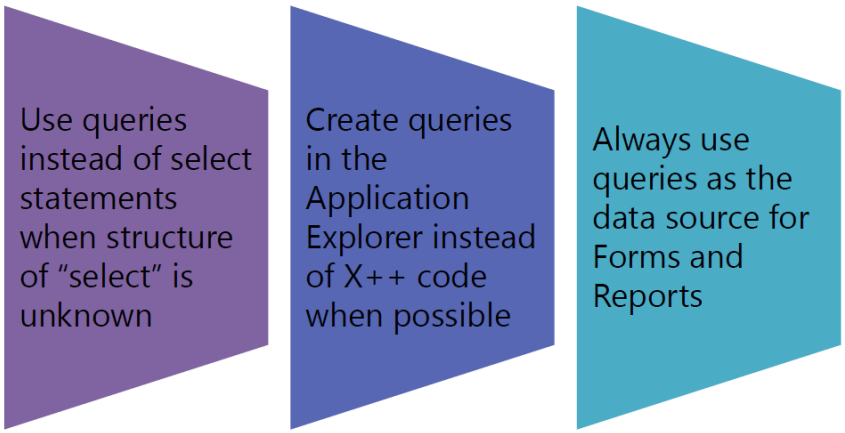
Lesson 7: Best Practices for Field groups



- Place at least two fields in the **AutoSummary** field group for each table. This will automatically populate the summary view of a fasttab in any form which references the table. The fields used for TitleField1 and TitleField2 are often candidates for this field group.

- If you do not put anything into the **AutoLookup** field group, the system will revert to the fields referenced in **TitleField1** and **TitleField2** plus the contents of the first unique index. This is acceptable for some applications, but you should consider whether you need to configure the group differently as you design a table.

Lesson 6: Best Practices for Queries



Module 07: Table Indexes

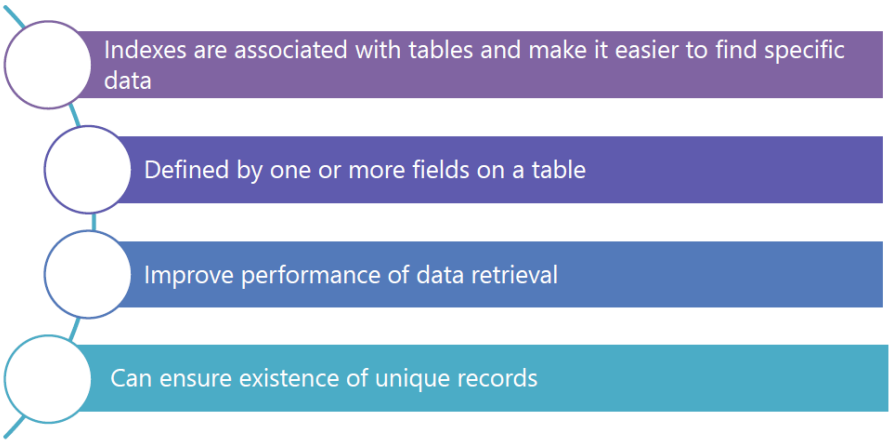
Lesson 1: Index types

Primary Index: A primary index provides a *unique key* for each record in a table. This index type is used for record caching in Microsoft Dynamics 365 for Finance and Operations. In order to be a unique index, the Allow Duplicates property must be set to "No". This prevents the insertion of duplicate key values in a table.

Clustered Index: A clustered index *organizes the data for a table* in the order of the index. A clustered index can also be unique, in the case that the primary index and the clustered index are the same. A phone book is a good example of a clustered index. The data in a phone book is first sorted by last name and then by first name. For each last and first name listed in the phone book there is a corresponding phone number and address. The data for a table can only be physically organized one way and therefore there can only be one clustered index

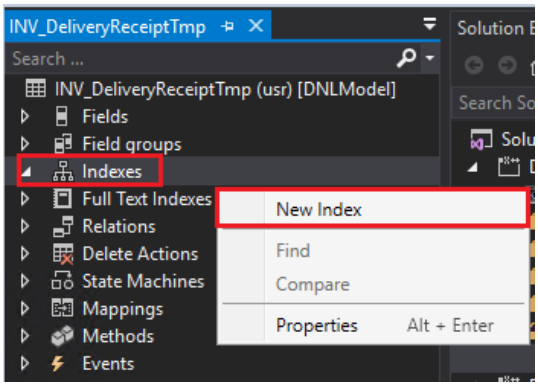
Non-Clustered Index: A non-clustered index provides a way to quickly reference data found in the clustered index or heap (table without a clustered index) using a specified set of columns. We call them "non-clustered" because they don't change the physical sorting of the data. An example of a non-clustered index is the index section at the back of a textbook. You can look up the topic you want, and the index provides a list of page numbers which have information on that topic. A hypothetical example would be adding a phone number index to a phone book so you could more quickly find an address by searching for a phone number. Without a phone number index, a reader would have to slowly scan every address in the phone book to find their match. The database term for this search without an index is a 'table scan'.

About Indexes



Create an Index

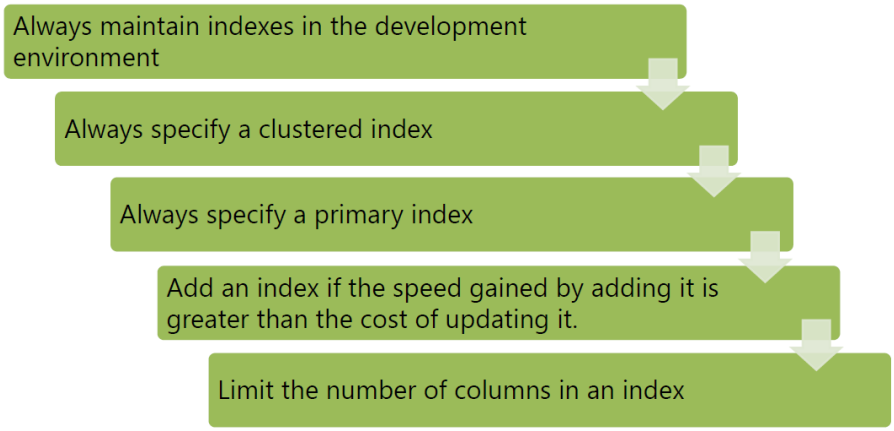
1. In the Solution Explorer pane, under your **Project**, scroll down to see the **Table**.
2. Right-click the table **TableName**.
3. Click **Open**.
4. When the table designer opens, right-click the **Indexes node**.
5. Click **New Index**.



6. Right-click **Index1**.
7. Click **Properties**.
8. Click the **Name** property.
9. Type "**VehicleIDidx**".
10. Scroll up to see **Allow Duplicates**.
11. Ensure that Allow Duplicates is set to **Yes**.
12. Right-click **VehicleIDidx**.

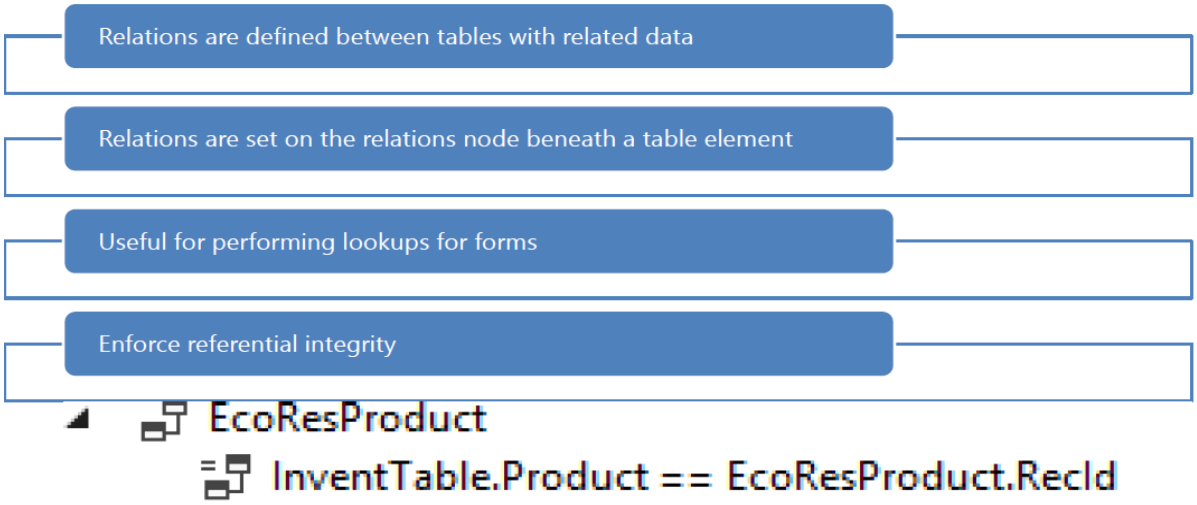
13. Click **New Field**.
14. When the new field is selected, in Properties, click **Data Field**.
15. Click the drop-down arrow.
16. Select **fieldName**.
17. Right click **Solution** → **Build**.

Lesson 2: Best Practices

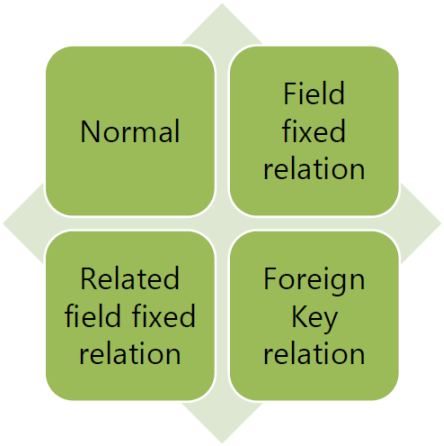


Module 08: Table Relations

Lesson 1: Relations



RELATION TYPES



- **Normal Relation** - used to specify relations without conditions

  1. In the **Field** property, we select the field in the current table that relates to a field in the related table.

  2. In the **Related Field** property, we select the field in the related table that relates to a field in the current table.

- **Field Fixed Relation** - used to specify field restrictions in a table. Usually used in conjunction with other relation types, which are ultimately AND'ed together

  1. In the **Field** property, we select the field in the current table which needs to be restricted.

  2. In the **Value** property, we enter the value we want to use to filter the selected field. This constrains the parent relation to only records in the table that match the specified field value. Only numeric values can be entered in the Value property. Field fixed relations can be created only on numeric fields.

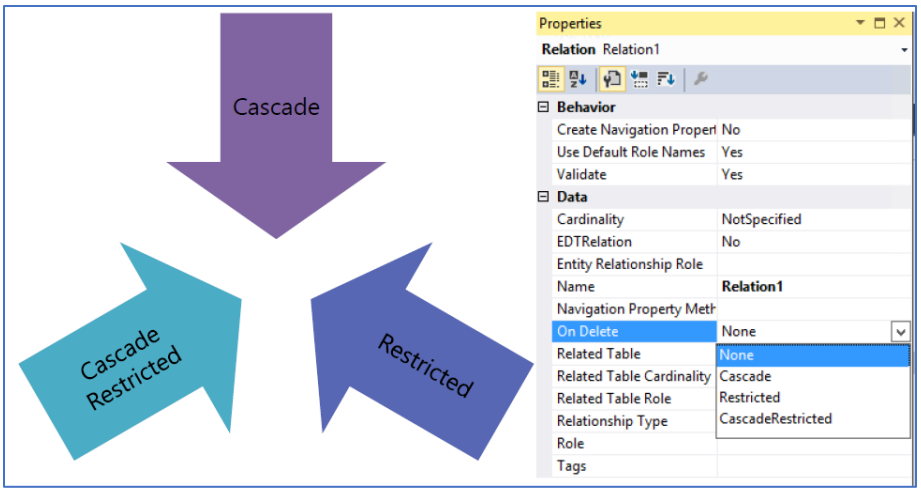
- **Related Field Fixed Relation** - used to specify field restrictions in a related table. Like **Field Fixed Relations**, it is usually used in conjunction with other relation types.
  1. In the **Related Field** property, we select the field in the related table which we want to restrict.
  2. In the **Value** property, we enter the filter value of the selected field. This causes only records in the related table that match that field value to be related.

- **Foreign Key Relation -** used to specify a correspondence between a foreign key field in one table to the primary key field in its parent table.
  We often use the term *child* to refer to a table that has a foreign key column, and we use the term *parent* to refer to the other table that supplies the value for the foreign key column.

  1. Set the **Table** property to the name of the parent table, the table that contains the primary key field.
  2. Set the **RelatedTableRole** property to a word or phrase that describes the purpose of the parent in the relationship.
  3. Set the **Name** property. A helpful value is a combination of the **Table** property and **RelatedTableRole** property values.
  4. Right-click the node for your relation, click **New**, and then click **ForeignKey**. Next click either **PrimaryKey** based or **Single** field **AlternateKey** based. A new field is instantly added to the child table. This field stores the foreign key values.

5. Under the **Fields** node, click the new field, and then change its Name property value.
6. For performance benefits, you might decide to add an index on the new foreign key field.

ON DELETE PROPERTY



None: The row in the parent table is deleted but nothing occurs in the child table that relates to this table.

Cascade: The row in the parent table is deleted and all relating data in the child table is also deleted.

Restricted: The row in the parent table is only deleted if there is nothing that relates to it in the child table.

Cascade + Restricted: Enables a cascade delete even though related records exist in child tables. A warning appears; if we click No, no deletion occurs; if we click Yes, the cascade deletion runs.

Lesson 2: Create Table Relations

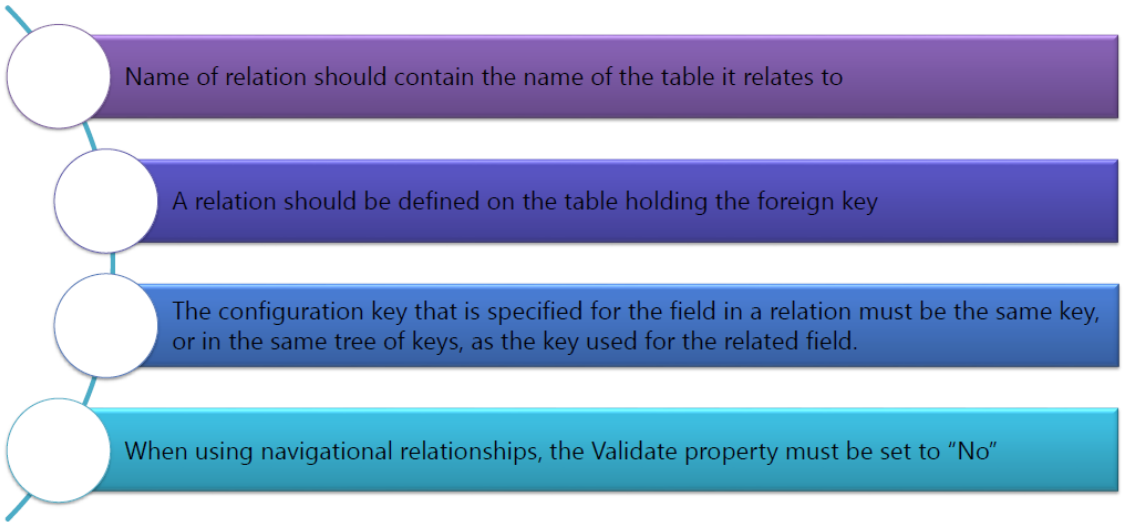
1. Create a table relationship.
   a. In the Solution Explorer, locate **DevelopmentBasicsFMSProject.**
   b. Right-click **dbfmVehicleMaintenance** and select **Open**.
   c. In the designer, right-click the **Relations** node and select **New > Foreign Key Relation**.
2. Set the properties.
   a. Right-click **RelationForeignKey1** and select **Properties.**
   b. Set the following properties:

- Cardinality: **OneMore**
- EDTRelation: **Yes**
- Name: **VehicleForeignKey**
- Related Table: **dbfmVehicleTable**
- Related Table Cardinality: **Exactly One**
- Relationship Type: **Association**

c. Right-click **VehicleForeignKey** and select **New > Normal**.
d. Right-click **dbfmVehicleMaintenace == dbfmVehicleTable** and select **Properties**.
e. Select the following properties.
    - Field: **dbfmVehicleID**
    - Name: **dbfmVehicleID**
    - Related Field: **dbfmVehicleID**

2. Build the solution.
    a. In the Solution Explorer, right-click **DevelopmentBasicsFMSProject**.
    b. Select Build.
    Review the Output screen to review process.

Lesson 3: Best Practices



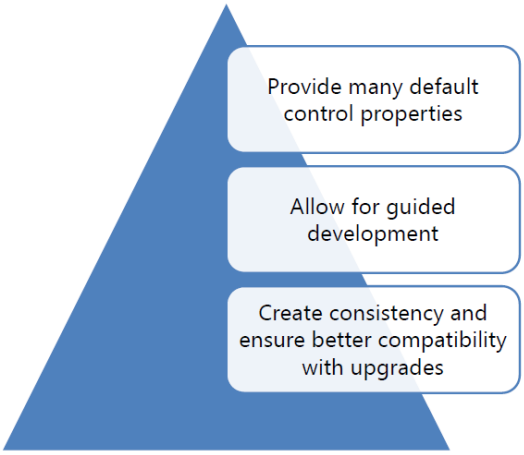
When you are using table relations:

- The name of the relation should contain the name of the table it relates to, as this makes it easy to identify.
- A relation should be defined on the table holding the foreign key, which is typically the child table. Additionally, the validate property on the relation must be set to Yes.
- When using navigational relationships, the Validate property must be set to "No". This is because navigational relationships do not enforce integrity constraints and are used for navigational purposes only.
    - For example, when a relation is specified for a form to open within another form, the validate property on the relation should be set to no. Defining them makes the application easier to navigate.

Module 09: Form Patterns

Lesson 1: Form Patterns and Sub-Patterns

ABOUT FORM PATTERNS

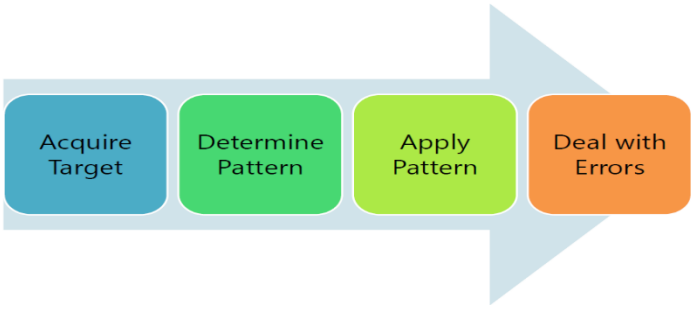


Form patterns are applied to forms and determine the specific layout as well as the controls and containers that are required for that particular pattern.

Sub-patterns are applied to certain controls or tabs on the form.

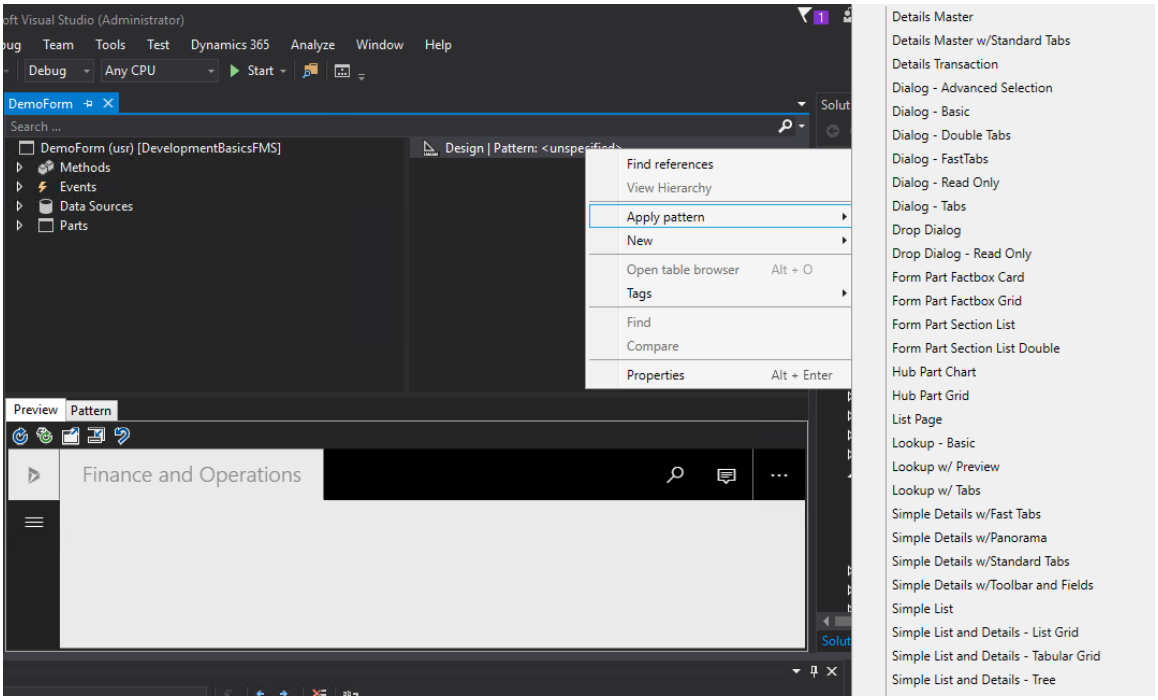
- Microsoft Dynamics 365 for Finance and Operations, Enterprise Edition, makes development easier by providing a guided experience for applying patterns to forms to ensure they are correct and consistent. They help validate forms and control structures and the use of controls in certain places. Using patterns also ensures that each new form encountered by a user is immediately recognizable in appearance and function. Form patterns can provide many default control properties, which leads to a more guided development experience. The use of form patterns also ensures better compatibility with upgrades.

STEPS FOR APPLYING PATTERN



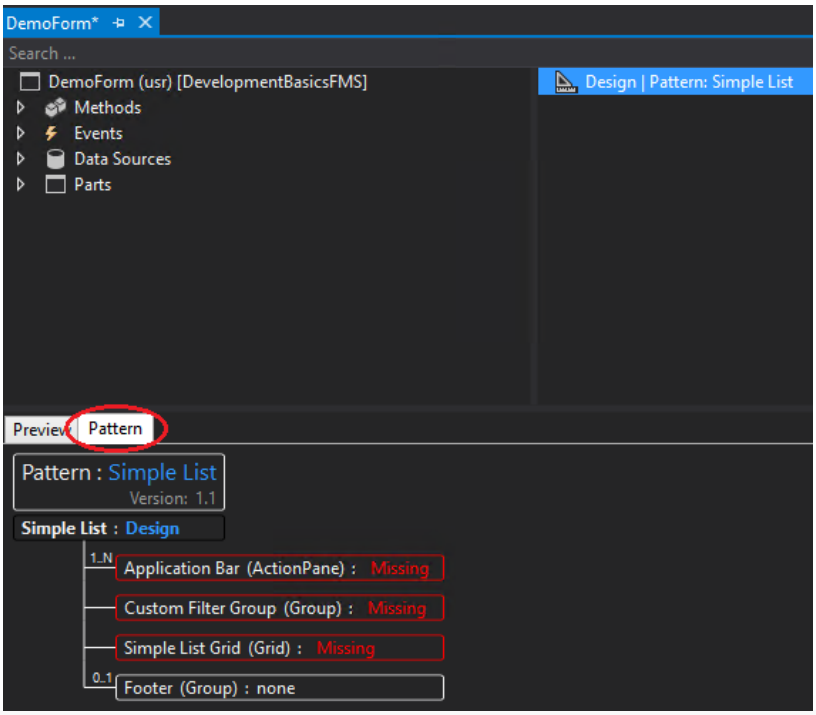
1. Specify form to use or apply target to.

2. Determine pattern to be use – right click on Design node → Click Apply Pattern → Select pattern



3. Apply desired pattern on form

4. Create controls based on form pattern requirements. All controls needed for the specified form pattern will be seen on Pattern tab.



a. From here you can easily start creating groups, controls, etc. that match up to the pattern per design document
b. Make sure to apply the sub-patterns appropriately as well

FORM PATTERN TYPES

1.**Details Master** - used for view, edit and also for entering data. Typical characteristic of this pattern are the fast tabs. There is the basic version and with standard tabs (>15 fasttabs). User can switch between, header and line view. Details master is typically used for display of masterdata for example CustTable, VendTable or product variants (EcoResProductVariants) forms.

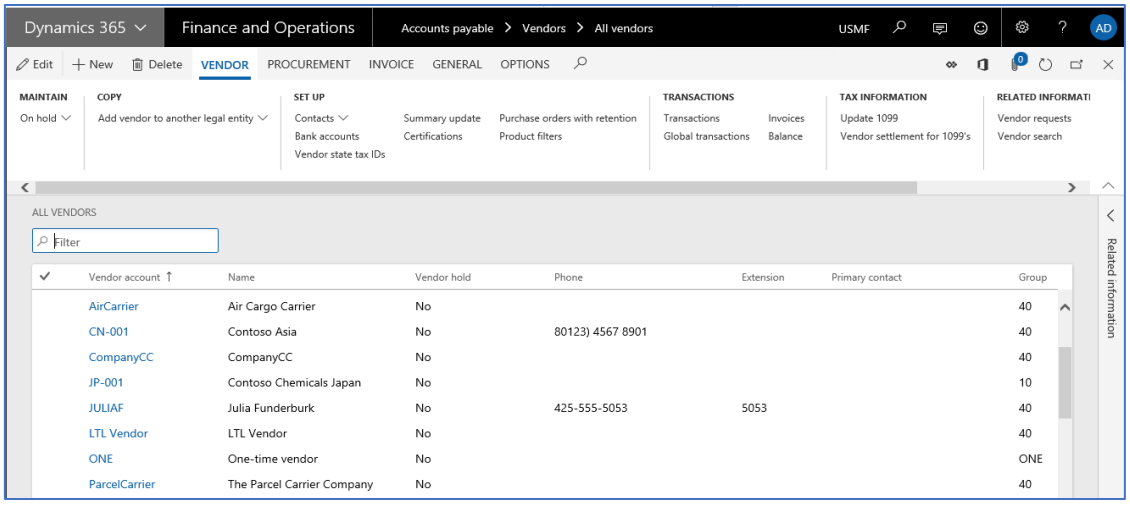


Figure 6 - Detail Master Grid View

2. **Form Part Fact Boxes**

- These are used to provide related information for a record
- This is to ensure that user does not have to open additional forms to get important information that they need such as totals, balances, or overdue orders.
- FactBox grid pattern is used when there is a child collection or potential for multiple rows of related information
- FactBox is created as a separate form with the FactBox pattern applied and then added as a part on another form.
- Types of FactBox
  - FactBox grid
    - Used when there's a child collection with the potential for multiple rows of related information
  - FactBox card
    - Used when there's simply a set of related fields

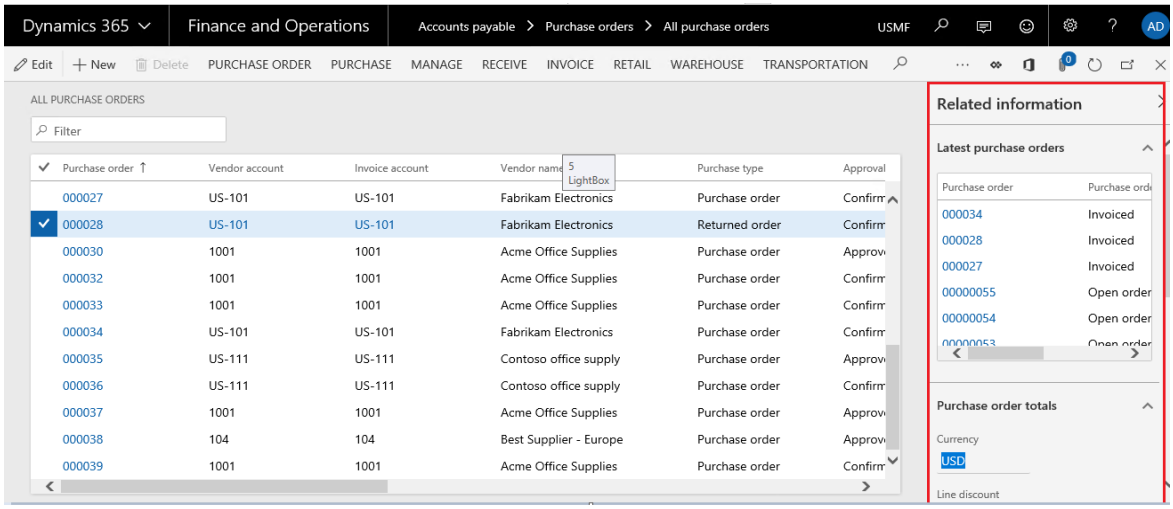


Figure 7 - FactBox

3. **Simple List**

- Displays details for a simple entity as a grid
- Typically has six fields or less per record with no child parent relationship

- List page is another type of form pattern which still displays data in a grid format

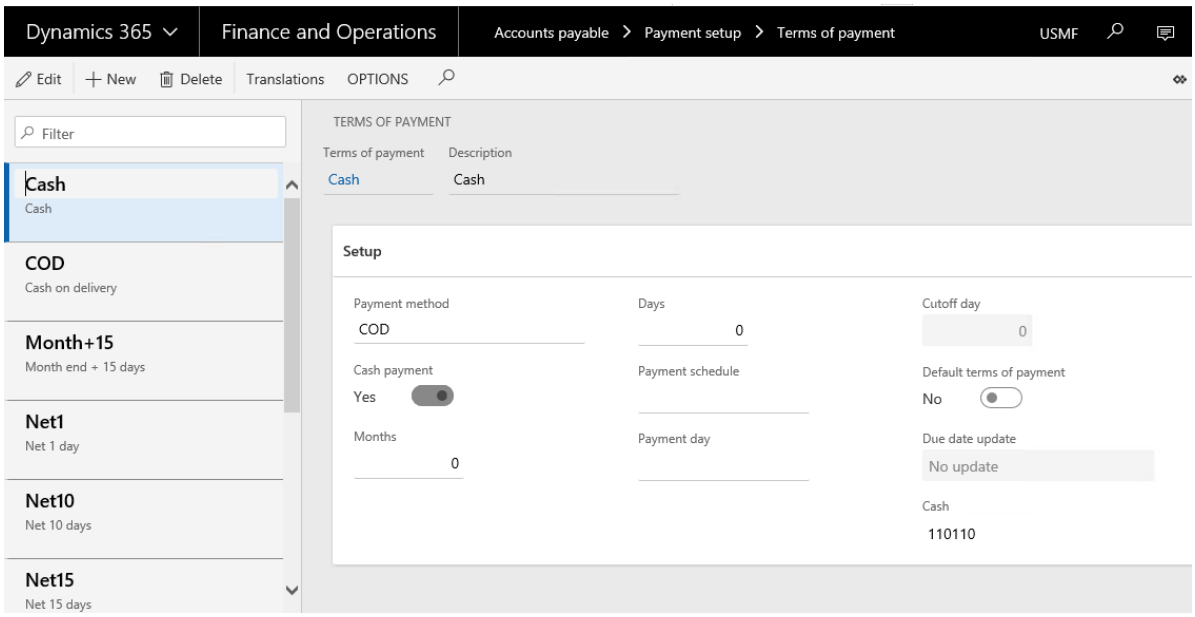


Figure 8 - Simple List

4. **Table of Contents**
   - This pattern should be used when two or more logically related forms are needed for a configuring setup
   - There is a vertical arrangement of tabs on the left which implies the order of completion.

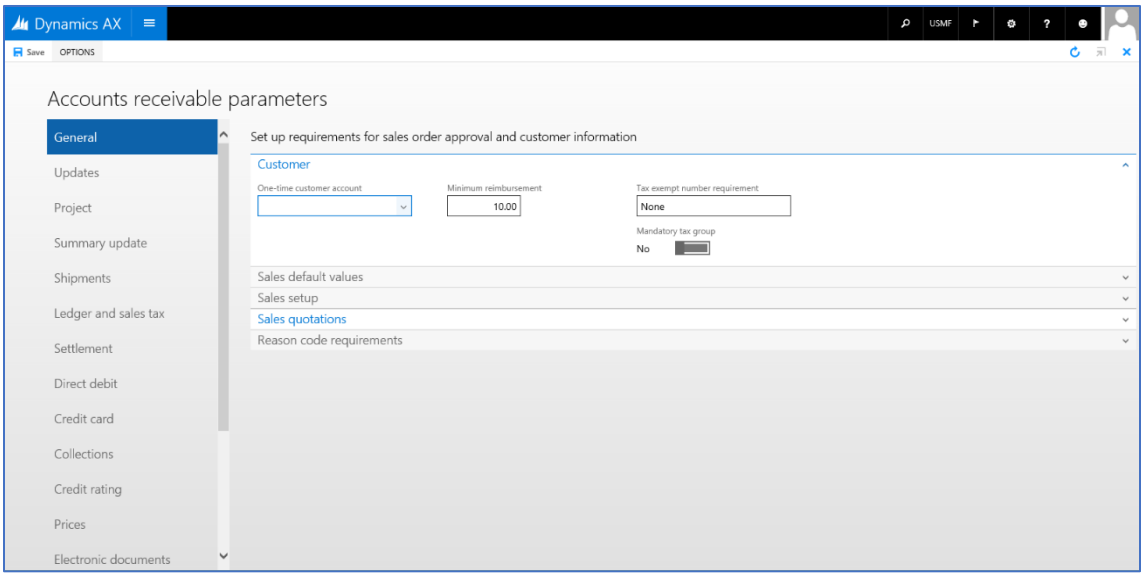


Figure 9 - Table of Contents

5. **Operation Workspaces**
   - Workspaces are a new concept that takes over as a primary way to navigate to tasks and specific pages.
   - This must be created for every significant activity supported in the product.
   - This must provide a one-page overview of an activity to help users understand the current status, the workload ahead, and the performance or of the process for the user

- Types:
    - Traditional workspace
    - Operational workspace

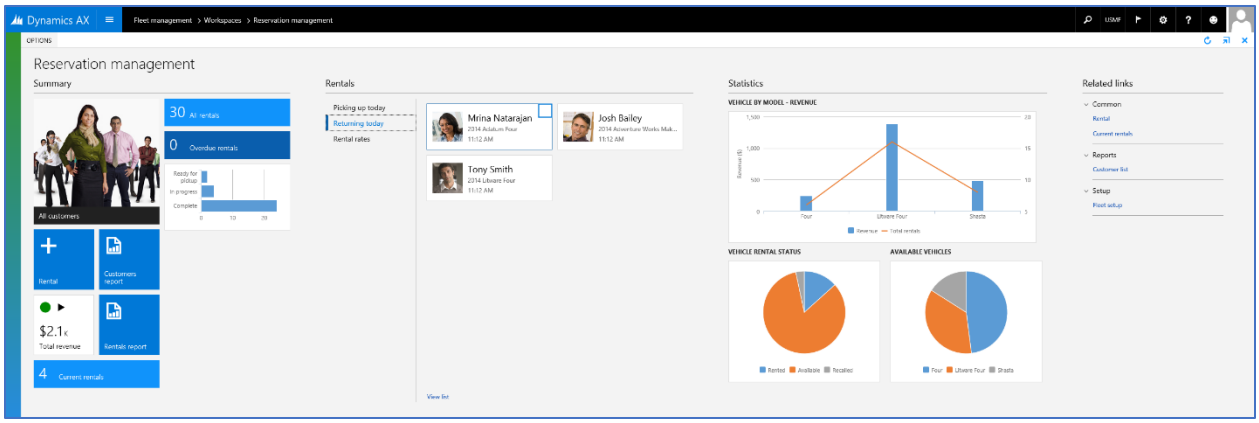
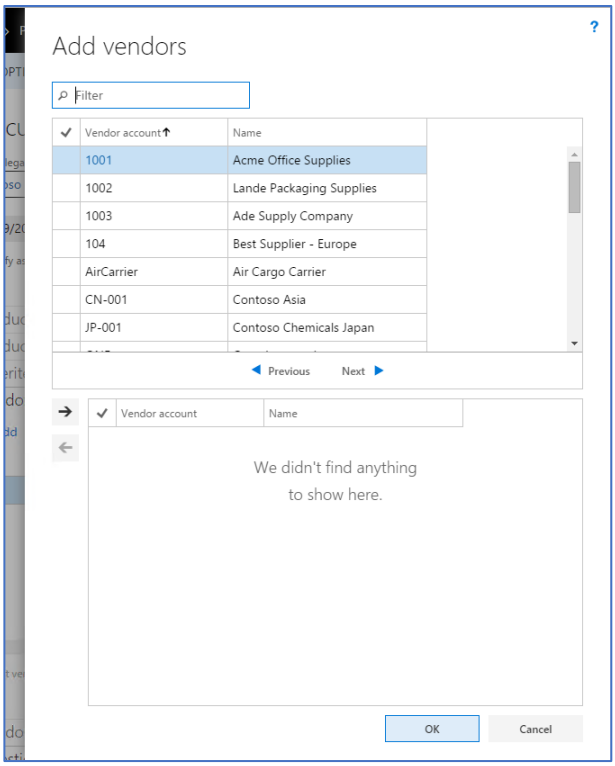


Figure 10 - Operational Workspace

SECONDARY FORM PATTERNS

1**. Advanced Selection form pattern** should be used when the primary user task is to select a set of items. This task is usually accomplished through a multi-select list. This pattern resembles the List panel pattern, in that the user selects items in one list and adds them to another. However, this pattern allows for custom filters and a “wide” list on top and uses most of the screen "real estate" of the page (typically, it's a Large dialog). Use this pattern when a user must be able to filter and select in a large, wide list.



3. Dialog Form Pattern

A **dialog box** represents an action or activity that users can explicitly commit or cancel. It's used when a user initiates a specific task or process, and the system requires user input about how or whether to proceed. Dialogs are modal and require that users interact with the controls in the dialog before they can return to the parent page. Dialogs also can have multiple sizes. Selection of a dialog size is subjective, and will vary, depending on the form elements that you've modeled on the dialog. The sizes are as follows:

- **Small** – This size is a one-column-wide dialog. If your dialog contains a relatively small amount of content (all simple fields, and no wide tables or other wide elements), you can probably use this size.
- **Medium** – This size is a two-column-wide dialog. If your dialog contains more content than can comfortably fit within a small dialog, but a full-width dialog isn't required, you should use this size.
- **Large** – This size is a three-column-wide dialog. If your dialog contains more content than can comfortably fit within a medium dialog, but a full-width dialog isn't required, you should use this size.
- **Full** – A large dialog is nearly the full width of the browser viewport. Its size varies, depending on the viewport width, and it will always be the largest dialog size option. Use this size if your dialog has a lot of wide elements, or if it requires an unusually large amount of horizontal space.

Five patterns are described in this document:

- **Dialog** – This is the basic dialog pattern. Use this dialog if you don't have a reason to use one of the other Dialog patterns.

  PATH: ProjTableCreate (Click Project management and accounting > Common > Projects > All projects, and then click New.)

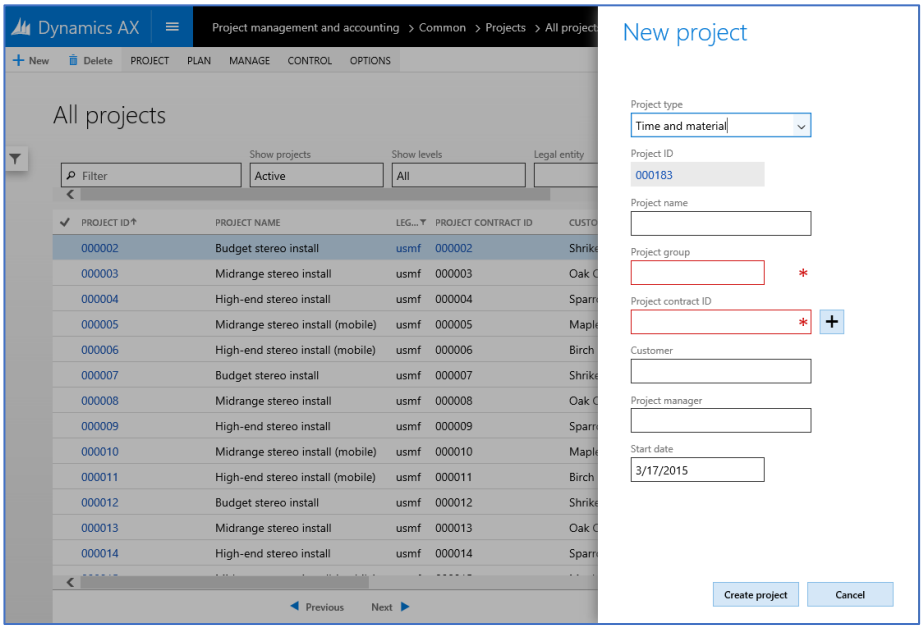


Figure 11 - Dialog (Basic)

- **Dialog w/tabs** – This is a more specific version of the Dialog pattern. It incorporates a Tab control in the dialog. You can also optionally provide a header for the Tab, and also a footer.

  PATH: CaseDetailCreate (Click Common > Common > Cases > All cases, and then click New.)

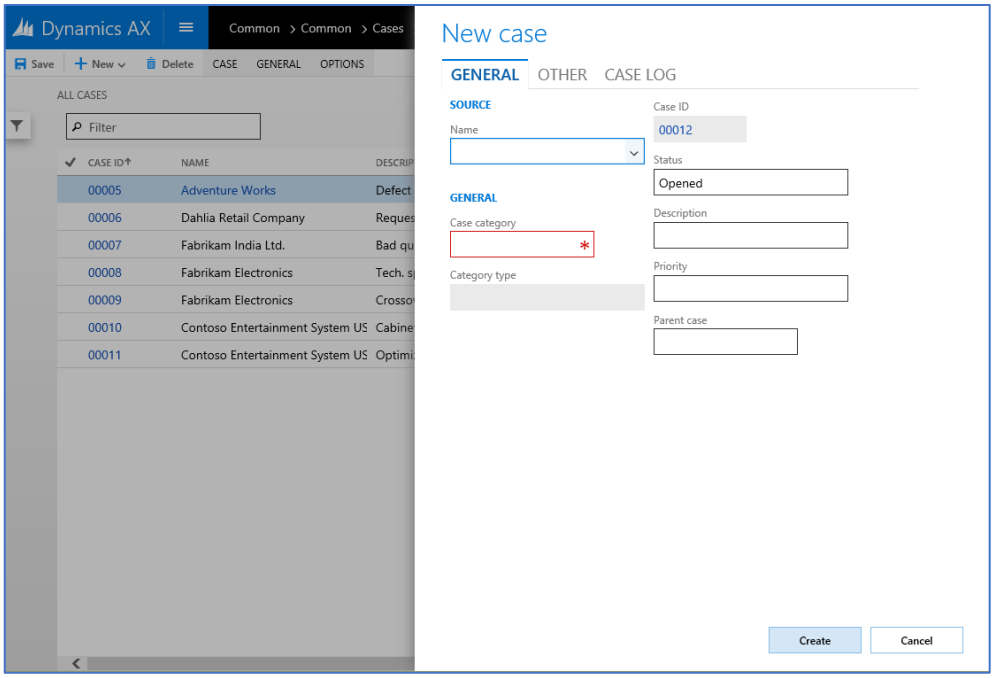


Figure 12 - Dialog w/ tabs

- **Dialog w/FastTabs** – This closely resembles the Dialog w/tabs pattern but uses FastTabs instead of regular tabs to organize the information.

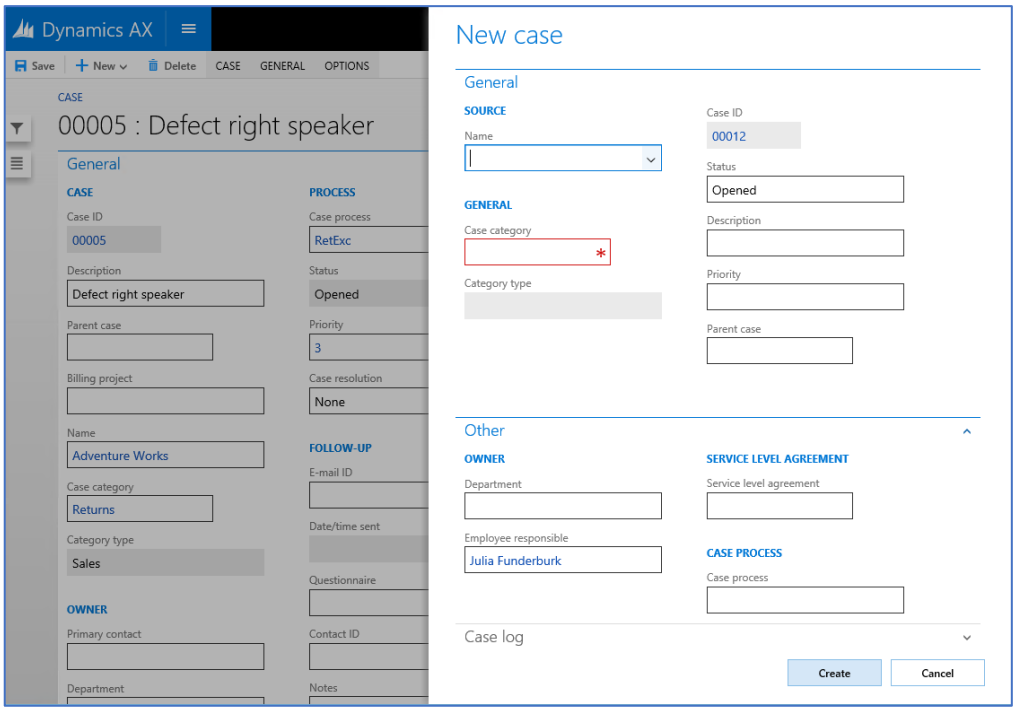


Figure 13 - Dialog with Fast tab

- **Dialog w/double tabs** – This closely resembles the Dialog w/tabs pattern but has a second Tab control immediately after the first one.

  PATH: PurchTableReferences (Click Accounts payable > Common > Purchase orders > All purchase orders**,** and then click General > Related information > Related orders.)

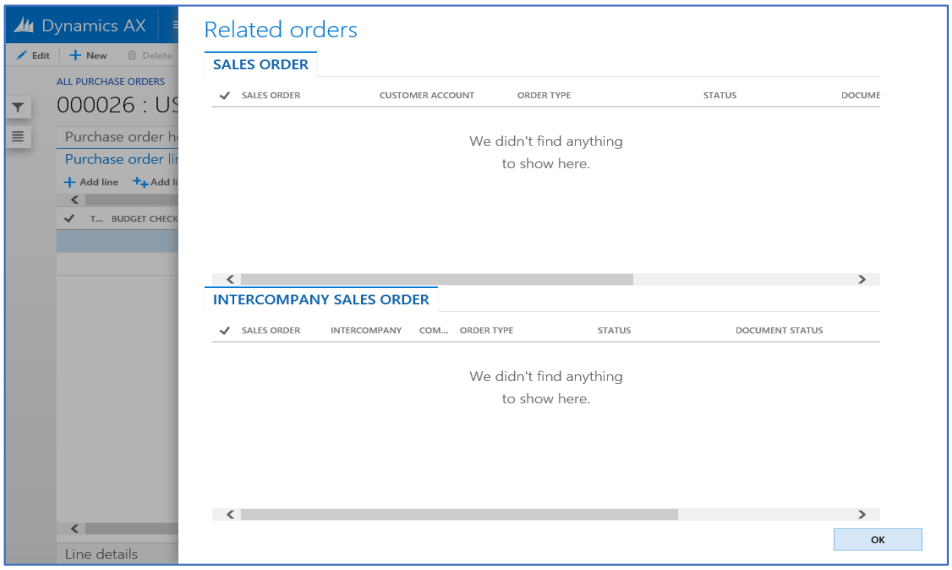


Figure 14 - Dialog w/ double tabs

- **Dialog (read only)** – This pattern is for informational forms that aren't editable. The user can still switch between tabs or a view selector, but direct manipulation of input fields isn't allowed. This dialog variation also includes a **Close** button instead of **OK** and **Cancel** buttons.

  Form: SalesTablePostings (Click Accounts receivable > Common > Sales orders > All sales orders, and then click General > Related information > Postings.)

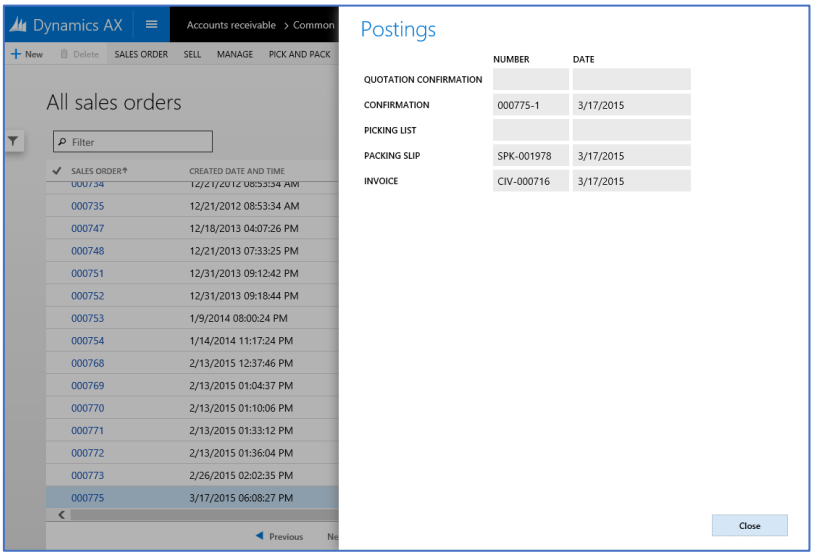


Figure 15 - Dialog (Read Only)

LOOKUP FORM PATTERNS

Custom lookup forms should be used when a standard framework-provided lookup (which is typically generated by using the AutoLookup field group that is defined on the table definition), would not provide the correct data, or when advanced visualization of the data is required. Three patterns are described in this document:

- **Lookup basic** – This is the basic Lookup pattern that has just one list or tree, and also optional custom filters and actions.

  Form: SysLanguageLookup (Click Settings > User settings on the navigation bar.)

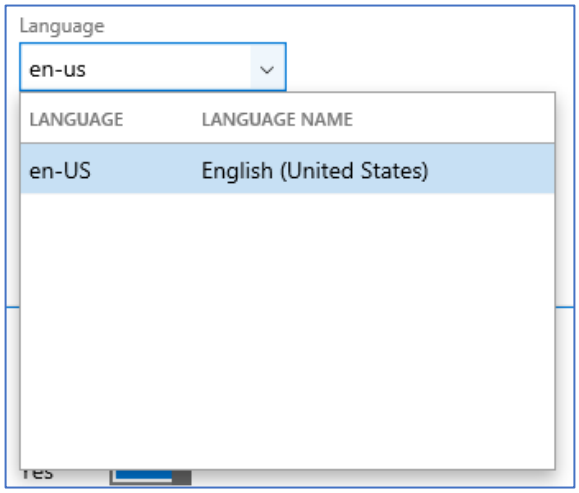
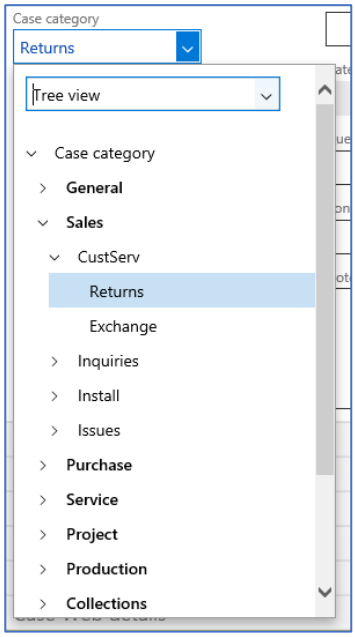


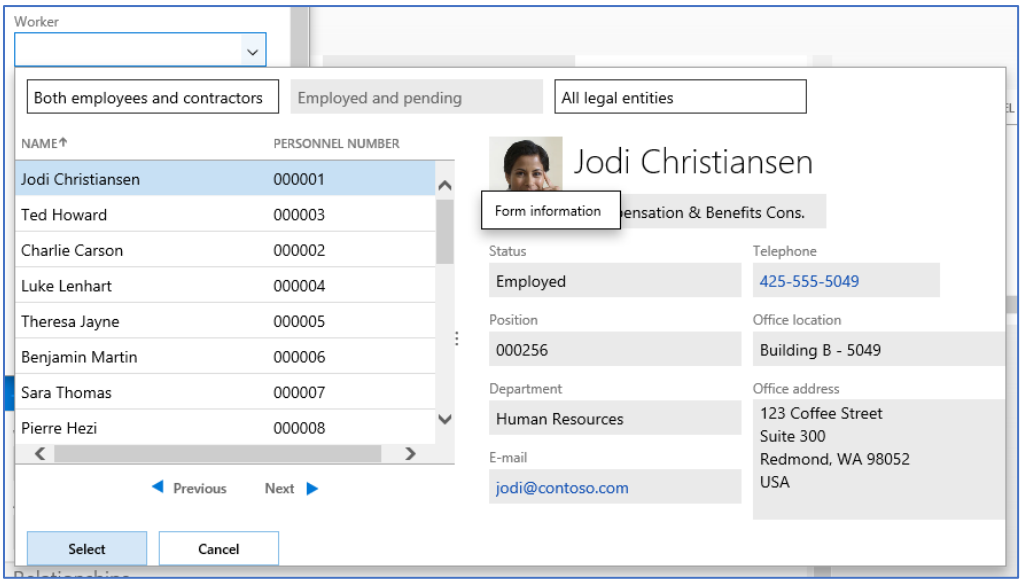
Figure 16 - Lookup basic

- **Lookup w/tabs** – This Lookup pattern is used when more than one view of the lookup can be made available to the user. Tab captions aren't shown. Instead, the tab is selected through a combo box.

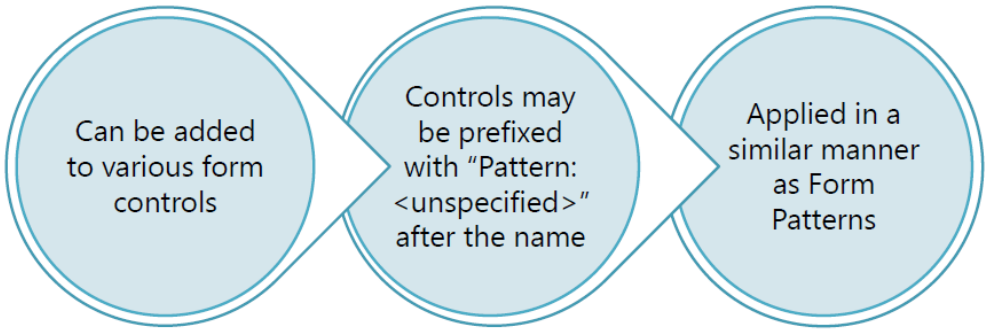
  Form: CaseCategoryLookup (Click Common > Common > Cases > All cases, and then select a case to go to the details.)



- **Lookup w/preview** – This more advanced Lookup pattern enables a preview of the current record in the lookup grid.



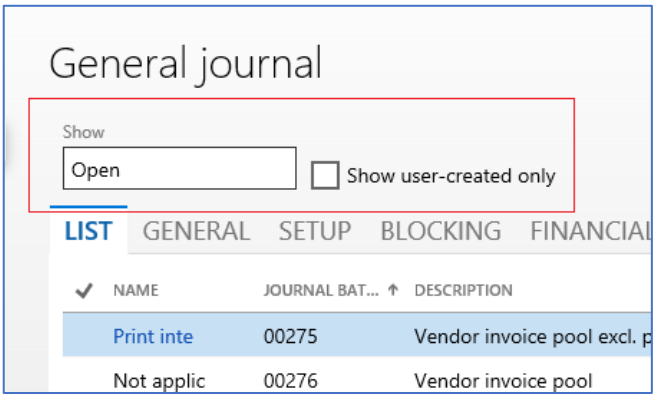
SUB-PATTERNS



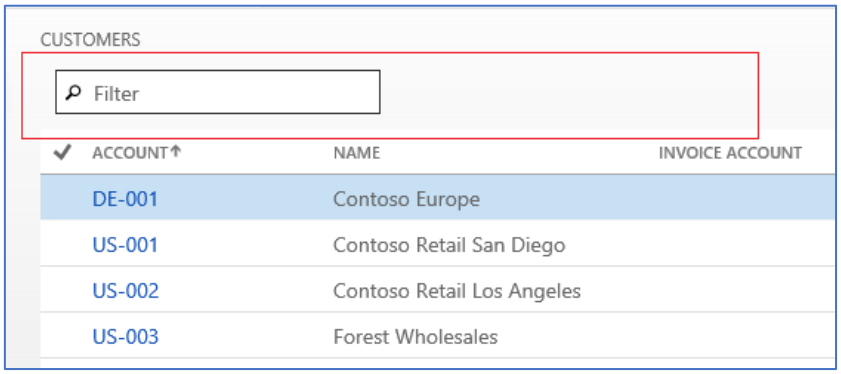
SUB-PATTERN TYPES

- **Custom Filter Group sub pattern** - used to show a small collection of input controls (no more than five) that apply a custom filter to a grid or form section. Fields in the Custom Filter Group should be limited to the following field types, which have constrained inputs and can be applied to the query:
  - StringEdits with Lookups
  - Date fields
  - ReferenceGroup
  - Comboboxes
  - Checkboxes
  - Quick Filter

- **Custom Filters** – In this subpattern, the QuickFilter control is optional.



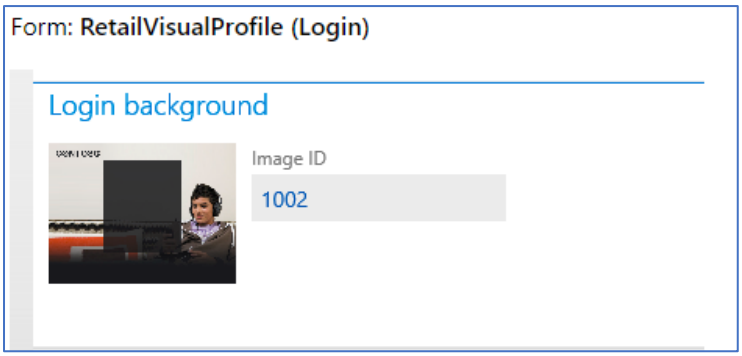
- **Custom and Quick Filters** – In this subpattern, the QuickFilter control is mandatory.



- **IMAGE PREVIEW** - used for most images that appear within a form container, especially within a FastTab or Group. This subpattern can be used in conjunction with the FieldsAndFieldGroup and FillText subpatterns to combine images and any associated fields. This subpattern isn't used for tiles or buttons, or for field status images.

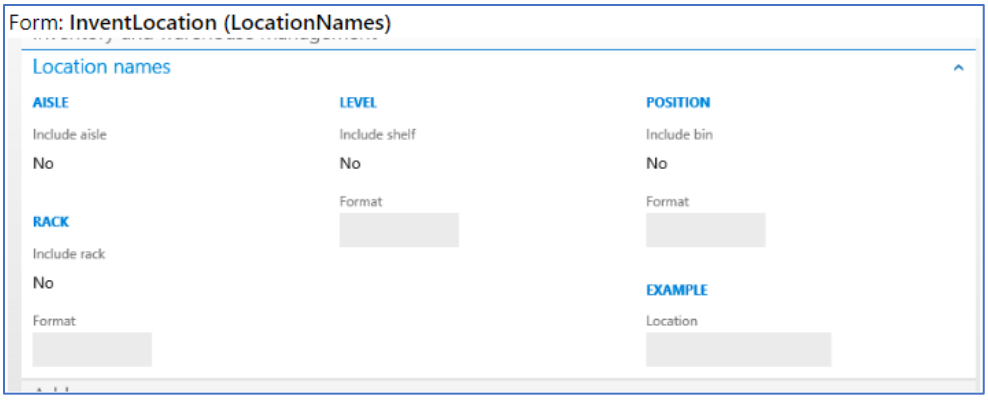
  Typical contents

  - Toolbar (ActionPane where **Style**=**Strip**)
  - Image
  - Can contain subpatterns:
    - Fields and Field Groups
    - Fill text

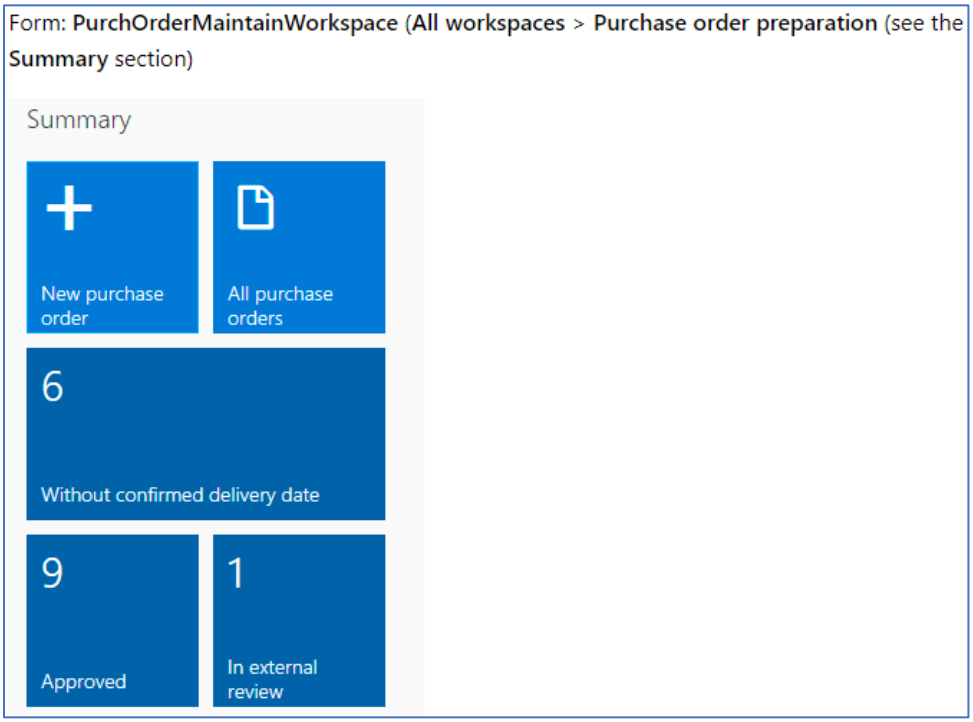


- **FIELDS AND FIELD GROUPS** - most common data entry subpattern and uses a dynamic number of columns to present multiple fields or groups of fields. This subpattern is not used with controls that have dynamic height or width (for example Grid, Tree, RadioButton, ListBox, or ListView), or controls that have larger height or width (for example, Chart). The group controls within this pattern can be used either to group fields under a label or to bind to a table field group.

  Typical contents
  - Groups or Fields as immediate children of the FastTab
  - Groups containing Fields
  - Can contain other subpatterns:
    - Horizontal fields and button group

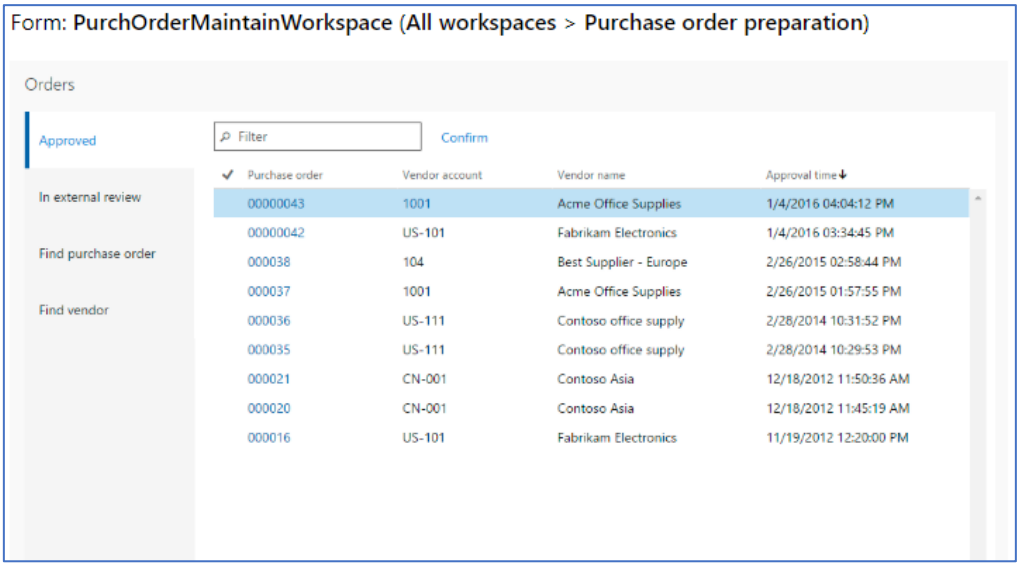


- **SECTION TILES** - used as part of the Operational Workspace pattern, specifically for the first panorama section (the **Summary** section) that contains a set of tiles, charts, and singleton cards.

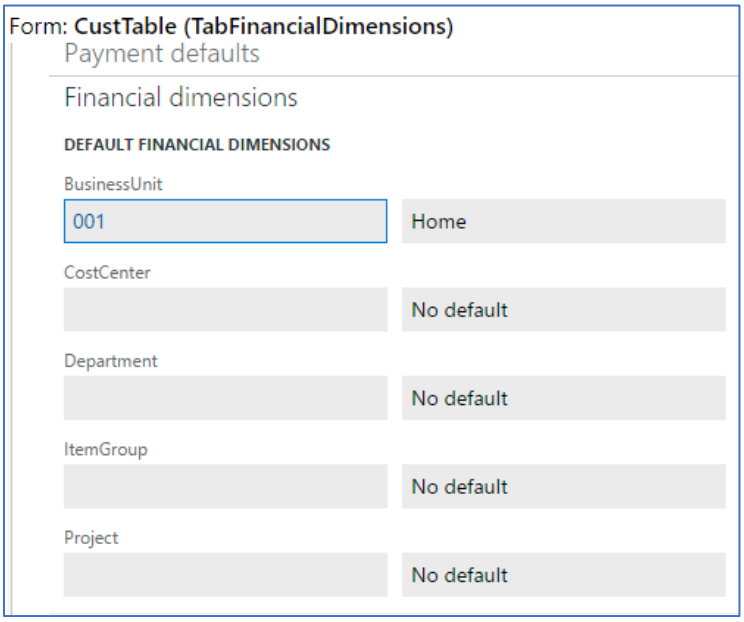


ADDITIONAL SUB-PATTERN TYPES

- **SECTION TABBED LIST** - used as part of the Operational Workspace pattern, specifically for a panorama section that contains a set of vertical tabs, each of which contains a filtered list of data.

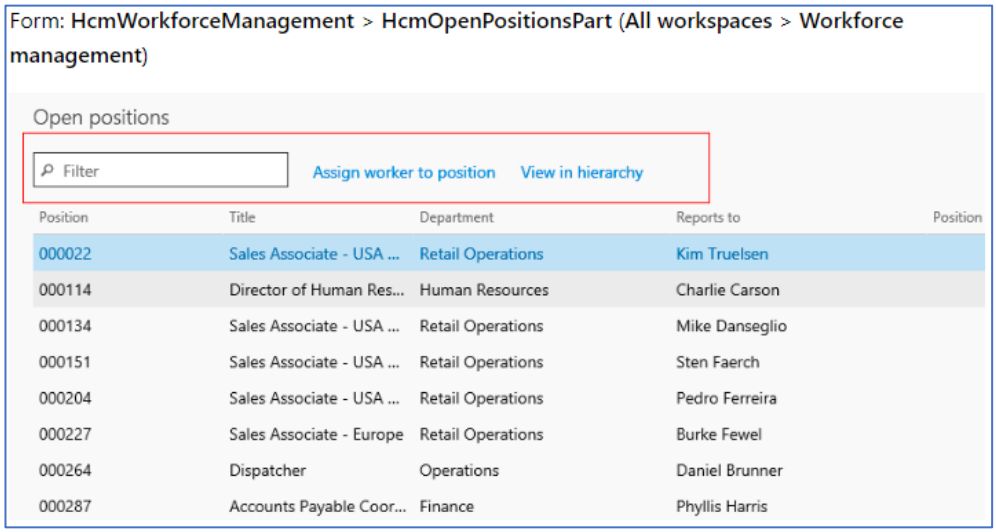


- **DIMENSION ENTRY CONTROL -** used when you have a group or tab page that uses the Dimension Entry control (DEC).

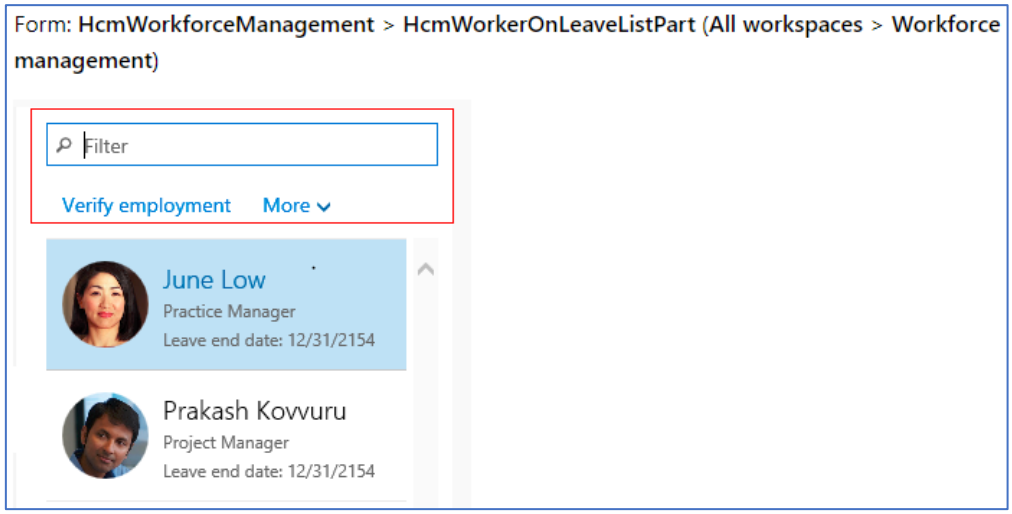


- **FILTERS AND TOOLBARS -** workspace-specific subpatterns that have been developed to show filters and/or actions inside panorama sections that host lists and charts. Fields in the filtering parts of these subpatterns should be limited to the following field types. All these field types have constrained inputs and can be applied to the query.

  - **Filters and Toolbar - Inline** – In this subpattern, any defined actions appear on the same line as the filter fields.



  - **Filters and Toolbar - Stacked** – In this subpattern, any defined actions appear on a separate line below the filter fields.



FORM PROPERTIES

Form Design View Edit Mode (Form.Design.ViewEditMode)

- **Auto**
  - Page opens in View mode, and has an Edit button that toggles to Done, Save, and Restore. Be aware that some page styles ignore the ViewEditModeproperty. For example, a dialog box doesn't use the ViewEditModeproperty.
- **View**
  - The page opens in View mode and has no buttons that are related to editing the page. The page is "always view."
- **Edit**
  - The page opens in Edit mode, and has Save and Restore buttons. The page is "always editable."

Form Root Data Source (Form.RootDatasource)

- AllowCreate
  - No
    - The page is effectively a read-only data source. It has no New and Delete buttons.
  - Yes
    - The page allows creates. It has a New button.
- AllowDelete
  - No
    - The page doesn't allow deletes. It has no Delete button.
  - Yes
    The page allows deletes. It has a Delete button

ABOUT FORM PATTERN TAB

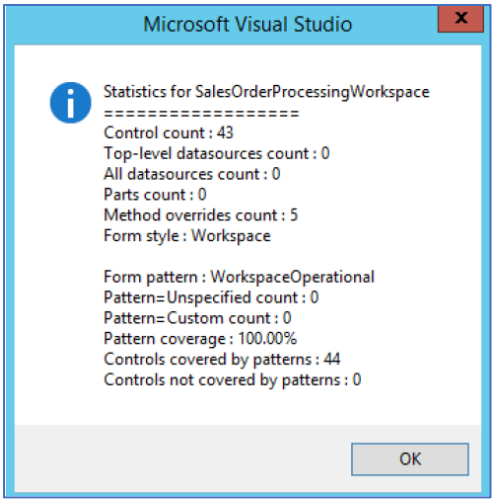
- Allows for guided development
- Ensures consistency
- Notifies user of errors and areas in need of correction

- This is a tab in Visual Studio that allows the developer to monitor the form development process and to make adjustments where needed

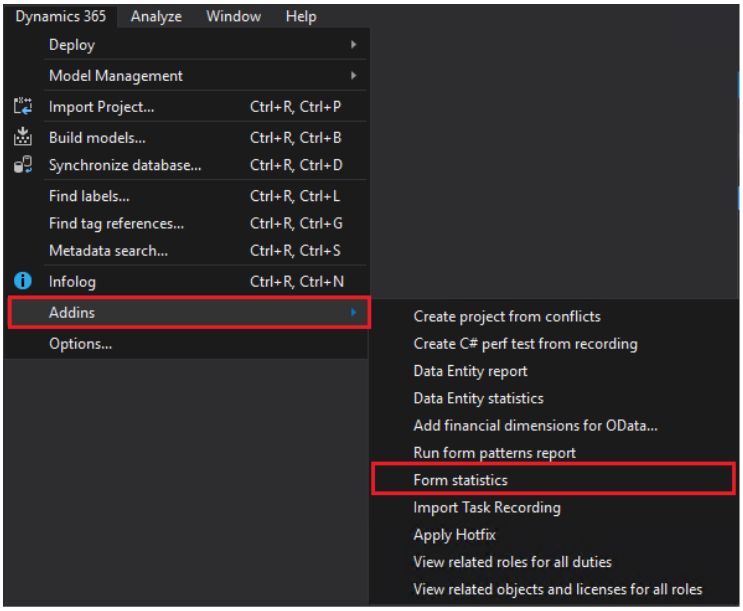
  It displays which controls are required and where they are needed to satisfy the requirements of the chosen form pattern.

FORM STATISTICS

Window that displays the summary of form statistics, form pattern name and percentage of controls that are covered by the pattern.



To generate this window, navigate to Dynamics 365 tab on Visual Studio → Click Addins → Select Form statistics



Module 10: Form Creations

Lesson 1: Create a form

Scenario

For the Fleet Management System, the user interface elements of the forms will need to be created. Following the standard, there will need to be full entry data pages for both the Vehicle and Vehicle Management tables. In addition, there will need to be a list page summarizing all of the Vehicles and link to the data pages.

Simon, the developer for Contoso, needs to create a list page form and supporting forms for the Vehicle and Vehicle Maintenance information.

This practice and Practice: Adding Elements will cover this scenario.

High Level Steps

1. Create the form.
2. Repeat for each additional form.
3. Build.

Detailed Steps

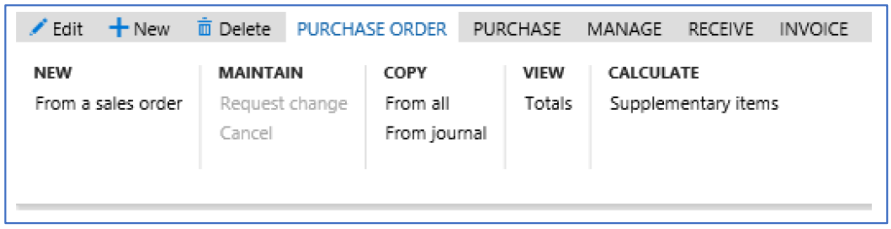
1. Create the form.
    1. In the **Solution Explorer**, right-click **DevelopmentBasicsFMProject** and select **Add > New item**.
    2. Under **Dynamics 365 Items,** select **User Interface**.
    3. Select **Form**.
    4. Type **dbfmVehicleDetails** in the **Name** field
    5. Click **Add**.
2. Repeat for each additional form.
    1. In the **Solution Explorer**, right-click **DevelopmentBasicsFMProject** and select **Add > New item**
    2. Under **Dynamics 365 Items,** select **User Interface**.
    3. Select **Form.**
    4. Type **dbfmVehicleMaintenanceDetails** in the **Name** field
    5. Click **Add**.
    6. In the **Solution Explorer**, right-click **DevelopmentBasicsFMProject** and select **Add > New item**
    7. Under **Dynamics 365 Items,** select **User Interface**.
    8. Select **Form.**
    9. Type **dbfmVehicleListPage** in the **Name** field.
    10. Click **Add**.
3. Build the solution.
    1. In the **Solution Explorer**, right-click DevelopmentBasicsFMSProject.
    2. Click **Build**.

FORM DESIGN LAYOUT

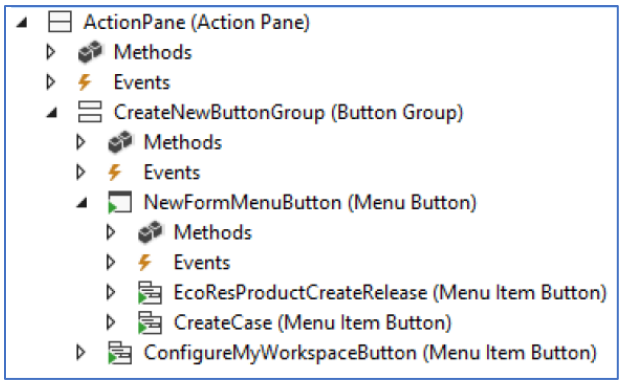
Container	Description
Form.Design	The root of the page. It functions as a special kind of container.
Group	The general-purpose container control in Microsoft Dynamics AX. Group controls can be nested as required.
Tab	A control that contains TabPage controls and has many possible **Tab.Style** values, such as **Tab**, **FastTab**, **Vertical Tab**, **Droplist** and **Panorama**.
TabPage	The appearance of each TabPage control depends on its **Tab.Style** value.
ButtonGroup	A special type of Group control that contains buttons.

ACTION PANE TAB

The action pane or application tab runs across the top of a form and contains buttons and tabs to perform certain actions within that form.



ACTION PANE HIERARCHY



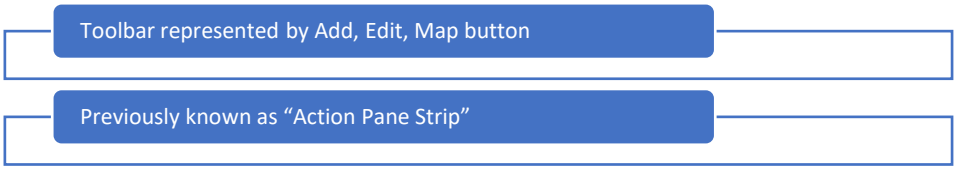
- Action Pane
  - ActionPaneTab1
    - ButtonGroup1
      - Button1
      - Button2
    - ButtonGroup2
      - Button3
  - ActionPaneTab2
    - ButtonGroup3
      - Button4

TYPES OF BUTTONS

Buttons are the foundation of action controls. They can be modeled inside of standard Action Panes or in Toolbars, which are discussed later in the article. They can also be added as stand-alone buttons on the page (for example, the **OK** and **Cancel** buttons at the bottom of a dialog box, or buttons for actions that are specific to an individual field). The following button types continue to be available:

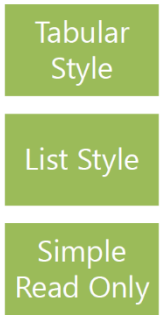
- A **button** is a basic button, for which the entire functionality must be implemented in code.
- A **command button** specifies a command or task to run.
- A **menu item button** specifies a menu item to navigate to or run.
- A **drop dialog button** opens a flyout dialog box, the contents of which are retrieved via a menu item.
- A **menu button** is a button container that opens a flyout that contains a list of other buttons.

TOOLBARS



Toolbars are action panes for which the style property is set to strip. They are used for actions that have specific context or aren't page-level actions and primarily for actions that are specific to a FastTab, tab, or grid.

GRID STYLES

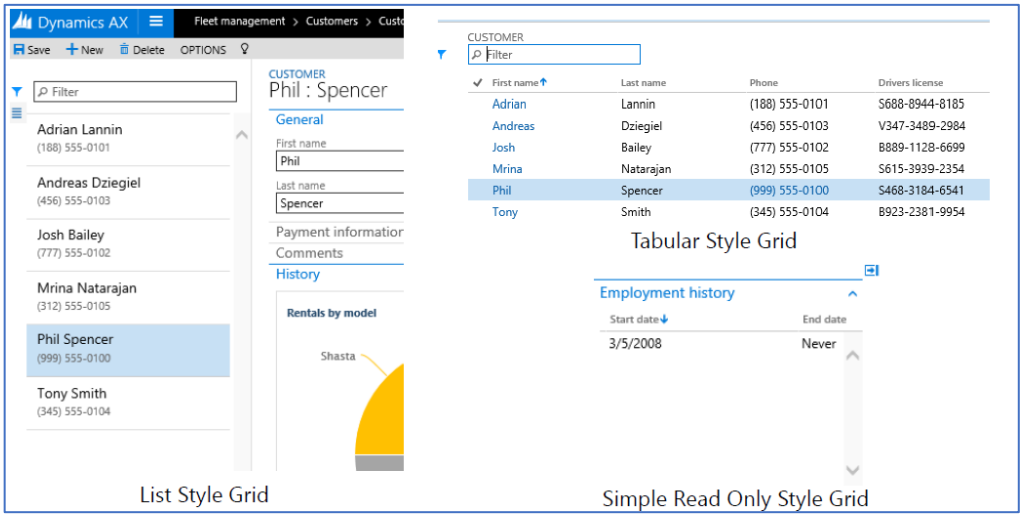


Property	Style
Style=Auto	Tabular grid
Style=Tabular	Tabular grid
Style=SimpleReadOnly	Simple read only
Style=List	List grid

Tabular grid is the traditional multigrid column. To present data in a tabular grid, you'll set the style property to either auto or tabular.

L**ist style** grid is a single column stacked format. To present data in a list grid, you'll set the style property to list. In a simple list grid when width mode is set to auto, the width of the grid is determined by the widest field. Other fields are arranged in a left-to-right stacked manner.

SimpleReadOnly grid which displays read-only data in a grid format. It is often used on form parts such as FactBoxes or info parts that display a small bit of read-only information.



FORM DESIGN: BEST PRACTICES

Module 11: Menus

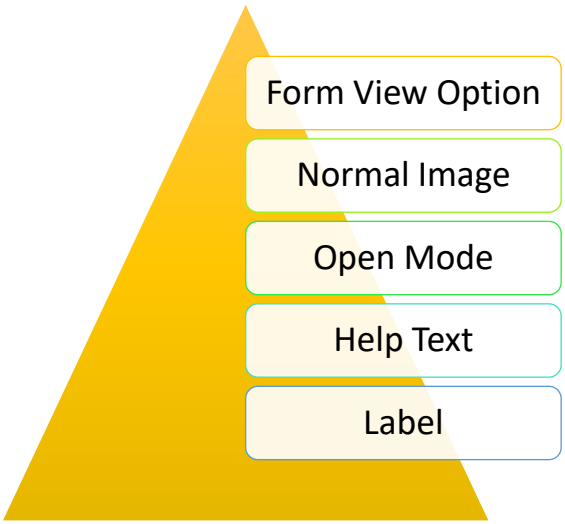
Lesson 1: Menu Items

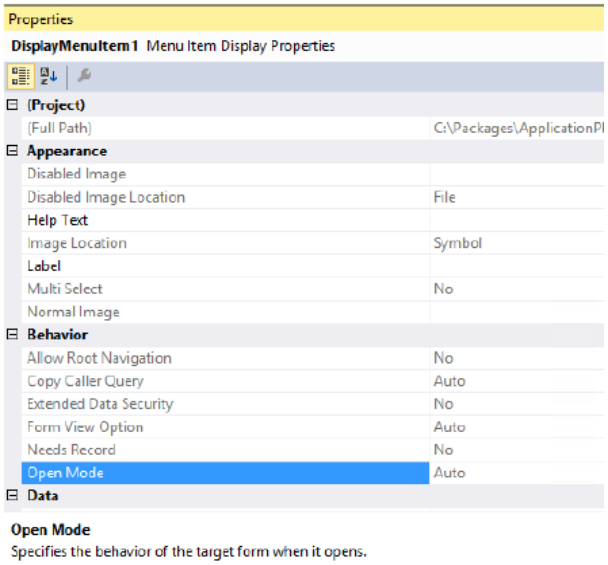
Display menu item - used to reference runnable objects like forms and dialogs. The basic purpose of these type of menu items is that they are used to present data

Output menu item - main purpose of this type of menu item is to print out the results of a query, etc. As such these menu items are mostly used for referencing classes.

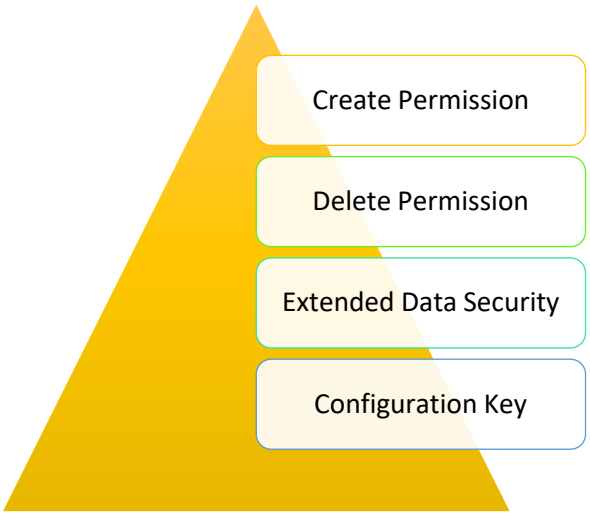
Action menu item - used in cases where you need to perform some action based on a menu item click. For instance, where you need to run a functionality in a class with a single click, you'll use the Action menu item.

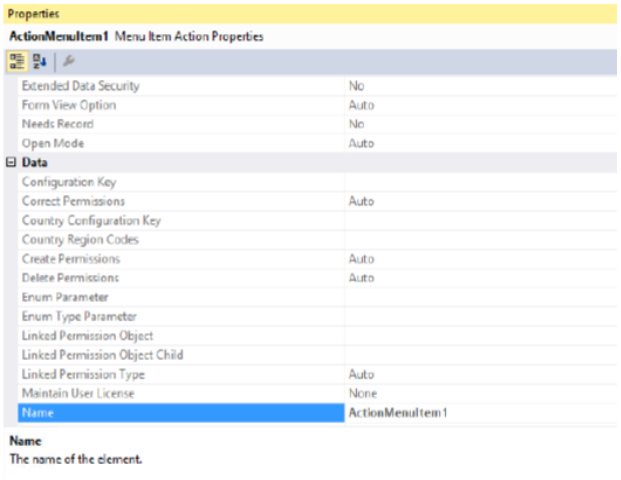
Key Property Values on Display Menu Items



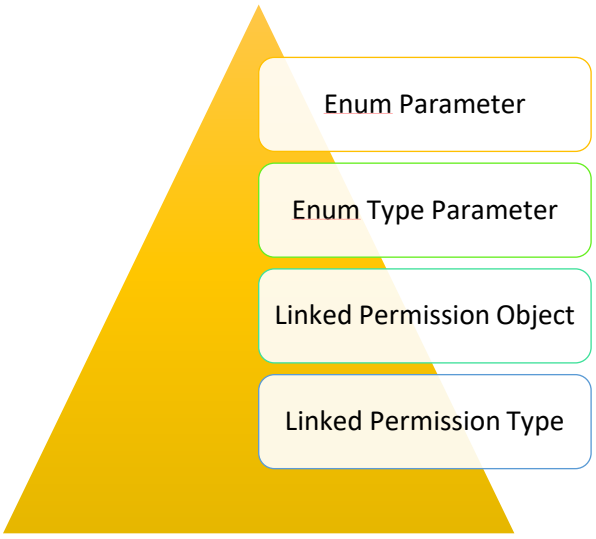


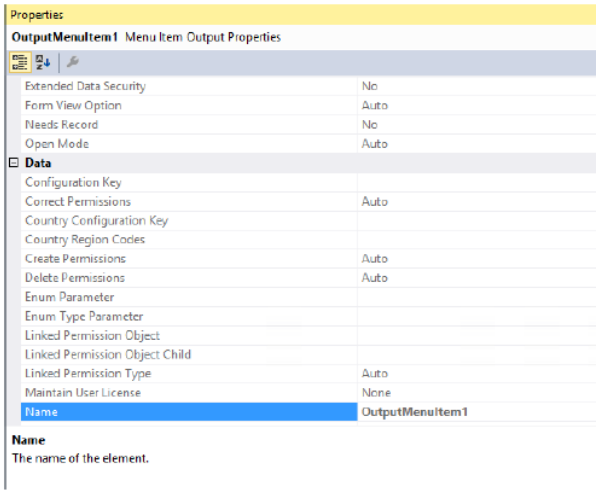
Key Property Values on Output Menu Items





Key Property Values on Action Menu Items





Create Menu Items

Scenario

For the Fleet management scenario, we will need to build menu items that point to the newly created forms. Then we will build a menu to contain the top level form, the List page, and then link all of the menus using Menu Extension to the existing menu structure of Microsoft Dynamics 365 for Finance and Operations. Lastly, we will create menu item buttons for the Vehicle and Vehicle Maintenance forms. This practice and Practice: Create a Menu will cover this scenario.

High Level Steps

1. Create a menu item.
2. Add menu item parameters.
3. Create additional menu items.
4. Build.

Detailed Steps

1. Create a menu item.
    3. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject**.
    4. Click **Add > New Item**.
    5. Under **Dynamics 365 Items**, click **User Interface.**
    6. Click **Display Menu Item.**
    7. Enter **dbfmVehicleDetails** in the **Name** field.
    8. Click **Add.**
2. Add menu item parameters.
    9. Right-click **dbfmVehicleDetails** in the middle pane and select **Properties**.
    10. Set the **Label** property to **Vehicle Details**.
    11. Set the **Object** property to **dbVehicleDetails**.
    12. In the **Solution Explorer**, right-click **dbVehicleDetails** under the **Display menu items** folder and select **Set as startup object.**
        Click **Start** on the menu. The newly built form will open in Internet Explorer.
    13. Close Internet Explorer.
3. Create additional menu items.
    14. On the far right of the page, click the **Solution Explorer** tab.
    15. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject**.
    16. Click **Add > New Item.**
    17. Under **Dynamics 365 Items,** click **User Interface.**
    18. Select **Display Menu Item.**
    19. Enter **dbfmVehicleMaintenance** in the **Name** field.
    20. Click **Add**.
    21. Right-click **dbfmVehicleMaintenance > Properties.**
    22. Set the **Label** property to **VehicleMaintenance**.
    23. Set the **Object** property to **dbVehicleMaintenanceDetails.**
    24. In the **Solution Explorer**, right-click **dbVehicleMaintenance** under the **Display Menu Items** folder and select **Set as startup object.**
    25. Click **Start.** The newly built form will open in Internet Explorer.
    26. Close Internet Explorer.
    27. On the far right of the page, click the **Solution Explorer** tab.
    28. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject** and select **Add > New Item.**
    29. Under **Dynamics 365 Items**, click **User Interface.**
    30. Select **Display Menu Item.**

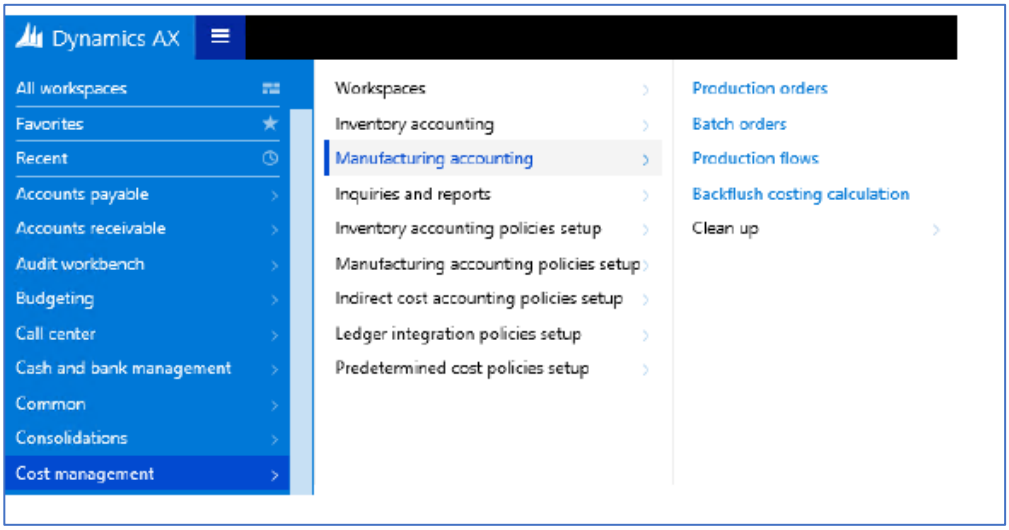
31. Enter **dbfmVehicleListPage** in the **Name** field.
32. Click **Add**.
33. Right-click **dbfmVehicleListPage > Properties.**
34. Set the **Label** property to **VehicleListPage.**
35. Set the **Object** property to **dbVehicleListPage**.
36. Right-click the **dbVehicleListPage** menu item in the **Solution Explorer** and select **Set as startup object**.
37. Click **Start**. The newly built form will open in Internet Explorer.
38. Close Internet Explorer.

4. Build.

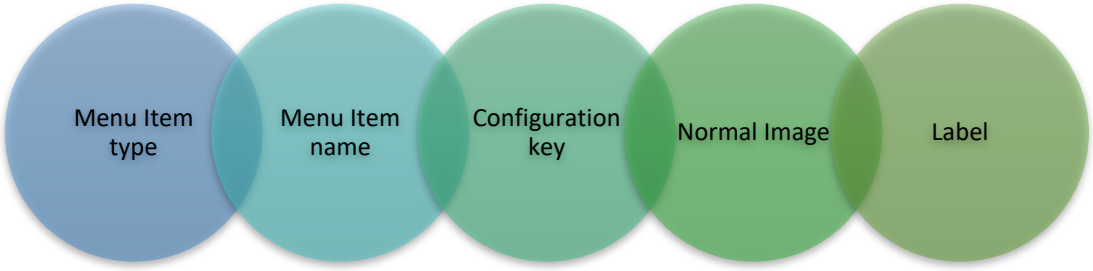
1. On the far right of the page, click the **Solution Explorer** tab.
2. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject**.
3. Click **Build**.
4. Review the Output screen to review process.

Lesson 2: Menus

Menus are prebuilt to include all of the accessible items in AX from the navigation pane



KEY PROPERTY VALUES OF MENU



Create a Menu

Scenario

This practice is a continuation of the previous practice.

High Level Steps

1. Create a menu.
2. Set the properties.
3. Extend the menu.
4. Set the properties.
5. Build.
6. Update forms with menus.
7. Build.

Detailed Steps

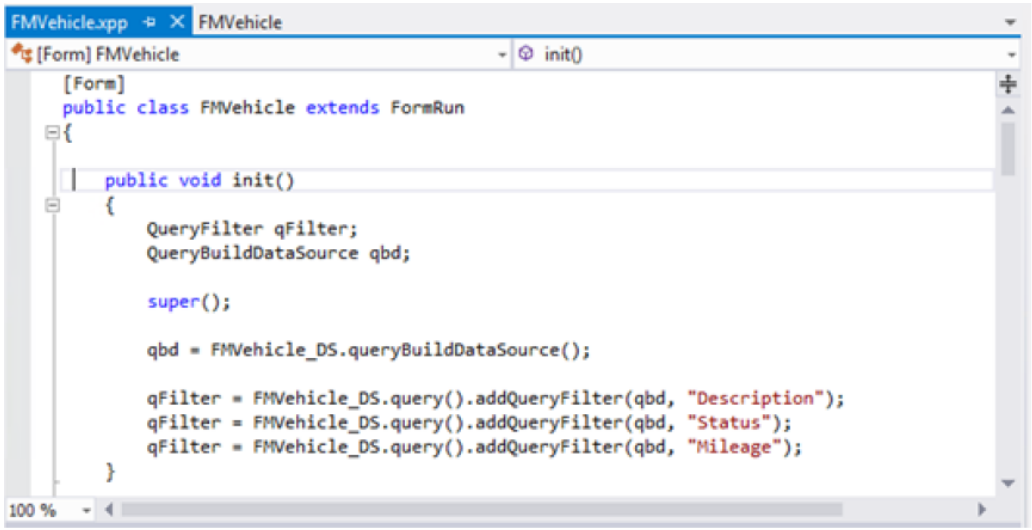
5. Create a menu.
    a. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject**.
    b. Click **Add > New Item**.
    c. Under **Dynamics 365 Items**, click **User Interface**.
    d. Click **Menu**.
    e. Enter **dbfmVehicleMenu** in the **Name** field.
    f. Click **Add**.
2. Set the properties.
    a. Right-click **dbfmVehicleMenu > Properties.**
    b. Set the **Label** property to **Vehicle**.
    c. Enter **dbfmVehicleListPage** in the **Menu Item Name** field.
3. Extend the menu.
    a. In the **Application Explorer**, expand **Application Suite**.
    b. Expand **User Interface** and expand the **Forms** node.
    c. Right-click **Organization Administration** and select **Create extension**.
4. Set the properties.
    a. In the **Solution Explorer**, right-click **OrganizationAdministration.Extention** and select **Open.**

b. Expand the **Resources** menu.
c. In the **Solution Explorer**, drag and drop **dbfmVehicleMenu** to **WrkCtrTable**.
d. Right-click **dbVehicleMenu** and select **New > Menu Item Reference**.
e. Right-click **MenuItemReference1 > Properties.**
f. Enter **dbfmVehicleListPage** in the **Menu Item Name** field.
g. Enter **Vehicles** in the **Name** field.
h. Set the **Visible** property to **Yes**.

5. Build.
    a. In the **Solution Explorer** , right-click **DevelopmentBasicsFMSProject**.
    b. Click **Build.**
    c. Review the Output screen to review process.
6. Update forms with menus.
    a. In the **Solution Explorer**, open **dbfmVehicleListPage.**
    b. Expand the **Design** node.
    c. Right-click **FromActionPaneControl1** and select **New > Button Group**.
    d. Right-click **FromButtonGroupControl1** and select **Properties**.
    e. Enter **Vehicles** in the **Captions** field.
    f. Right-click **FromButtonGroupControl1** and select **New > Menu Item Button.**
    g. Right-click FromMenuFunctionButtonControl and select **Properties**.
    h. Enter **Vehicle** in the **Name** field.
    i. Enter **dbfmVehicleDetails** in the **Menu Item Name** field.
    j. Set the **Form View Option** property to **Details**.
    k. Set the **Big** property to **Yes**.
    l. Right-click **FromButtonGroupControl1** and select **New > Menu Item Button.**
    m. Right-click **FromMenuFunctionButtonControl** and select **Properties**.
    n. Enter **VehicleMaintenance** in the **Name** field.
    o. Enter **dbfmVehicleMaintenance** in the **Menu Item Name** field.
    p. Set the **Form View Option** property to **Details**.
    q. Set the **Big** property to **Yes**.
7. Build.
    a. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject**.
    b. Click **Build**.
    c. Review the Output screen to review process.

Module 12: X++ Overview

Lesson 1: Code Editor



The **X++** code editor is a text editor that supports color-coding, word completion, automatic indenting, scripting, zoom, multiline editing, and many other features that are available in **Visual Studio**.

To open the code editor, double-click a code element in the AOT. Alternatively, click **Edit** in the shortcut menu for an element, or create a new job from the **Jobs** node in the AOT.

The editor enables you to perform standard editing functions, such as copy, cut, and paste.

The editor has shortcut menus (activated by right-clicking) where the available commands depend on whether you have selected text in the editor.

The standard features that a developer expects from the code editor are supported. For example, **sections of codes** are **collapsible** using these plus and minuses here on the left-hand side of the code editor.
IntelliSense is also a feature that provides guidance as you write or modify code.
The X++ language is fully integrated into the Visual Studio environment
You can navigate to methods and classes in the code editor window using the navigational drop-down menus located at the top of the code editor window

BENEFITS OF X++ and .NET CIL

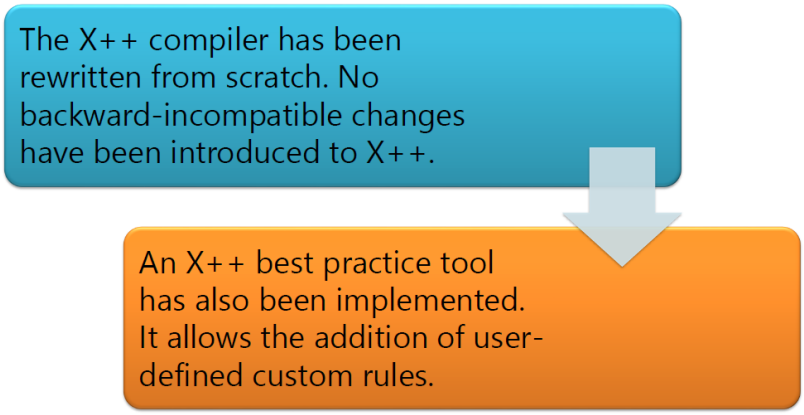
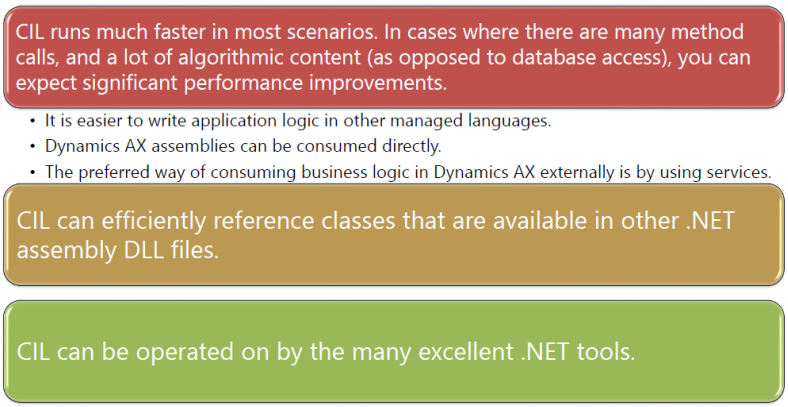
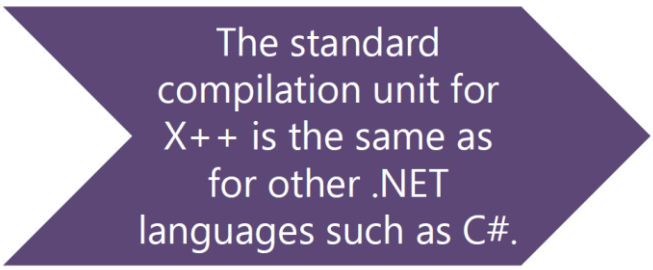


Figure 17 Benefits of X++ and .NET CIL

COMPILER





Lesson 2: Create runnable class

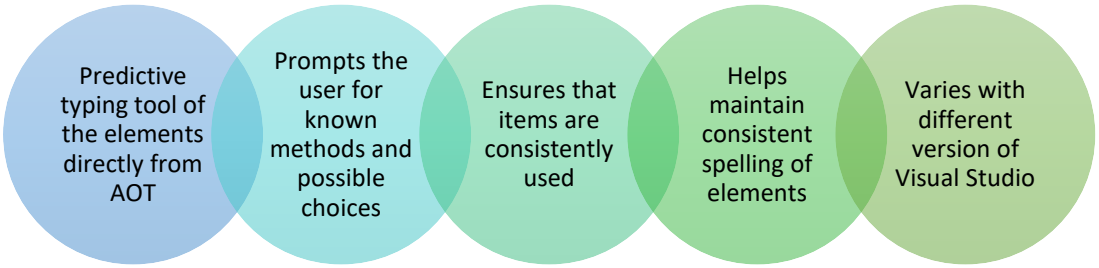
1. Create a class.
    a. In the **Solution Explorer**, right-click **DevelopmentBasicsFMSProject** and select **Add > New Item**.
    b. Under **Dynamics 365 Items**, select **Code**.
    c. Select **Runnable Class (job)**.
    d. Enter **dbfmFleetManagementWelcome** in the **Name** field.
    e. Click **Add**.
2. Input code.
    a. Add the 'info(…' line below:

```
class dbfmFleetManagementWelcome
{
public static void main(Args _args)
{
info("Welcome to the Fleet Management Module");
}
}
```

    b. Click **Save**.
3. Set as the startup object and start.
    a. In the **Solution Explorer**, right-click **dbfmFleetManagementWelcome** and select **Set as Startup Object.**
    b. Click **Start**.

Lesson 3: IntelliSense

IntelliSense is a code completion tool that is built into Microsoft Visual Studio. It is one of a few similar tools that allows intelligent code completion or text completion on different platforms. Using various algorithms, IntelliSense attempts to guess what the developer wants to type in order to complete a line of code. Using this tool may reduce typographical and syntactical errors.



Lesson 5: Data Types

PRIMITIVE DATA TYPES

Anytype	A placeholder for any data type.
Booleans	Can only contain the values false and true.
Dates	Contains day, month, and year.
Enums	An abbreviation for enumerated text—a set of literals.
GUIDs	A globally unique identifier.
Integers	A number without a decimal point. To declare an integer, use the keyword int.
Reals	Numbers with a decimal point; also called decimals.
Strings	A number of characters. To declare a string, use the keyword str.
TimeOfDay	Contains hours, minutes, and seconds. To declare a time, use the system type timeOfDay.
utcdatetime	Contains year, month, day, hour, minute and second.

Figure 18 - Primitive Data Types

COMPOSITE DATA TYPES

X++ composite data types	Description
Arrays	An array is a list of items with the same data type and the same name—only the index differs.
Containers	A container is a dynamic list of items containing primitive data types and/or some composite data types.
Classes as Data Types	A class is a type definition that describes both variables and methods for instances (objects) of the class.
Tables as Data Types	All tables defined in the database can be handled as class definitions.

Figure 19 - Composite Data Types

EXTENDED DATA TYPES

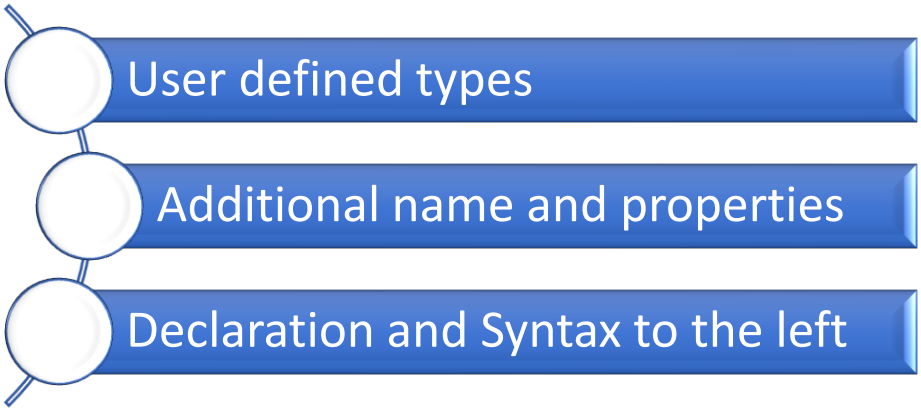
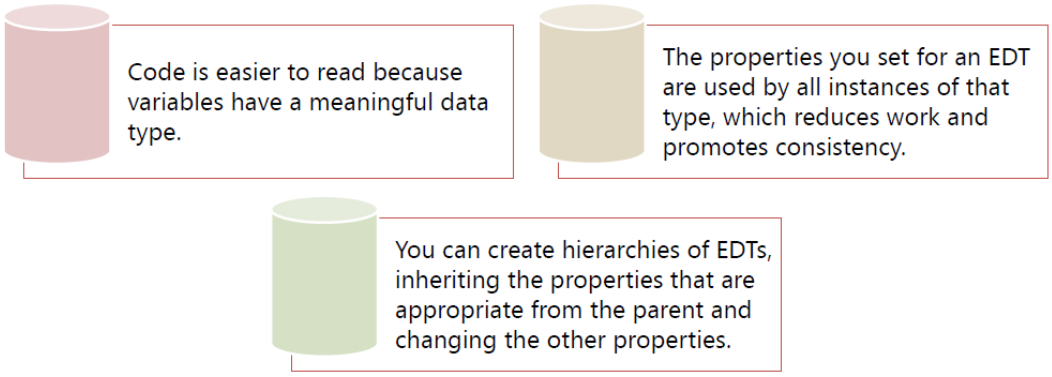


Figure 20 - Extended Data Types

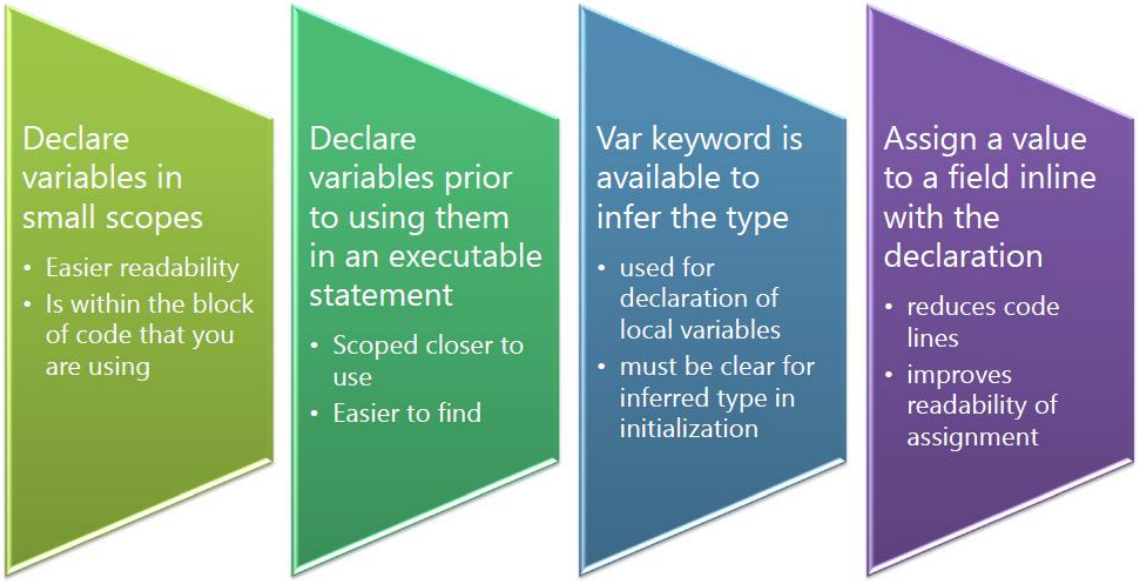
BENEFITS OF EDT



EDT CODE SAMPLE

```
class DevBasicsEDTSample
{
public static void main(Args _args)
    {
        DBVehicleType vehicleType;
        vehicleType = 'Truck';
        info(vehicleType);
    }

}
```

Lesson 4: Variable Declaration



Lesson 6: Key Operators

ASSIGNMENT OPERATORS

Operator	Description
=	Assigns the expression on the right of the equal sign to the variable on the left.
+=	Assigns the variable on the left the current variable value plus the expression on the right.
++	Increments the variable by one.
-=	Assigns the variable on the left the current variable value minus the expression on the right.
--	Decrements the variable by one.

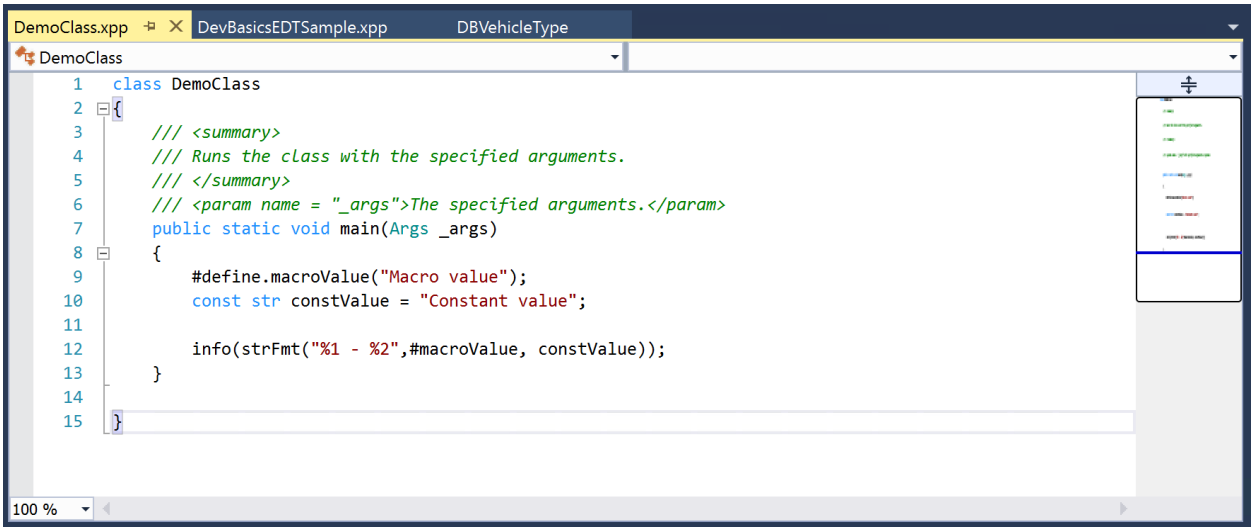
ARITHMETIC OPERATORS

Operator	Term	Description
<<	Left shift	Performs expression2 left shift (a multiplication by 2) on expression1.
>>	Right shift	Performs expression2 right shift (a division by 2) on expression1.
*	Multiply	Multiplies expression1 by expression2.
/	Divide	Divides expression1 by expression2.
DIV	Integer division	Performs an integer division of expression1 by expression2.
MOD	Integer remainder	Returns the remainder of an integer division of expression1 by expression2.
~	Not	Unary operator. Performs a binary not-operation.
&	Binary AND	Performs a binary and-operation on expression1 and expression2.
^	Binary XOR	Performs a binary XOR-operation on expression1 and expression2.
|	Binary OR	Performs a binary or-operation on expression1 and expression2.
+	Plus	Adds expression1 to expression2.
-	Minus	Subtracts expression2 from expression1.
?	Ternary operator	Takes three expressions: expression1 ? expression2 : expression3. If expression1 is true, expression2 is returned; otherwise, expression3 is returned.

RELATIONAL OPERATORS

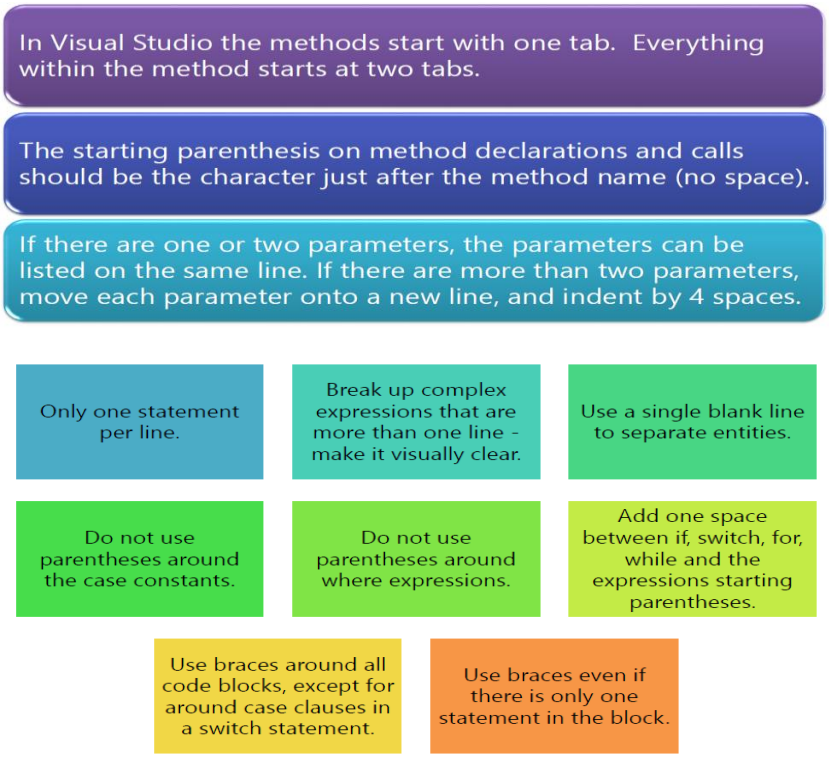
Operator	Description
like	Returns true if expression1 is like expression2.
!	Not.
!=	Inequality operator (not equal to).
#	Prefix on macro names.
&&	Logical AND.
||	Logical OR.
<	Less than.
==	Returns true if both expressions are equal.
>	Greater than.
>=	Greater than or equal.

MACROS AND CONST

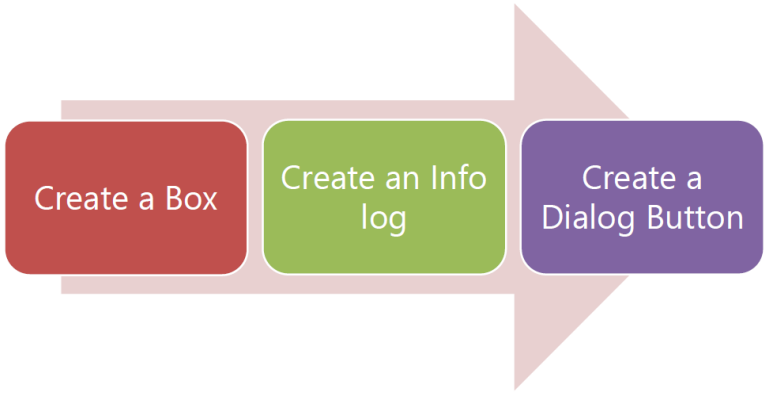


- Macros can still be created in Dynamics 365 for Finance and Operations to store a value that will not be modified. They are created in either a macro file or by using the #define. Keyword.
- However, there is a new modifier to create constant variables. By placing the keyword “const” before a variable it will create a value that cannot be modified.

Lesson 7: Basic Syntax



BASIC OUTPUT



OUTPUT SYNTAX

Box

```
Box::info("Body text", "title",
"Help text");

Box::warning("This posting needs
receiving first.", "warning title",
"Help text");
```

Info Log

```
info("Information notice to the
interface");

warning("Warning notice to the
interface");

error("Error notice to the
interface");
```

- Dialog

```
DialogButton   dialogButton;

dialogButton = Box::yesNoCancel ("Do you want to
continue?",DialogButton::Yes,"Inquire","Help text");
```

SUPPORTED STATEMENTS AND LOOPS

IF..ELSE AND SWITCH STATEMENTS

```
int oilchange = 3;

if(oilchange >=2)
{
   info("Repeat customer");
}

   else
   {
     info("New customer");
   }
```

```
int oilchange = 3;
str answer;

switch(oilchange)
  {
     case 1:
       answer = "First time customer";
       break;
     case 2:
       answer = "Repeat customer";
       break;
     default:
       answer = "Frequent customer";
       break;
   }
   info(strFmt(answer));
```

WHILE AND FOR

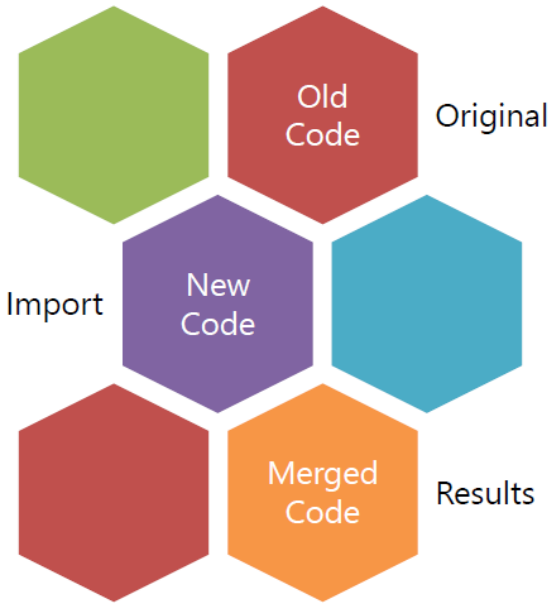
While Example

```
int counter1 = 0;

while (counter1 != 4)
{
   info(int2Str(counter1));
   counter1++;
}
```

For Example

```
for (int counter2 = 0; counter2
     !=4; counter2++)
{
    info(strFmt("counter2 is %1",
    counter2));
}
```

Lesson 8: Comparison Tool

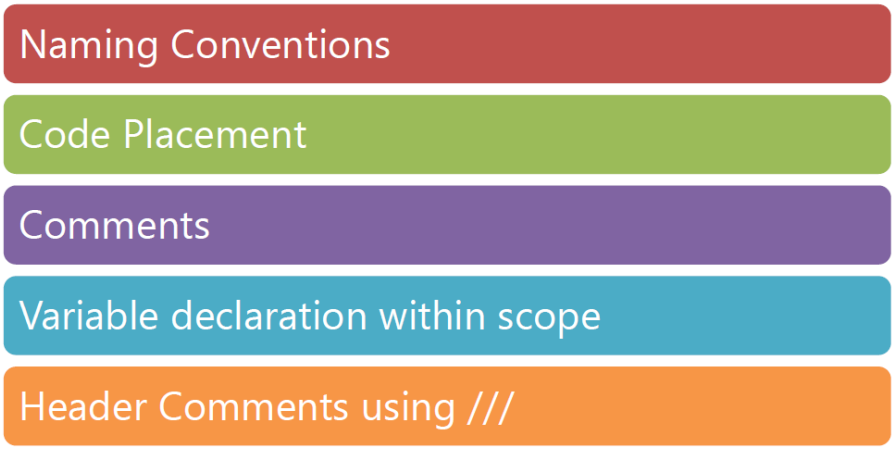


When using **comparison tools**, you could have the original code in the left pane, the new code in the center pane, and the merged code in the right pane and interact with all three. To install the comparison tools, it is required to have source control both configured and installed. In addition, there are visual studio extensions that can be installed for comparison. Refer to MSDN for the version of Visual Studio that you are operating for both the extensions and command line options to compare code versions.

Lesson 9: Debugger

- Tools like the Autos window in Visual Studio will show important information about the state of the application.
- Another tool specific to Dynamics 365 for Finance and Operations Preview is the Infolog. Frequently, info() statements are added to code to log status messages as the application is running. You can view these Infolog messages directly within Visual Studio. On the VIEW menu, choose Infolog.

Lesson 10: Best Practices



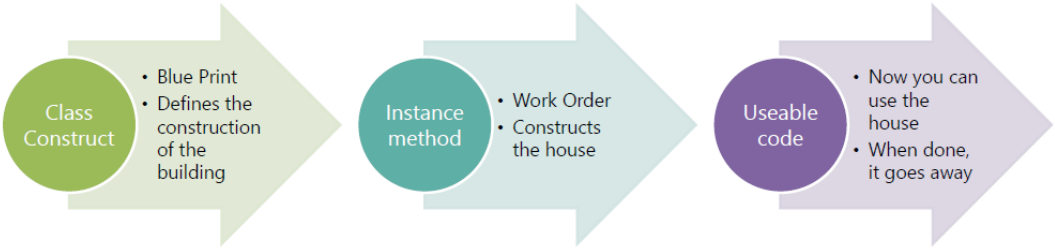
Module 13: Classes

A **class** is a software construct that defines the **data** and **methods** of the specific concrete objects that are subsequently constructed from that class. The data represents the state of the object. The methods represent the behavior of the object.

Variables are the data for the class. Variables in an X++ class are specific to objects that are constructed from that class. Every object constructed from the class declaration has its own copy of the variables. Such variables are known as instance variables. You cannot declare static variables in X++.

Methods are the sequences of statements that operate on the data. Methods are typically declared to operate on the instance variables of the class and are known as instance methods or object methods. In X++ you can also declare static methods.

Lesson 1: Class Structure



Lesson 2: Create a Base Class

1. Create a new class.
    a. In the Solution Explorer, right-click **DevelopmentBasicsFMSProject** and select **Add > New Item**.
    b. Under **Dynamics 365 Items**, select **Code**.
    c. Select **Class**.
    d. In the **Name** field, enter **dbFmVehicleEntry**.
    e. Save the changes.
2. Create methods for dialog creation.
    a. Add a 'setupDialog' method which enables users to enter Make, Model, and VIN information. A
    b. Add the class member variables used as part of the 'setupDialog' method:

```
Dialog        dialog;
DialogField   dialogFromMake;
DialogField   dialogFromModel;
DialogField   dialogFromVIN;

public void setupDialog()
{
    dialogFromMake = dialog.addField(extendedtypestr(dbMake),"Make");
    dialogFromModel = dialog.addField(extendedtypestr(dbModel),"Model");
    dialogFromVIN = dialog.addField(extendedtypestr(dbVIN),"Vehicle Identification Number");
}
```

    Add a 'getFromDialog' method which populates a new dbVehicleTable buffer with the information entered into the dialog by the user. Also add the class member variables used as part of the 'getFromDialog' method:

```
                                dbVehicleTable vehicleTable;

public void getFromDialog()
{
    str Make,Model,VIN;
    Make = dialogFromMake.Value();
    Model = dialogFromModel.Value();
    VIN = dialogFromVIN.value();
    vehicleTable.dbMake = Make;
    vehicleTable.dbModel = Model;
    vehicleTable.dbVIN = VIN;
}
```

    c. Validate the new methods against complete code at the end of this document.
    d. Save the changes.

3. Create methods for vehicle creation.
    a. Add a 'new' method which creates a dialog for vehicle detail entry:

```
public void new()
{
    dialog = new Dialog("Quick Vehicle Entry Form");
}
```

b. Add a 'run' method which sets up the dialog, runs the dialog, waits for user input, then inserts the new vehicle record if the user has provided input:

```
public void run()
{
    this.setupDialog();
    dialog.run();
    dialog.wait();
    if (dialog.closedOk())
    {
       this.getFromDialog();
                                    if (vehicleTable.validateWrite())
                                    {
                                            ttsbegin;
                                    vehicleTable.insert();
                                            ttscommit;
                                    }
    }
}
```

c. Add a 'main' method which instantiates a new dbFmVehicleEntry object and runs it:

```
public static void main(Args _args)
{
dbfmVehicleEntry vehicleEntry = new dbfmVehicleEntry();

                                    vehicleEntry.run();
                                    }
```

d. Validate the new methods against complete code at the end of this document.
e. Save the changes.

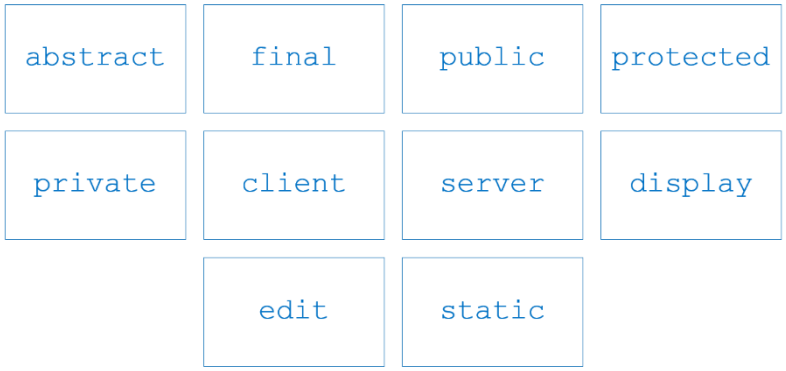
4. Build and run the changes.
   a. In the Solution Explorer, right-click the **dbFmVehicleEntry** class.
   b. Select **Set as Startup Object**.
   c. Click the **Start** button.

Complete Code

```
class dbfmVehicleEntry
{
   Dialog dialog;
   dbVehicleTable vehicleTable;
   DialogField dialogFromMake;
   DialogField dialogFromModel;
   DialogField dialogFromVIN;
   public static void main(Args _args)
   {
      dbfmVehicleEntry vehicleEntry = new dbfmVehicleEntry();
      vehicleEntry.run();
   }
   public void new()
   {
```

```
        dialog = new Dialog("Quick Vehicle Entry Form");
    }
    public void run()
    {
        this.setupDialog();
        dialog.run();
        dialog.wait();
        if (dialog.closedOk())
        {
            this.getFromDialog();
                                    if (vehicleTable.validateWrite())
                                    {
                                            ttsbegin;
                                    vehicleTable.insert();
                                            ttscommit;
                                    }
        }
    }
    public void setupDialog()
    {
        dialogFromMake = dialog.addField(extendedtypestr(dbMake),"Make");
        dialogFromModel = dialog.addField(extendedtypestr(dbModel),"Model");
        dialogFromVIN = dialog.addField(extendedtypestr(dbVIN),"Vehicle Identification Number");
    }
    public void getFromDialog()
    {
        str Make,Model,VIN;
        Make = dialogFromMake.Value();
        Model = dialogFromModel.Value();
        VIN = dialogFromVIN.value();
        vehicleTable.dbMake = Make;
        vehicleTable.dbModel = Model;
        vehicleTable.dbVIN = VIN;
    }
}
```

Lesson 3: Methods



There are several modifiers that can be applied to method declarations. Some of the modifiers can be combined (for example, final static). Some modifiers can only be used methods, while others can be used on classes or methods.

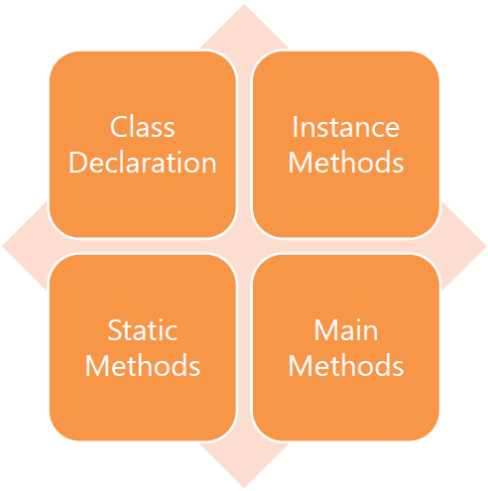
The following list describes the method modifier keywords of the X++ language.

- **abstract**: The method is declared but not implemented in a parent class. The method must be overridden in subclasses. If you try to create an object from a subclass where one or more of the abstract methods belonging to the parent class have not been overridden, you will get a compiler error.

Note: Classes can also be abstract. Sometimes a class represents an abstract concept, but it should not be instantiated: Only subclasses should be instantiated. Such base classes can be declared abstract. Consider the case where the programmer wants to model the concept of an account. Accounts are abstract—only derived classes (ledger accounts and so on) exist in the real world. This would be a clear case for declaring the Account class abstract.

- **final**: Indicates that the method cannot be overridden in any class that derives from its class.
- **public**: Methods that are declared as public are accessible anywhere the class is accessible and can be overridden by subclasses. Methods that have no access modifier are implicitly public.
- **protected**: Methods that are declared as protected can only be called from methods in the class and in subclasses that extend the class where the method is declared.
- **private**: Methods that are declared as private can be called only from methods in the class where the private method is declared.
- **client**: Establishes the location where the method is to be executed (on the client). Can only be used on static methods. If the method is not static, specify the location by using the class property RunOn.
- **server**: Establishes the location where the method is to be executed (on the server). Can only be used on static methods. If the method is not static, you need to specify the location using the class property RunOn.
- **display**: Indicates that the method's return value is to be displayed on a form or a report. The value cannot be altered in the form or report. The return value is typically a calculated value, for example, a sum.

- **edit**: Indicates that the method's return type is to be used to provide information for a field that is used in a form. The value in the field can be edited.
- **static**: Specifies that the method is a class method and does not operate on an object. Static methods cannot refer to instance variables and are invoked by using the class name rather than on an instance of the class (MyClass::aStaticProcedure()).



Methods can be viewed from the Application Object Tree (AOT) or within the code itself. Method declarations consist of a header and a body. The method header declares the name and return type (possibly void) of the method, the method modifiers, and parameters. The method body consists of variable declarations, method declarations, and statements.

- **Class Declaration** - The Class Declaration specifies the name of the class and necessary variables. It can also specify inheritance.

- **Instance Methods** - Instance methods, or object methods, are embedded in each object that is created from the class. They are called by using the following syntax:

objectHandleName.methodName();

You must instantiate the object before you can use the method.

- **Static Methods** - Static methods, or class methods, belong to a class and are created by using the keyword static. They are called by using the following syntax:

ClassName::methodName();

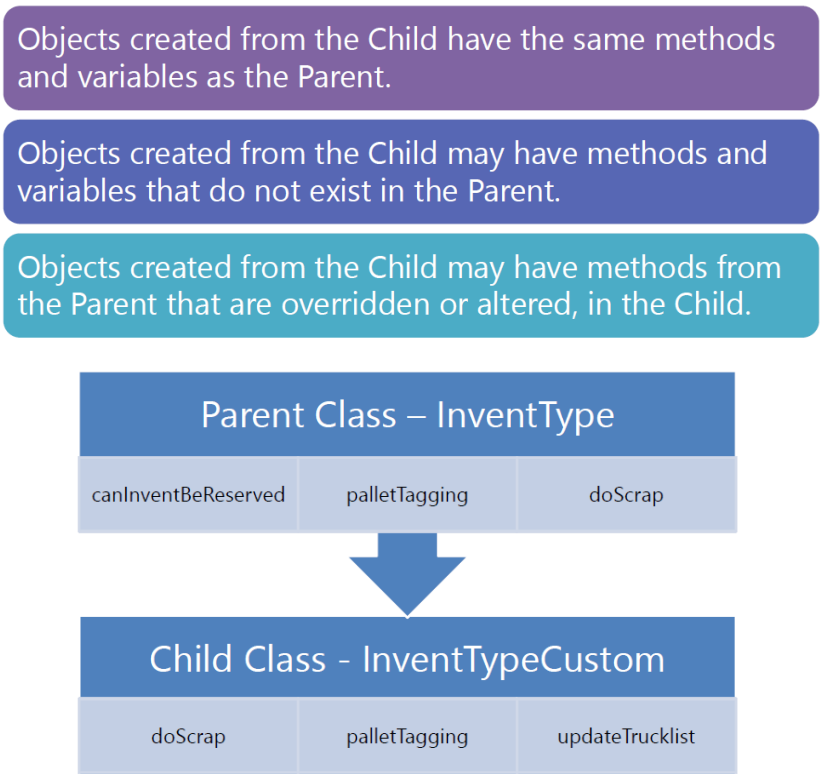
You do not need to instantiate an object before you use static methods. Static methods are widely used in Microsoft Dynamics 365 for Finance and Operations, Enterprise Edition to work with data that is stored in tables.

- **Main Methods** - A main method is a class method that is executed directly from a menu option.

```
static void main (Args _args)
{
    // Your X++ code here.
}
```

The method should only create an instance of the object and then call the necessary member methods. The _args parameter allows you to transfer data to the method.

Lesson 4: Class Inheritance

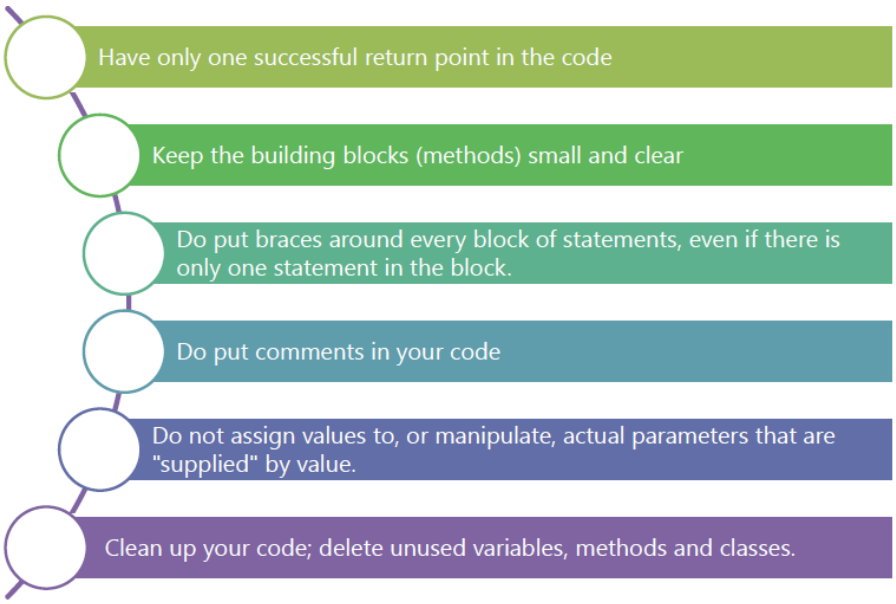


VARIABLES AND METHOD DECLARATION

Important best practice is to declare variables and methods as locally as possible.

- **Public**
  - Declare variables and methods for safe re-use by making them public. It is important to ensure that they are explicitly declared so that there is no ambiguity as to the intention of the programmer.
- **Private**
  - Declare variables and methods as private to keep the context of the variable local or within scope of only that method. This prevents accidental miss use or unintentional variable re-use.
- **Protected**
  - This is a blend of public and private. Protected keeps the variable public only for the subclasses that may extend or overlay yet restricts access to outside object calls.

Lesson 5: Best Practice



Module 14: Database Manipulation

Lesson 1: Data Retrieval

- Returning data for view is a common feature
- You will need to read records from the database
- Single calls can be possible for uniquely index tables
- More effective searches iterate over the whole table to ensure no item is missed

Lesson 2: Reading Records

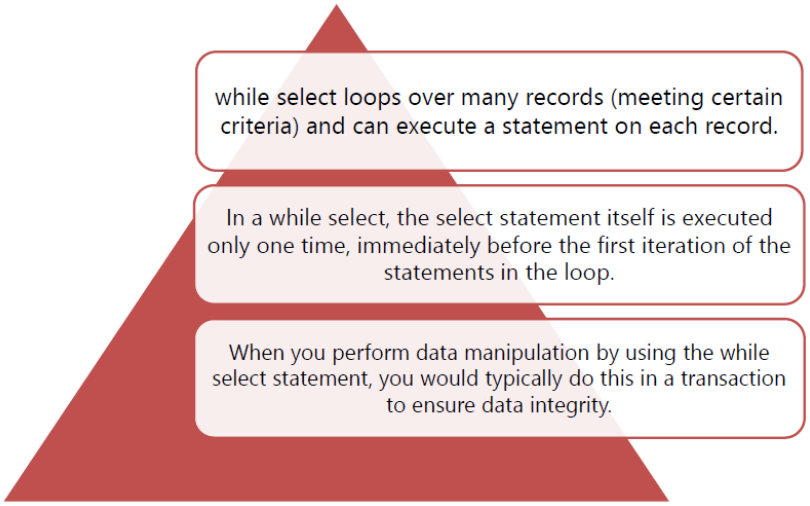
SELECT STATEMENT

- Using the "select" statement
- Using a table variable with a select statement
- Fetching additional records with the "next" statement and "while select" statements
- Results of a select statement are returned in a table buffer variable.

SELECT STATEMENT SYNTAX

SelectStatement	=	select Parameters
Parameters		[[FindOptions] [FieldList from]] TableBufferVariable [IndexClause] [Options] [WhereClause] [JoinClause]
FindOptions	=	crossCompany | reverse | firstFast | [firstOnly | firstOnly10 | firstOnly100 | firstOnly1000] | forUpdate | noFetch | [forcePlaceholders | forceLiterals] | forceselectorder | forceNestedLoop | repeatableRead | validTimeState
FieldList	=	Field { , Field } | *
Field	=	Aggregate (FieldIdentifier) | FieldIdentifier
Aggregate	=	sum | avg | minof | maxof | count
Options	=	[order by , group by , FieldIdentifier [asc | desc] { , FieldIdentifier [asc | desc] }] | [IndexClause]
WhereClause	=	where Expression
JoinClause	=	[exists | notexists | outer] join Parameters

WHILE SELECT STATEMENT

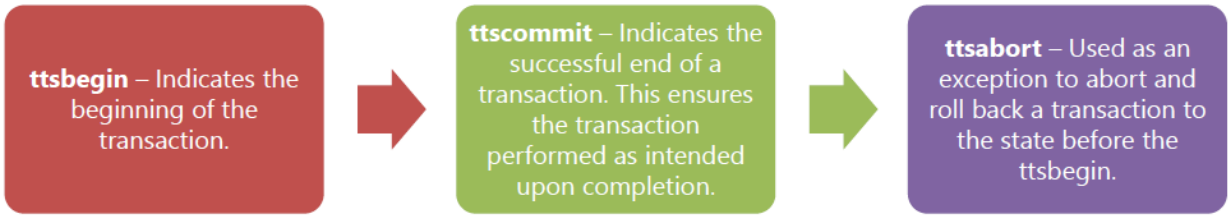


SAMPLE WHILE SELECT STATEMENT

```
CustTable       custTable;
DirPartyTable   dirPartyTable;

while select dirPartyTable order by Name
    join custTable
        where custTable.Party == dirPartyTable.RecId
{

    info(dirPartyTable.Name+', '+custTable.AccountNum);
}
```

Lesson 3: Transaction Integrity Checking

The following keywords help in integrity checking:

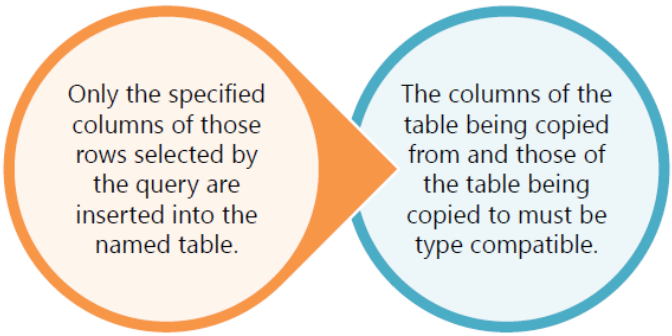


This check ensures that no record can be updated or deleted except from within the same transaction scope as it was selected for update. Integrity is ensured by using the following statements:

- **ttsBegin**: marks the beginning of a transaction. This ensures data integrity and guarantees that all updates performed until the transaction ends (by ttsCommit or ttsAbort) are consistent (all or none).
- **ttsCommit**: marks the successful end of a transaction. This end and commits a transaction. MorphX guarantees that a committed transaction will be performed according to intentions.
- **ttsAbort**: allows you to explicitly discard all changes in the current transaction. As a result, the database is rolled back to the initial state—nothing will have been changed. Typically, you will use this if you have detected that the user wants to break the current job. Using ttsAbort ensures that the database is consistent.

Nested **ttsbegins** and **ttscommits** are ignored in that a lock is held until the last **ttscommit** is reached. However, the system does keep track of the ttslevel. Each time a **ttsbegin** is called, the ttslevel increases by one. Every **ttscommit** decreases the level by one. If possible, you will want to make the tts blocks as small as required. When a record is selected for update, it will lock the records or table to prevent another process from also modifying them. The shorter the transaction block the less possibility for potential deadlocks.

Lesson 4: Data Insert



There are three methods for inserting records.

- **Insert** is the most common method and inserts a single record at a time.
- **InsertRecordSet** is used for bulk inserts and often used when inserting into a table from another.
- **InsertDatabase** is used for when performance is needed and the insert method may perform slowly. When using this method, the records will be inserted as other logic continues to process. When a record is inserted, logic will also be called from the insert method on the table. This can be bypassed by setting the table variable property of SkipDataMethods.

SAMPLE INSERT STATEMENT

```
InventItemPriceToleranceGroup   inventItemPriceToleranceGroup;

ttsBegin;
inventItemPriceToleranceGroup.itempricetolerancegroupid = '4%';
inventItemPriceToleranceGroup.name = 'TelevisionSeriesOne';
inventItemPriceToleranceGroup.insert();

inventItemPriceToleranceGroup.itempricetolerancegroupid = '6%';
inventItemPriceToleranceGroup.name = 'TelevisionSeriesTwo';
inventItemPriceToleranceGroup.insert();
ttsCommit;

info("Yahoo!");
```

At the very beginning of this code we are initializing a variable table buffer which is the **InventItemPriceToleranceGroup** table, and then here we're calling out how we're going to use it in our code. Now, below this is where we begin inserting our records, and we can note that by our **ttsbegin** keyword for data integrity. So here we can see that we are inserting these records for these specific fields. So, we have the **ItemPriceToleranceGroupID**, we're explicitly setting that to four percent. We can then see the name field. And then we are going to be setting that as **TelevisionSeriesOne**, and then we also have our insert method where we can see we are going to be inserting these records. And then we're also inserting another block of data. So, we

have the GroupID at six percent. TelevisionSeriesTwo, and then we're inserting that data. And then at the very bottom, if everything has run smoothly and everything is being inserted, we have our **ttscommit**. And then once all of our records have been inserted, we're displaying an **infolog** that says the records have been inserted.

Lesson 5: Data Update

Update command modifies existing data in a table with the contents of a table buffer. The record is first retrieved from the database using a select statement. The data is then modified by assigning the new values to the fields in the table buffer. The new values are then committed to the database using the update method. Before records can be updated, you'll use select **ForUpdate** to exclusively set a record for update. This tells SQL that a record is to be updated and allows the database to lock that record so that another user cannot modify it at the same time. Update statements must always be placed within a transaction block. Also, a bulk update can be performed by utilizing **Update_RecordSet**.

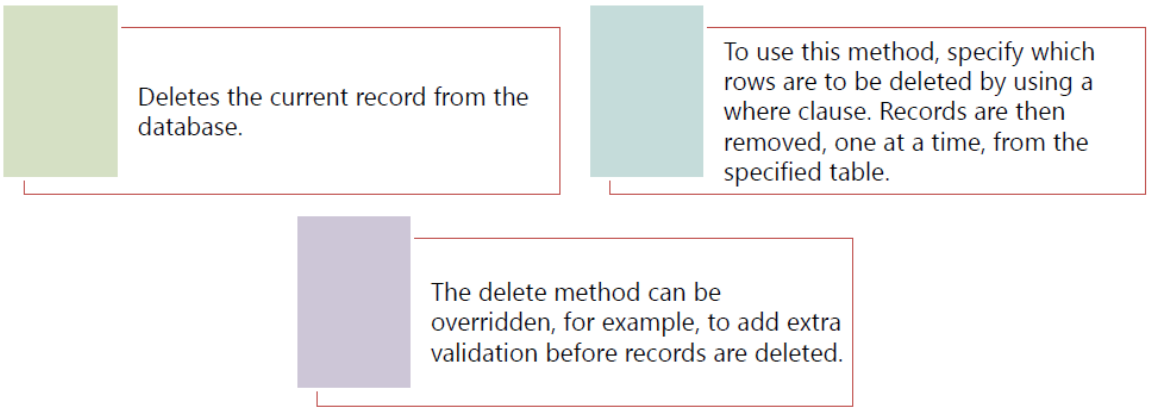
SAMPLE UPDATE STATEMENT

```
InventTable          inventTable;
InventItemGroupItem  inventItemGroupItem;

    ttsBegin;
    while select forupdate inventTable
    exists join inventItemGroupItem
        where InventItemGroupItem.ItemId == inventTable.ItemId
        && InventItemGroupItem.ItemGroupID == 'TV&Video'
{
    inventTable.ItemPriceToleranceGroupId ="7%";
    inventTable.update();
    info("this works!");

}
     ttsCommit;
```

We have here is a while select **ForUpdate** statement here on line 11. And so, what we want to update is the **InventTable**, as you can see here. And we're going to say we have an exists join for the **InventItemGroupItem** table, and we're basically saying here where the ItemID in the InventItemGroup item table is equal to the ItemID in the InventTable, and the ItemGroupID equals TV and video, we want to go ahead and update that record to be seven percent. And then once that is done, we want an infolog to display that the record has updated. And then of course we have our ttscommit. Here we have an **UpdateRecordSet** example. So here we're using our InventTable and InventItemGroupItem table. We have our ttsbegin, and then we have our UpdateRecordSet. So, we're using the InventTable here, and this is what we are setting here. So, the

ItemPriceToleranceGroupID, we're setting that explicitly to seven percent, and the ItemBuyerGroupID, setting that to demo. And then we have an exists join for the InventItemGroupItem table where the ItemID in the InventItemGroupItem table is equal to the ItemID in InventTable, and the ItemGroupID equals TV and video. And then we perform our ttscommit.

Lesson 6: Data Deletion



To use delete method, you'll specify which rows are to be deleted by using a where clause. Records are then removed, one at a time, from the specified table. The delete method can be overridden. So, for example, to add extra validation before records are deleted. For performance, book deletes can be performed utilizing the delete from.

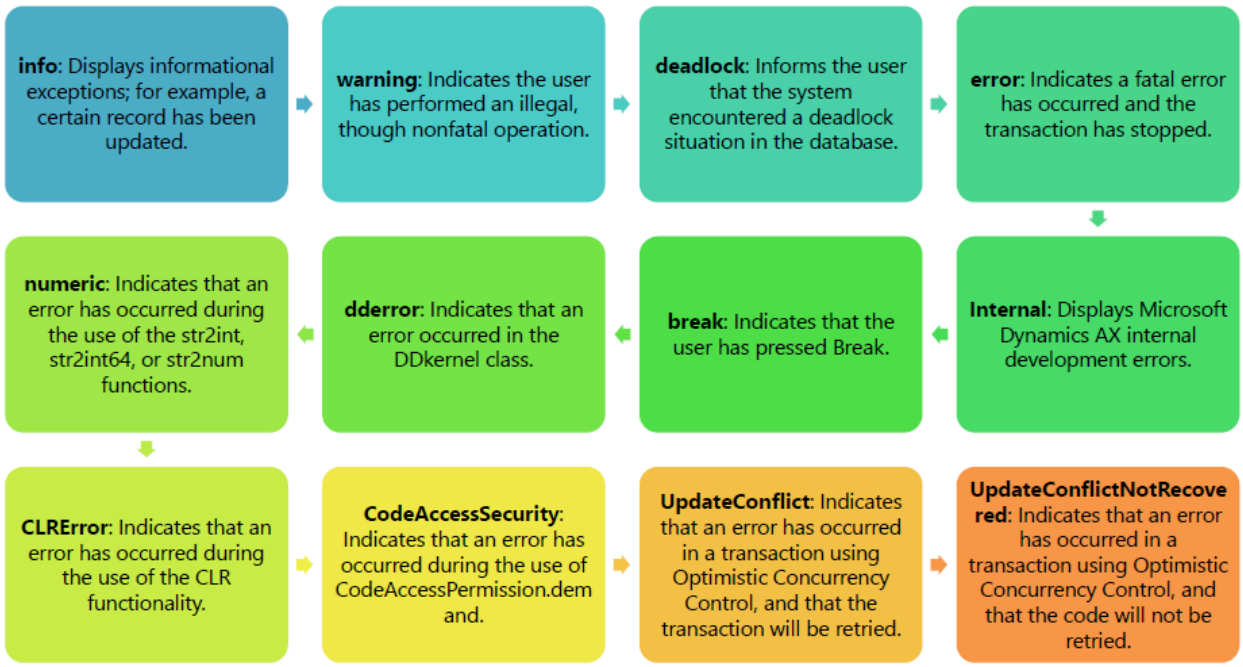
SAMPLE DELETE STATEMENT

```
InventTable          inventTable;
InventItemGroupItem  inventItemGroupItem;

ttsBegin;
while select forupdate inventTable
exists join inventItemGroupItem
    where InventItemGroupItem.ItemId == inventTable.ItemId
    && InventItemGroupItem.ItemGroupID == 'TV&Video'
{
          inventTable.delete();

          info("this works!");
}
 ttsCommit;
```

We have our **ttsbegin** and we're using a while select **ForUpdate** statement here. So, we are using the while select ForUpdate on the InventTable. And then we have our exists join on the InventItemGroupItem table. And we're basically saying where the ItemID in the InventItemGroup item table is equal to the ItemID in the InventTable, and where the InventItemGroupItem ItemGroupID is equal to TV and video, we want to perform a deletion. And here we can see that delete method. And then once that is done, we're going to display an infolog that says record deleted, and then we finish off our block with the ttscommit. Lastly, we have the SampleDelete_From. So again, we're initializing our InventTable and InventItemGroup item table. We are deleting from. So basically, we're using this **delete from**, and we're saying we're deleting from the **InventTable**. And then we're saying we have our exists join. So, we're using the InventItemGroupItem where the InventItemGroup ItemID is equal to the ItemID in the InventTable. And the ItemGroup ID is equal to TV and video. That is where we're going to perform that deletion. And then we have our **ttscommit** to make sure everything processes smoothly.

Module 15: Exception Handling

Lesson 1: Exception Types

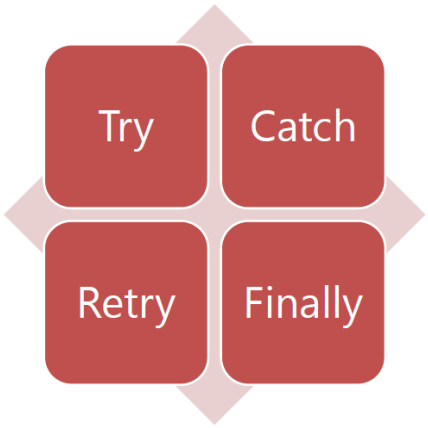


There are many different types of exceptions and the types differ depending on what caused the error. A majority of exception types are determined by the kernel and not normally thrown by the application code. All exception types, however, can be caught and it is the developer's responsibility to determine which exceptions need to be handled. The exception type is identified using the system enum exception. Because it is a system enum it cannot be modified so users cannot add new exception types.

- **Info** displays informational exceptions. So, for example, a certain record has been updated. We also have
- **warning**. This indicates the user has performed an illegal though a nonfatal operation.

- **Deadlock**. This informs the user that the system encountered a deadlock situation in the database, where two processes are trying to access the same record.
- **Error** indicates a fatal error has occurred and the transaction has stopped. Internal displays Microsoft Dynamics 365 for Finance and Operations internal development errors. A break indicates that the user has pressed break.
- **dderror** indicates that an error occurred in the ddkernel class.
- **Numeric** indicates that an error has occurred during the use of the string to integer, string to integer 64, or string to num functions.
- **CLRError** indicates that an error has occurred during the use of the CLR function. This is an error that you must often check for with integrations.
- **CodeAccessSecurity** indicates that an error has occurred during the use of the codeaccesspermission.demand.
- **UpdateConflict** indicates that the error has occurred in a transaction using optimistic concurrency control and that the transaction will be retried.
- **UpdateConflictNotRecovered** indicates that an error has occurred in the transaction using optimistic concurrency control and that the code will not be retried.

Lesson 2: Key Commands



The **try** command signifies the start of a block of code that you want to control with the X++ exception handling system. Any exceptions that are thrown in the block of code can be caught and handled accordingly. The block of code inside the try statement must be contained between curly braces. Throw statements can be used to trigger a catch statement. Errors are thrown so they can be picked up by error handling or just stop code execution.

Catch statements come after the block of try code and define what code is executed when each exception is thrown. You do not have to define a catch statement for every possible exception. However, each try statement must have at least one catch statement. A generic catch statement can be used to catch all thrown errors.

A **retry** command tells the system to go back and try the statement and attempt to execute the code again. Any data that was loaded before the try command will remain as it was. But any data retrieved or modified after the try statement will be refreshed. When a deadlock exception is thrown, all locks on tables that this process holds are released, which may allow for other process or processes that are deadlocked to continue. By calling a retry, the process can attempt to update the record again and may now be available to complete. It is a best practice to retry -- that a retry uses a counter so that the number of retries can be controlled and a user does not become stuck in a loop.

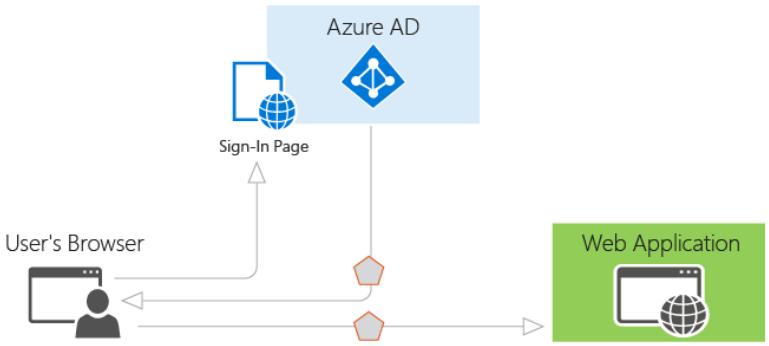
And lastly, we have the **finally** keyword. It is now available to follow and try and catch keywords. The semantics are identical to the semantics in C#. The statements provided in the finally clause are executed irrespective of whether the try block threw any exceptions.

Module 16: Security Basics

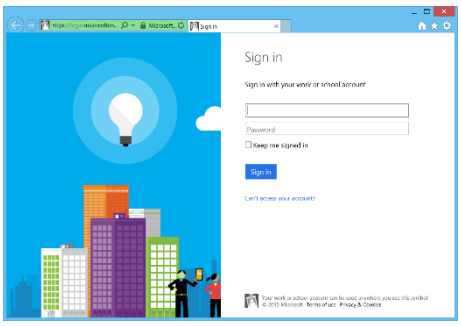
Lesson 1: Security Architecture

AUTHENTICATION FOR USER INTERFACE

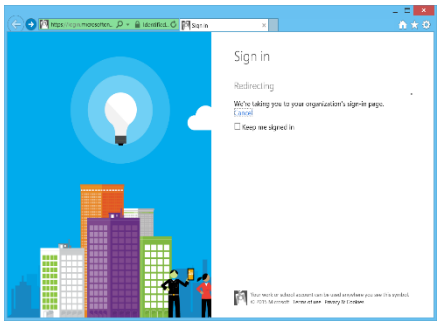
Ss



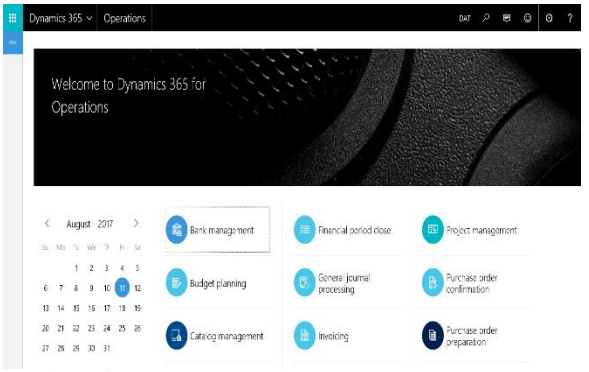
1. Sign-in via AAD (can be multi-factor AuthN)



2. AAD redirects to 'Dynamics 365 for Finance and Operations'



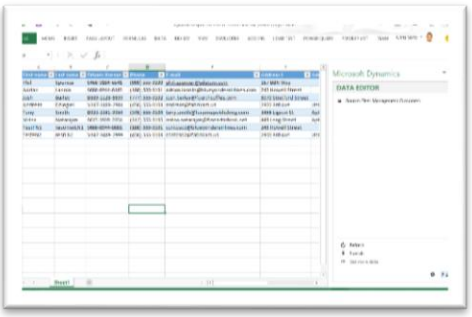
3. 'Dynamics 365 for Finance and Operations' start page displayed



AUTHENTICATION FOR SERVICES

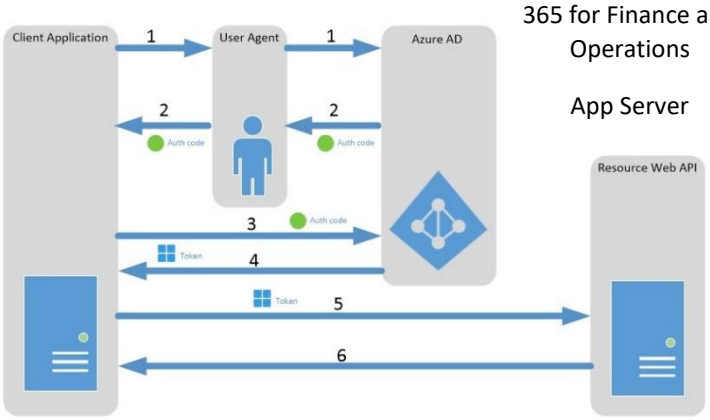
Power BI **Apps granted delegated access (but don't get to see credentials)**

Apps that consume Dynamics 365 for Finance and Operations web services



Microsoft Dynamics 365 for Finance and Operations

App Server



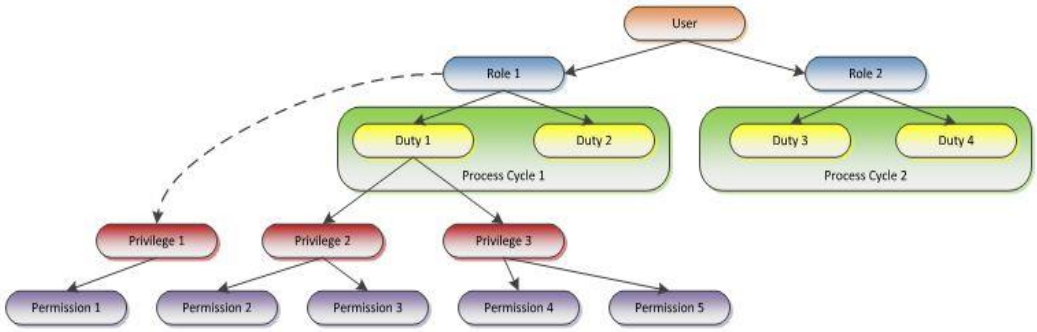
OData service, JSON custom service and REST metadata service are supported through the standard Oauth 2.0 authentication.

Certain applications can consume Microsoft Dynamics 365 for Finance and Operations Web services.

So, for example, we have Power BI, which is consumed from the OData service.

We also have Microsoft Azure Active Directory, or AAD. This is the native client application. So, this flow uses a user name and password for authentication and authorization. Then we also have the Web application which is the confidential client. A confidential client is an application that can keep a client password confidential to the world. The authorization server sign this client password -- assigns this client password to the client application. This will be supported post RTW.

SECURITY STRUCTURE IN AX



The **security** model is hierarchical and each element in the hierarchy represents a different level of detail.

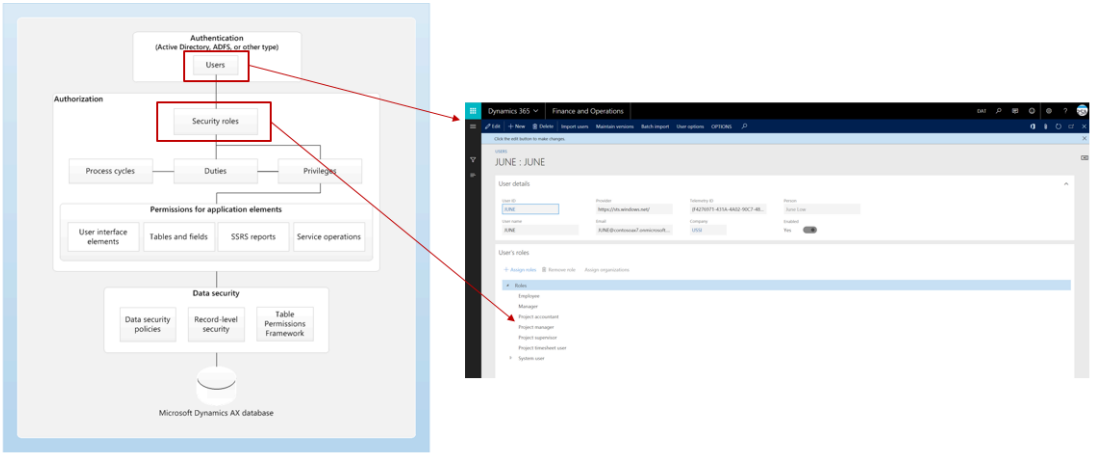
- **Permissions** represents access to individual securable objects such as menu items and tables.
- **Privileges** are composed of permissions and represent access to tasks such as canceling payments and processing deposits.
- **Duties** are composed of privileges and represent parts of the business process

Lesson 2: Roles Based Security

SECURITY AUTHORIZATION ACCESS CONTROL

Security Roles

All users must be assigned to at least one security role in order to have access to D365. The security roles that are assigned to a user determine the duties that the user can perform and the parts of the user interface that the user can view. Administrators can apply data security policies to limit the data for the access of user roles.



For example, a user in a role may have access to data only from a single organization. The administrator can also specify the level of access that the users in a role have to current, past, and future records and also users in a role

can be assigned privileges that allow them to view records for all periods, but that allow them to modify records only for the current period.

By managing access through security roles, administrators save time because they do not have to manage access separately for each users. Security roles are defined one time for all organizations. In addition, users can be automatically assigned to roles based on business data.

For example the administrator can set up a rule that associates a Human resources position with a security role. Any time that users are assigned to that position, those users are automatically added to the appropriate security roles. Users can also be automatically added to or removed from roles based on the Active Directory groups that they belong to.

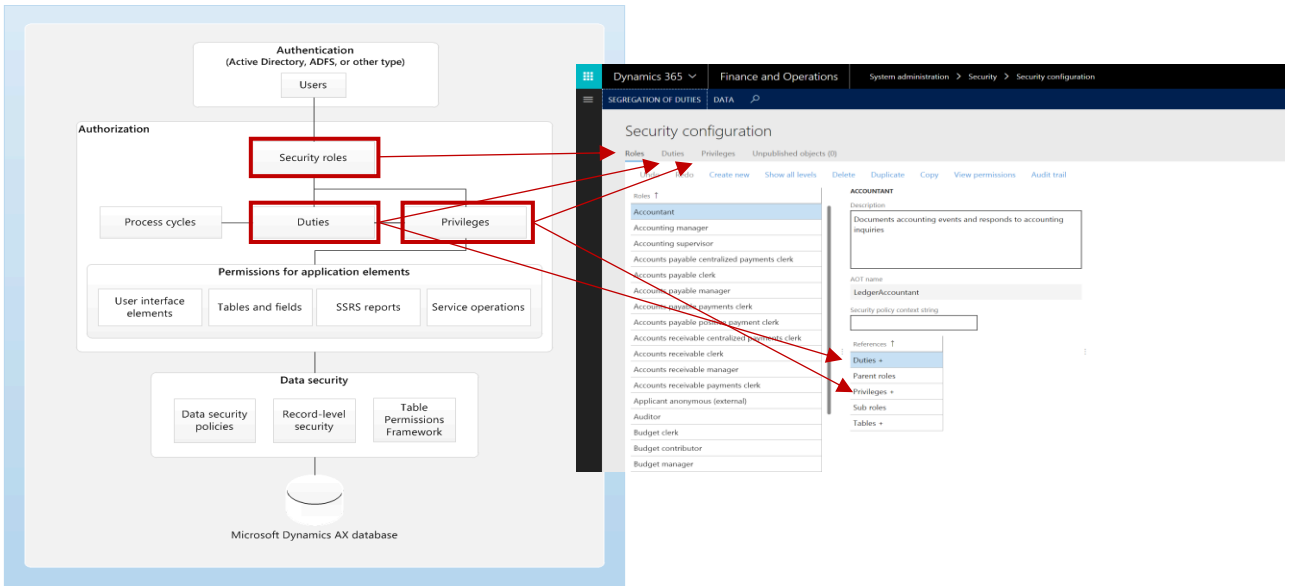
Security roles can be organized into a hierarchy. A role hierarchy enables roles to be defined as combinations of other roles.

For example the sales manager role can be defined as a combination of the manager role and the salesperson role. In the security model of D364, duties and privileges are used to grant access to the program also it can be assigned to maintain revenue policies and review sales orders duties.

By default, sample security roles are provided. All functionality in Microsoft Dynamics 365 for Finance and Operations is associated with at least one of the sample security roles. The administrator can assign users to the sample security roles, modify the sample security roles to fit the needs of the business, or create new security roles. By default, the sample roles are not arranged in a hierarchy.

Process Cycles

A business process is a coordinated set of activities in which one or more participants consume, produce, and use economic resources to achieve organizational goals.
To help the administrator locate the duties that must be assigned to roles, duties are organized by the business processes that they are part of. In the context of the security model, business processes are referred to as process cycles.



For example, in the accounting process cycle, you may find the Maintain ledgers and Maintain bank transactions duties. Process cycles are used for organization only. The process cycles themselves cannot be assigned to roles.

Duties

Duties correspond to parts of a business process. The administrator assigns duties to security roles. A duty can be assigned to more than one role. In the security model for Microsoft Dynamics 365 for Finance and Operations, duties contain privileges.

For example the Maintain bank transactions duty contains the Generate deposit slips and Cancel payments privileges. Although both duties and privileges can be assigned to security roles, it is recommended that you use duties to grant access to Microsoft Dynamics 365 for Finance and Operations.

You can assign related duties to separate roles. These duties are said to be segregated. By segregating duties, you can better comply with regulatory requirements, such as those from Sarbanes-Oxley (SOX), International Financial Reporting Standards (IFRS), and the United States Food and Drug Administration (FDA). In addition, segregation of duties helps reduce the risk of fraud, and helps you detect errors or irregularities. Default duties can provide administrator to modify the privileges that are associated with a duty, or create new duties.

Privileges

In the security model for Microsoft Dynamics 365 for Finance and Operations, a privilege specifies the level of access that is required to perform a job, solve a problem, or complete an assignment. Privileges can be assigned directly to roles. However, for easier maintenance, we recommend that you assign only duties to roles. A privilege contains permissions to individual application objects, such as user interface elements and tables.

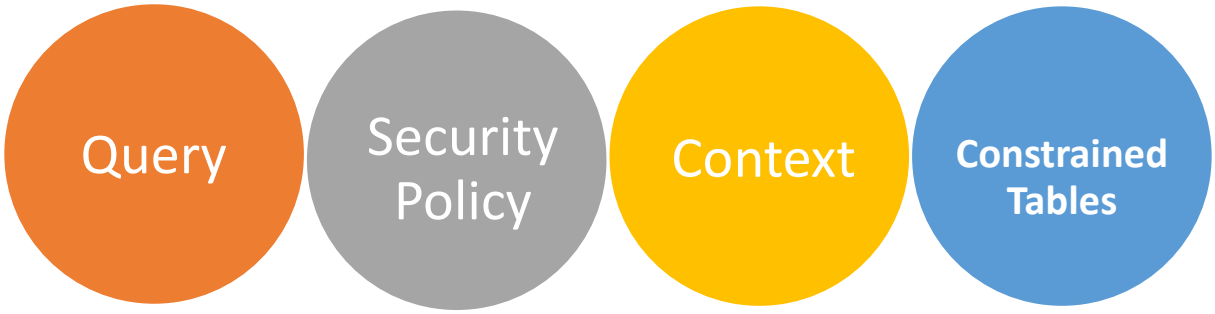
For example the Cancel payments privilege contains permissions to the menu items, fields, and tables that are required to cancel payments.

By default, privileges are provided for all features in Microsoft Dynamics 365 for Finance and Operations. The administrator can modify the permissions that are associated with a privilege, or create new privileges.

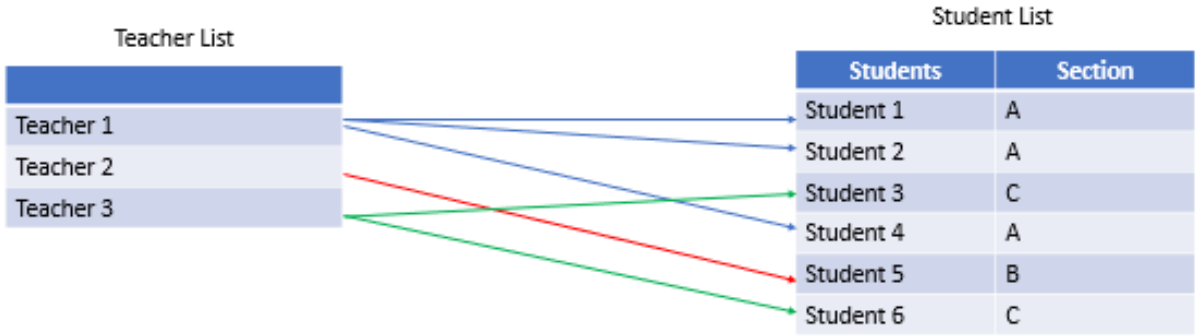
Permissions

In the security model for Microsoft Dynamics 365 for Finance and Operations, a permission grants access to logical units of data and functionality, such as tables, fields, forms, and server side methods. Only developers can create or modify permissions. The screen shot on top shows the Security configuration form where system administrators can create and edit roles and view the duties, privileges, and so on that are related.

Lesson 3: Extensible Data Security



XDS - Stands for **Extensible Data Security**. Enables developers and System administrator to secure data in shared tables such that user have access only the part of the table that is allowed by the enforced policy.

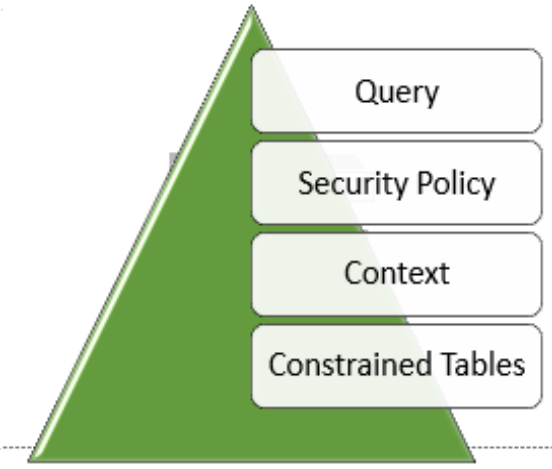


Assumption

- Teacher lists are the roles
- Student list is a table containing student records
- The objective of XDS is to create a policy that Teacher 1 can only see students from Section A, Teacher 2 can only see students from Section B, Teacher 3 can only see student records from Section C.

ELEMENTS used in XDS

Query defined the filters or limits that should be placed on the primary tables or tables



Security Policy linked to the query and primary table that you want to restrict data

Context rule which could base on a string for a role or group of roles.

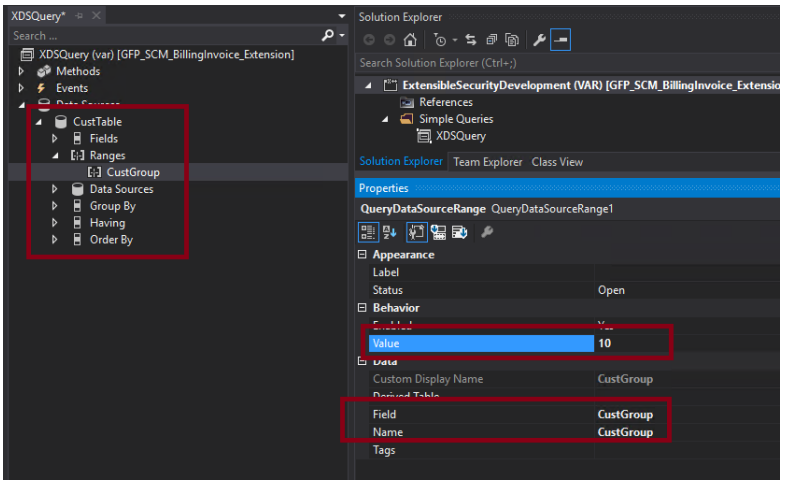
Constrained tables related tables that store data from the primary table that you're attempting to restrict a subset of data for

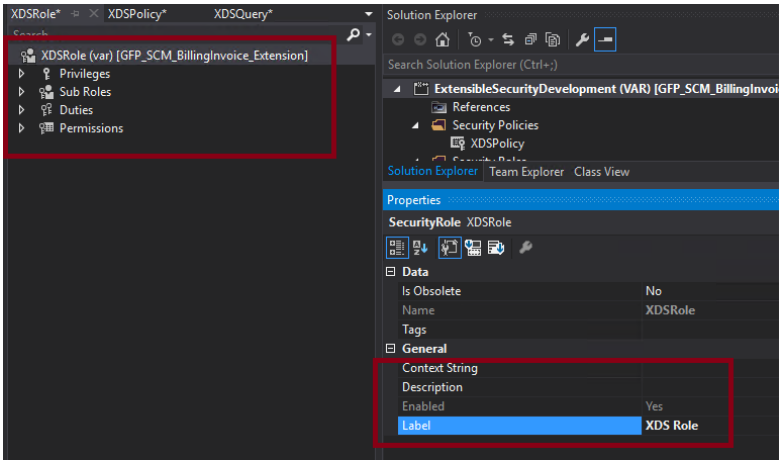
HOW TO CREATE XDS POLICY

1. Create a query

    a. Table to be used: CustTable
    b. Query Property
        i. Dynamics Fields: Yes

Range: CustGroup = '10

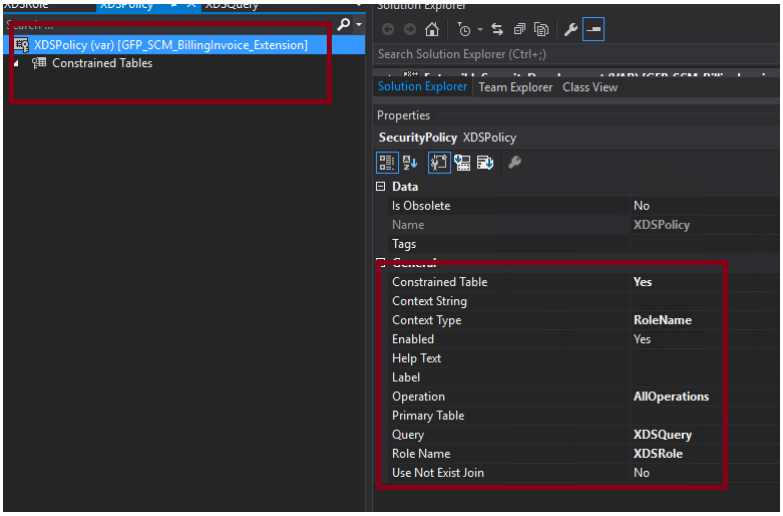




2. Create Security Role

Note: You may create new role or use existing roles.

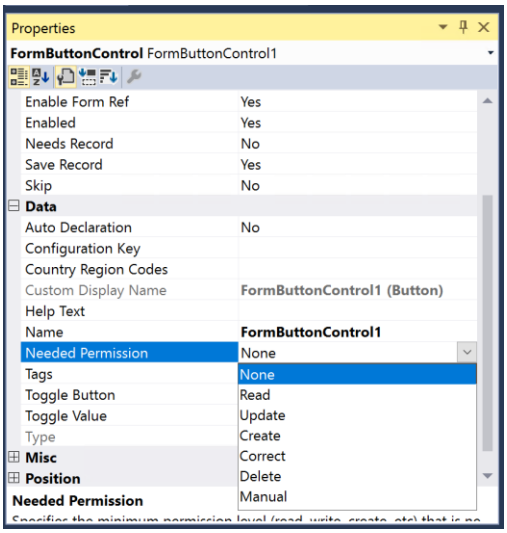
3. Create Security Policy
    a. Constrained Table = Yes
    b. Context Type = RoleName
    c. Operation = AllOperation
    d. Query = XDSQuery
    e. Role Name = XDSRole

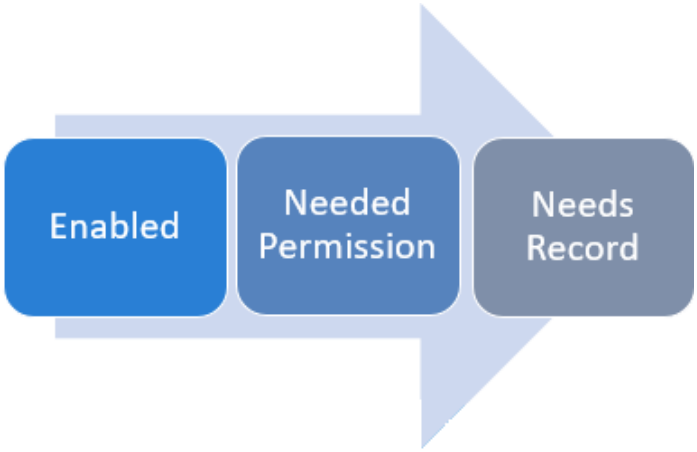


4. Attach role to user

Lesson 4: Security Properties on Key Elements

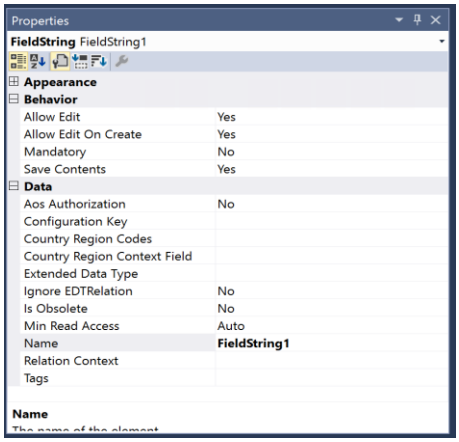
BUTTON PROPERTIES

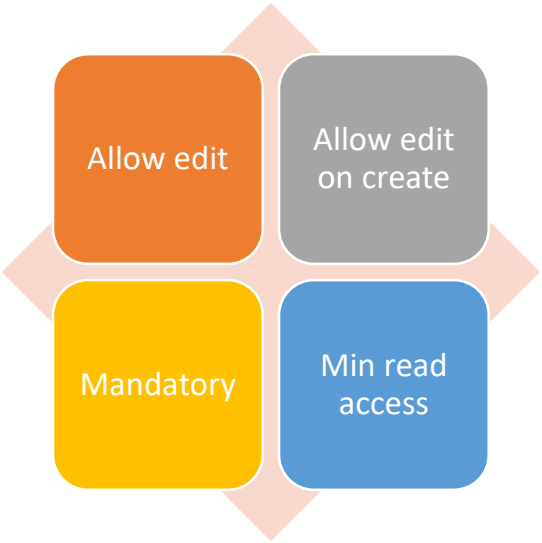




Needed permission allows for different levels of control and access.
Starting at the lowest level of only being able to view/read the control all the way up to the highest level of allowing for deletion.

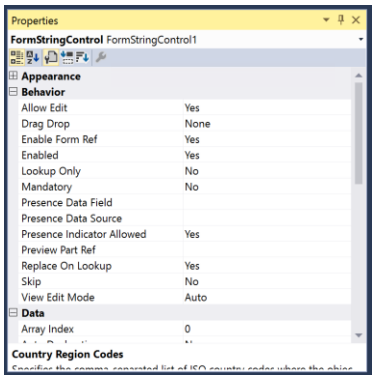
TABLE FIELDS

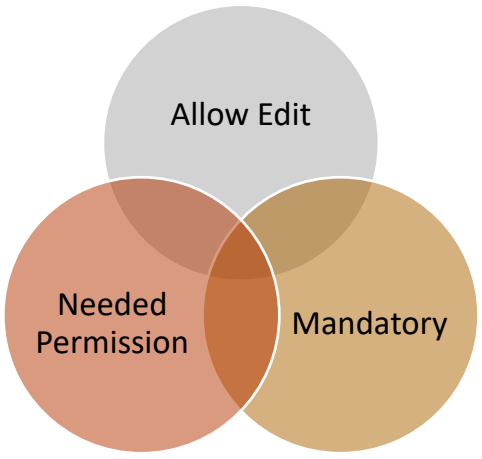




Allow edit defines whether you can change the value of the field.
Allow edit on create allows you to only edit the field until the record is saved. Once saved the field value is locked down.
Mandatory is used to require the field to be entered. The record cannot be saved until the field has a value. For a string this means the value cannot be blank, and for a number this means that the value cannot be 0.
Min read access is used to determine whether the field can be auto authorized for access.

FORM FIELDS





- **Allow Edit** - Specify whether you can modify the data in the control.

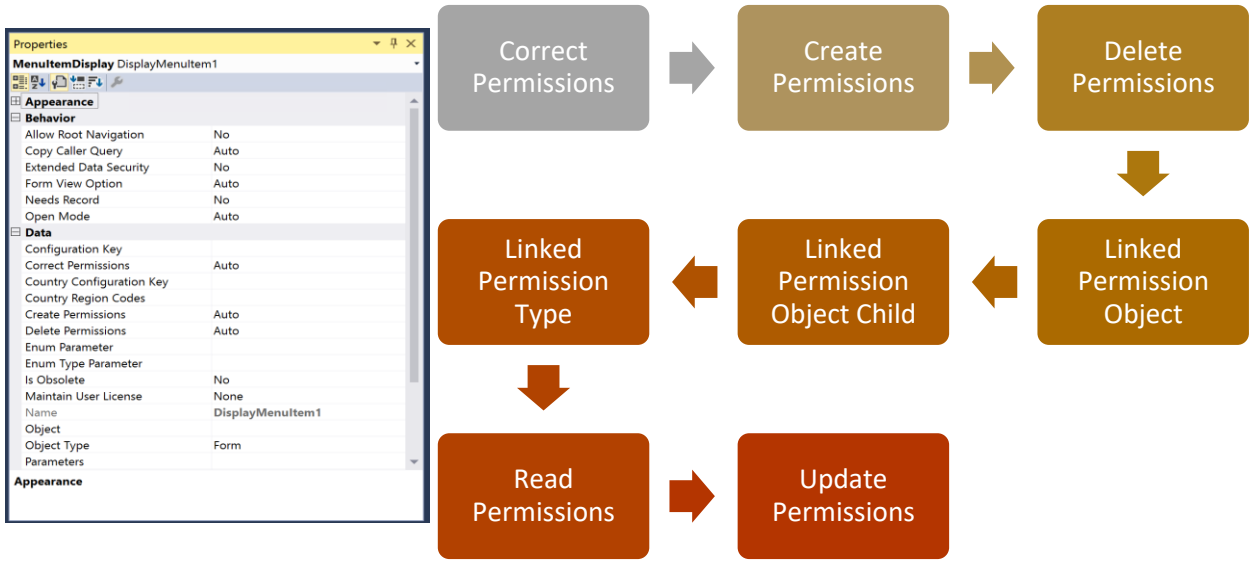
When this property is set on a container control, modifications are disabled or enabled for all controls within the container.

- **Needed Permission** - Specifies the minimum permission level needed to allow access to the control. The values for the NeededPermission property represent a hierarchy. Read is the weakest permission, and Delete is the strongest. Read is included by Update, which is 1 level higher in the

hierarchy. The CREATE permission is above both the read and update permissions, therefore read and update are both included with the create permission.

- **Mandatory** - Determines whether users must enter data in the control.
  If Mandatory is set to Yes, a red wavy line is displayed on empty fields, and a warning is displayed if the user tries to navigate away from the field without completing it.
  The values for the NeededPermission property represent a hierarchy. Read is the weakest permission, and Delete is the strongest. Read is included by Update, and Update is included by Create. Therefore Read is also included by Create permission.

MENU ITEMS



- **Correct permissions**: Specifies whether correct permission will be available to select when privileges are assigned to the menu item. The options are as follows:
  Auto – the permission will be available to select as a privilege on this menu item Privileges node under the Entry Points node.
  No – the permission will not be available to select as a privilege on the menu item.
  The default value is Auto.

- **Create permissions**: Specifies whether create permission will be available to select when privileges are assigned to the menu item. The options are as follows:
  Auto – the permission will be available to select as a privilege on this menu item Privileges node under the Entry Points node.
  No – the permission will not be available to select as a privilege on the menu item.
  The default value is Auto.

- **Delete permissions**: Specifies whether delete permission will be available to select when privileges are assigned to the menu item. The options are as follows:
  Auto – the permission will be available to select as a privilege on this menu item Privileges node under the Entry Points node.
  No – the permission will not be available to select as a privilege on the menu item.

The default value is Auto.

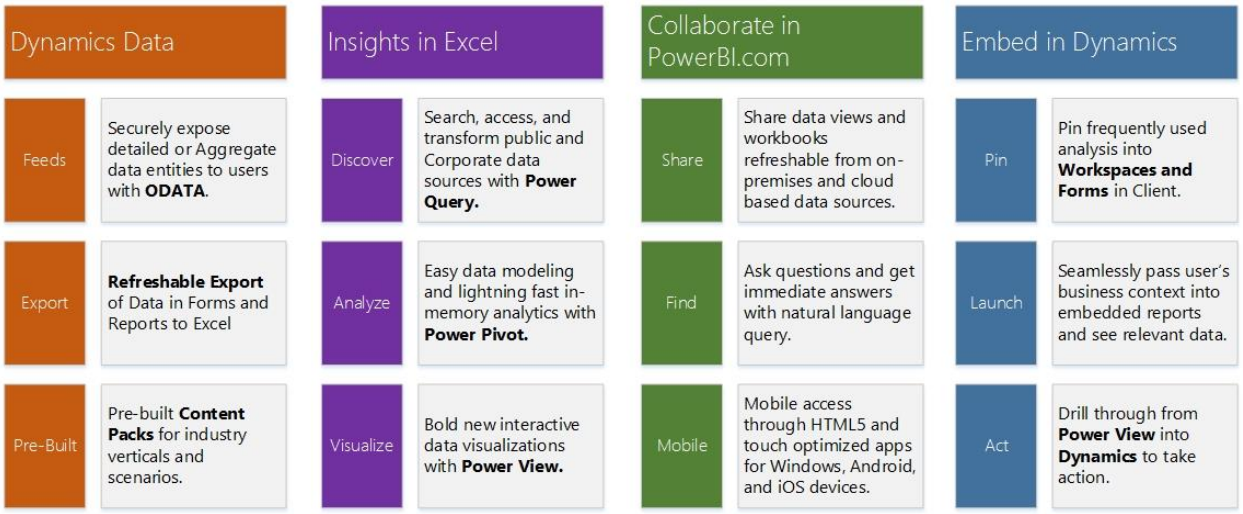
- **Linked Permission Type**: Specifies the type of the object pointed to by the LinkedPermissionObject property. For example, this could be a Form.
  Linked Permission Object Child
  Linked Permission Object: Specifies the name of another object (for example, a form or report) whose permissions are to be applied to this menu item. Typically used with Action menu items.

- **Read permissions**: Specifies whether read permission will be available to select when privileges are assigned to the menu item. The options are as follows:
  Auto – The permission will be available to select as a privilege on this menu item Privileges node under the Entry Points node.
  No – The permission will not be available to select as a privilege on the menu item.
  The default value is Auto.
  Update permissions:
  Specifies whether update permission will be available to select when privileges are assigned to the menu item. The options are as follows:
  Auto – The permission will be available to select as a privilege on this menu item Privileges node under the Entry Points node.
  No – The permission will not be available to select as a privilege on the menu item.
  The default value is Auto.

Module 17: Introduction to Advance Topics

Lesson 1: Introduction to Business Intelligence

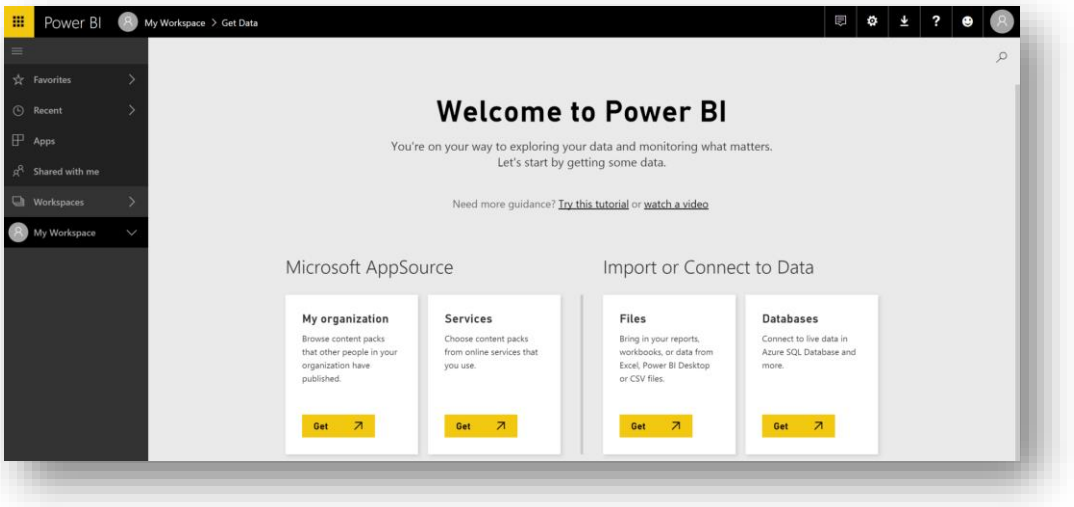
- Power BI Integration



Microsoft Power BI is a collection of tools and services that enables interactive visualizations and dashboards. Power BI is the tool that many users choose when they want to create interactive visualizations and self-service reports for Microsoft Dynamics 365 for Finance and Operations, Enterprise Edition You can access Power BI visualizations from PowerBI.com with a free account.

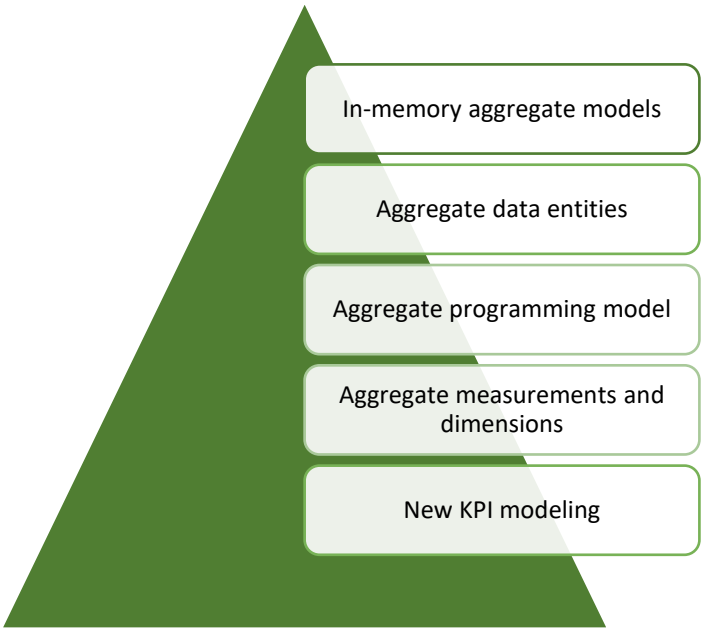
Dynamics 365 for Finance and Operations, Enterprise Edition integrates with Power BI by being a first-class source of data to Power BI authoring tools such as Microsoft Excel. After reports are authored, they can be published to PowerBI.com for collaboration and mobile access. Although dashboards and reports can be used on PowerBI.com, they can also be pinned to the Dynamics 365 for Finance and Operations, Enterprise Edition client to provide interactive visuals that are related business processes.

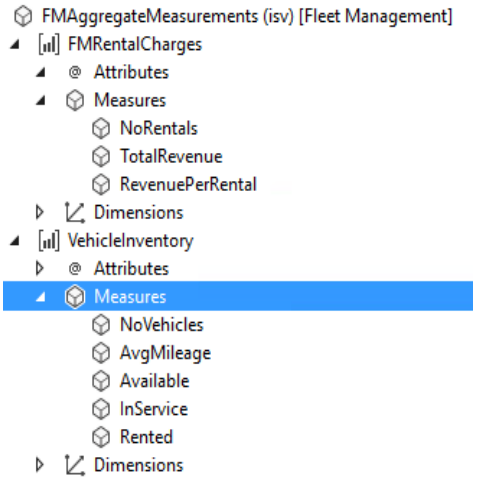
Pre-built Dynamics 365 for Finance and Operations, Enterprise Edition content packs from Microsoft, and also from independent software vendors (ISVs) and partners, offer ready-made analytical dashboards and reports for industry verticals. Users can deploy these content packs from PowerBI.com.



Tiles and charts created in Power BI can be pinned in the e client as well. As users view reports and visuals on PowerBI.com, they can drill back into the Dynamics 365 for Finance and Operations, Enterprise Edition client to take action on trends and exceptions. Users see only data that is relevant to them, and that they have access to in Dynamics 365 for Finance and Operations, Enterprise Edition

- BI Features





In memory real time aggregate models have replaced SSAS cubes in Dynamics 365 for Finance and Operations, Enterprise Edition.

Aggregate data entities are read-only data entities that are used for reporting purposes. To consume aggregate data when you build charts and other client controls, add the aggregate data entity to a form as a data source. You can also consume aggregate data entities programmatically.

Aggregate programming model enables a developer to consume aggregate data programmatically using either X++ or C# code. Data retrieved using Aggregate programming model can be used as a data source in forms.

An aggregate measurement is a model that contains a collection of measures together with their corresponding dimensions. Measures are aggregate numbers, such as Total Sales or Number of Orders. Dimensions are slicers, such as Product, Vendor, or Customer, that help you analyze the measure. For example, the measure of Total Sales isn't useful unless it can be sliced by Product, Region, and Customer.

Aggregate measurements are the evolution of (AX 2012) analysis cubes.

Method expressions enable a developer to build rich expressions using aggregate data. Method expressions are written in X++. KPIs can be modeled using Method Expressions thereby eliminating the need to build MDX expressions.

Aggregate dimensions are shared across a Dynamics 'e' implementation. Aggregate measurements associate themselves with relevant aggregate dimensions. For example, Total Sales can be associated with Customer, Product, and Sales Region dimensions. However, Total Sales can't be associated with Vendor and Warehouse dimensions.

Aggregate dimensions and aggregate measurements are modeled by using Dynamics 365 for Finance and Operations, Enterprise Edition Visual Studio tools.

In Dynamics 365 for Finance and Operations, Enterprise Edition users can use a rich client form to modify a KPI definition. Users can also define new KPIs by using the aggregate data that is contained in aggregate measurements. After a KPI is defined, users can customize it at run time.

- Types of BI
  1. Contextual BI
     This refers to providing required insights as part of the user experience so that the user has relevant insights to not only achieve the task but be highly productive during the course of the day.
  2. Embedded BI
     This refers to the analytic content being embedded within the Finance and Operations user experience. Contextual BI and embedded BI teams are closely related
  3. Self-Service BI
     This refers to enabling the user to tweak existing or create new analytic content such as reports, KPIs and dashboards.

Lesson 2: Introduction to Reporting Services

- Reporting Services Overview
  In Dynamics 365 for Finance and Operations, Enterprise Edition reporting services are hosted in the Microsoft Azure Compute service. If you're developing in a one-box environment, the services also run locally in the Azure compute emulator.

  In a one-box environment, developers can modify, create, and preview reports, from end to end, in Microsoft Visual Studio 2013. A separate process isn't required to add reports to the Application Object Tree (AOT) on the application server. Changes to reports that are packaged together with other solution updates are uploaded and deployed to the cloud only after development is completed in the local environment.

The enhanced report viewing experience for end users in the Microsoft Dynamics 365 for Finance and Operations, Enterprise Edition client is the same as the report preview in Microsoft Visual Studio. By pressing Ctrl+F5, you can build and preview the report in an Internet Explorer window. The report appears exactly as it would appear in the client.

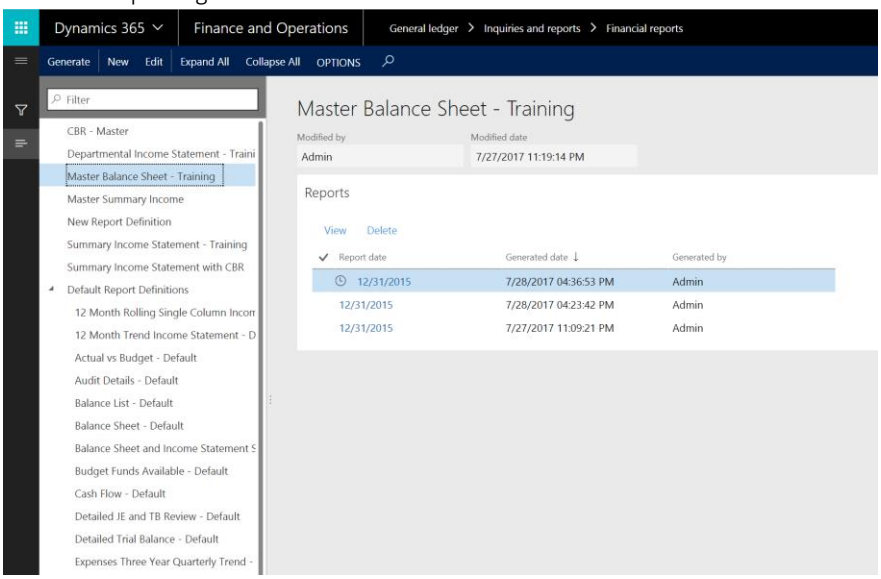
- Development Process
  You don't have to set up or administer a local report server for development, because the reporting services run in the Azure compute emulator, together with the application server.
  After reports have been deployed to the local reporting services, they can be accessed from the client.
  The process of developing a report is easier in Microsoft Dynamics 365 for Finance and Operations, Enterprise Edition because you can create and validate a reporting solution entirely within Visual Studio 2013.

  1. Create a reporting project and the query in Visual Studio.
  2. Edit the report in Visual Studio.
  3. In Visual Studio, add the report to a menu item, and set the menu item as a startup object.
  4. Use the AOT to deploy the report to the report server.
  5. Press Ctrl+F5 to verify the report in the Dynamics 'e' client.
  6. When the hole solution is completed, deploy it to the cloud in one package.

  Note: There is no longer a separate preview of the report design from the model editor.

- Advantages of SSRS Reports
  1. Back office capabilities - such as email support, scheduled executions via batch, and print archive.
  2. Precision designing
  3. Parameterized views and drill-throughs

- Financial Reporting



The Financial reporting area is where users can access 22 default financial reports.

These reports were previously accessed using an integrated reporting tool called Management Reporter. With e, the Financial Report Designer is within the application. This feature allows users to run financial

statements, such as a balance sheet and income statements, directly from the e web client in the browser. The 22 default reports that come with e, these can be edited, and new ones can also be created.

The financial reporting menu can be found in the following places in Microsoft Dynamics e:

General Ledger > Inquiries and reports > Financial reporting.

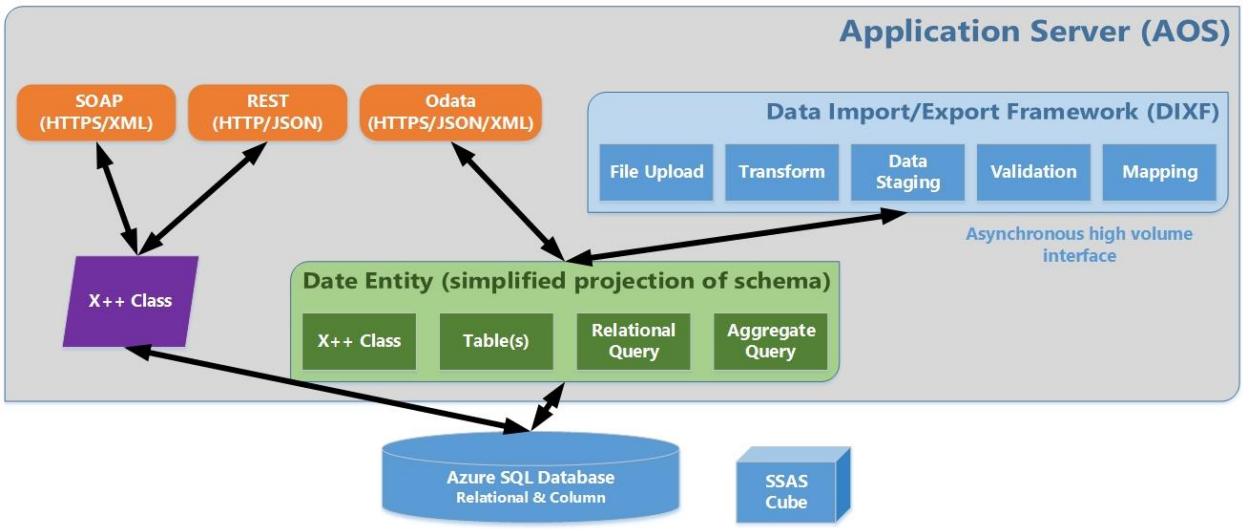
The financial reporting functions are available to users who have the appropriate privileges and duties assigned to them in order to run reports. Users with the Security Administrator role assigned to them can view, edit and manage the financial reports.

In order to create and generate financial reports for a legal entity, the following needs to be set up for that legal entity:

Fiscal calendar
Ledger
Chart of Accounts
Currency

Lesson 3: Introduction to Services and Integration

- Service and Integrations Overview



All the service groups under the AOTService group node are automatically deployed.

All services that must be deployed must be part of a service group. The diagram lists all the service endpoints that are available in Dynamics 'e 7.'

Consuming external web services is also supported by creating a class library project in Visual Studio.

- Open Data Protocol (Odata) V4
  OData is a standard protocol for creating and consuming data. OData applies web technologies such as HTTP and JavaScript Object Notation (JSON) to provide access to information from various programs.

  It supports Uniform URL conventions for features like filtering and sorting data. More information on ODATA URL syntax can be found at odata.org

OData entities in Dynamics ‘ax 7′ are based on the concept of an updatable view. When the **IsPublic** property for an updatable view is set to **TRUE**, that view is exposed as a top-level OData entity.

e introduces a public OData service endpoint that enable access to Dynamics ‘e′ data in a consistent way across a broad range of clients.

- Office Integration

  Excel Data Connector app

  Microsoft Excel can change and quickly analyze data. Dynamics ‘e′ includes an Excel Data Connector app that interacts with Excel workbooks and Dynamics ‘e′ OData services that are created for publicly exposed data entities. The Excel Data Connector app enables Excel to become a seamless part of the Dynamics ‘e′ user experience.

  The Excel Data Connector app is built by using the Apps for Office framework. It runs in a task pane that contains all app-related actions and information. Office applications are web applications that run inside an embedded Internet Explorer browser window.

  The excel data connector app supports data refresh and publishing back to e.

  The Excel Data Connector app is used in three primary scenarios:

  Workbook Designer – In Workbook Designer, the user can select an entity and some fields. The entity and its fields are bound to the workbook. Any data changes can be published back, provided that all the key fields have been included.

  Custom-generated export – Developers can use the Export application programming interface (API) to add a custom export to any page. These custom exports make it easier to edit the data in Excel.

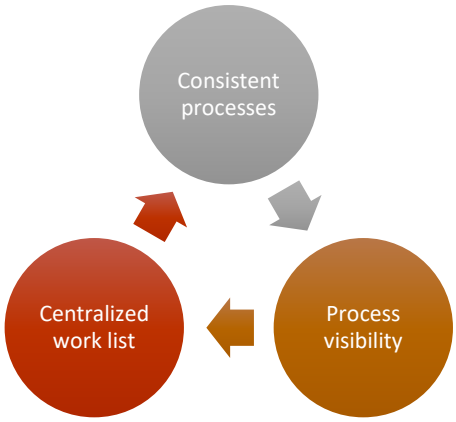
  Template export – Developers can use the Export API to add a template export to any page. These template exports are like custom-generated reports, but they use a predefined template. These template exports also make it easier to edit the data in Excel.

  The export to excel feature is a static export of grid data, directly to an excel spreadsheet. This is useful for quick analysis of data.

- Workflows

  Another integrated feature of e is the workflow editor. A workflow represents a business process. It defines how a document flows, or moves, through the system by showing who must complete a task, make a decision, or approve a document. For example, the following figure shows a workflow for expense reports.

  The workflow editor is a program that you click one time to download. The editor communicates with Dynamics ‘e′ by using services. Therefore, Dynamics ‘e′ can carry a rich, graphical workflow design experience. Workflow development wizards have been ported into Microsoft Visual Studio.



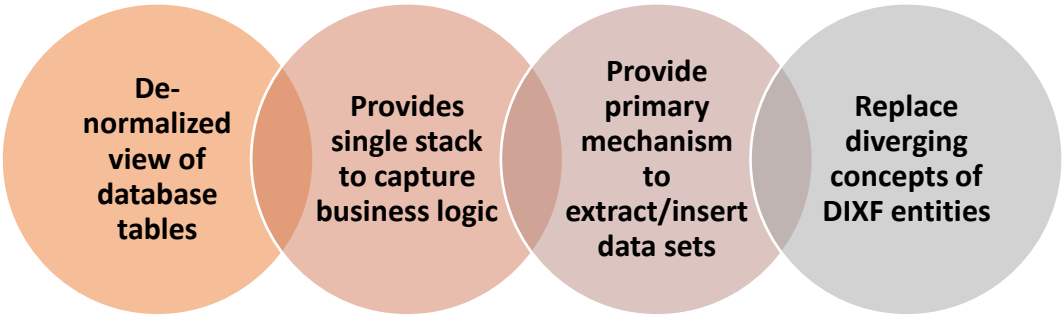
Consistent processes – You can define how specific documents, such as purchase requisitions and expense reports, are processed. By using the workflow system, you ensure that documents are processed and approved in a consistent and efficient manner.

Process visibility – You can track the status, history, and performance metrics of workflow instances. This helps you determine whether changes should be made to the workflow to improve efficiency.

Centralized work list – Users can view a centralized work list that displays the workflow tasks and approvals that are assigned to them. This work list is available from the Role Center pages in the Microsoft Dynamics 365client and Enterprise Portal for Microsoft Dynamics 365 for Operations

Lesson 4: Introduction to Data Entities

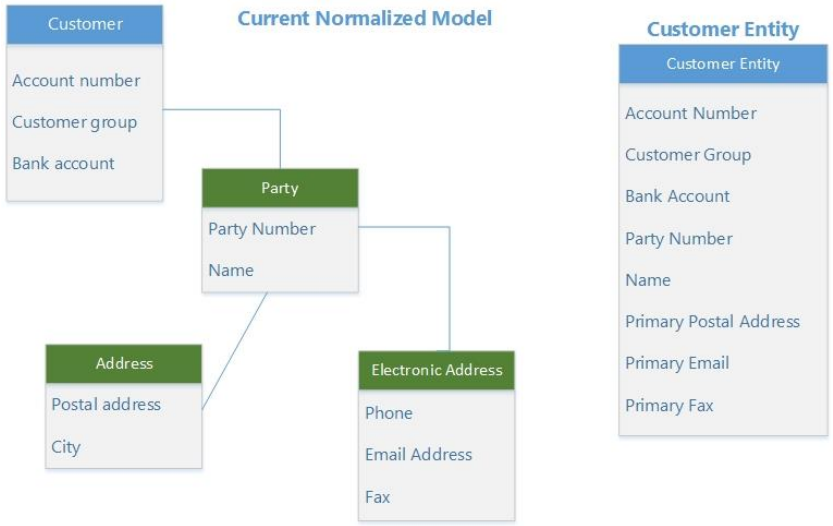
About Data Entities



A data entity is an abstraction from the physical implementation of database tables. For example, in normalized tables, a lot of the data for each customer might be stored in a customer table, and then the rest might be spread across a small set of related tables. In this case, the data entity for the customer concept appears as one de-normalized view, in which each row contains all the data from the customer table and its related tables.

Single framework to be used in multiple scenarios.

- De-normalized View



For example, a normalized table. A lot of the data for each customer might be stored in a customer table and the rest might be spread across a small set of related tables. In this case, the data entity for the customer concept appears as one denormalized view in which each row contains all the data from the customer table and its related field.

- Data Entity Categories

Parameter - Functional or behavioral parameters.
Required to set up a deployment or a module for a specific build or customer.
Can include data that is specific to an industry or business. The data can also apply to broader set of customers.
Tables that contain only one record, where the columns are values for settings. Examples of such tables exist for Account payable (AP), General ledger (GL), client performance options, workflows, and so on.
Reference - Simple reference data, of small quantity, that is required to operate a business process.
Data that is specific to an industry or a business process.
Examples include units, dimensions, and tax codes.
Master - Data assets of the business. Generally, these are the "nouns" of the business, which typically fall into categories such as people, places, and concepts.
Complex reference data, of large quantity. Examples include customers, vendors, and projects.
Document
Worksheet data that is converted into transactions later.
Documents that have complex structures, such a several line items for each header record. Examples include sales orders, purchase orders, open balances, and journals.
Transaction
The operational transaction data of the business.
Posted transactions. These are non-idempotent items such as posted invoiced and balances. Typically, these items are excluded during a full dataset copy.
Examples include pending invoices.

- Data Entity Use Cases

First it's synchronous service or OData. Data entities enable public Application Programming Interfaces or APIs on entities to be exposed, which enable synchronous services. Synchronous services are used for a few different purposes. For one, it can be used for Office integrations or third-party mobile apps. Next is asynchronous integration. Data entities also support asynchronous integration through a data management pipeline. This enables asynchronous and high-performing data insertion and extraction scenarios. So this can include interactive file-based import export or recurring integration, so file queue and so on. Next, we also use this for data migration as well as business intelligence. Data entities can be added to a project just like any other element. When you initially add it, the data wizard will guide you through the process of adding and configuring the data entity. At completion, Visual Studio will create a data entity along with an optional staging table. You can also right click a table in the designer in Visual Studio and choose add-ins and create a data entity. Let's take a look in Visual Studio.

- Creating Data Entities

  Data entities can be added to a project just like any element. When you initially add it, the data entity wizard guides you through the process of adding and configuring it. At completion, it will create a data entity along with an optional staging table.

  You can also right click a table in the table designer in Visual Studio and choose Addins > Create data entity

Module 18: Code Extensions

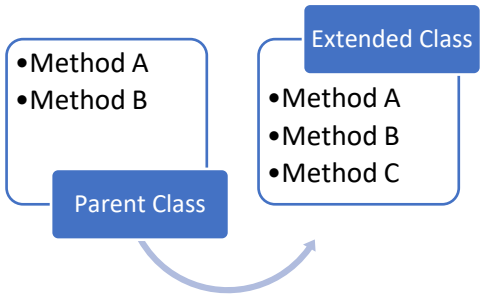
Lesson 1: Extension Method

- Extensions

  When we create a model in a new package, this indicates that it is a new separate assembly. Our model or package can reference other packages and models and, therefore, other elements. However, a specific element can only exist in one model. So we can reference the ven table from the Application Suite model in our new model, but we cannot customize that element as part of the Application Suite model. We can only change that specific element or XML file if we are overlaying in the Application Suite model. So if we choose to customize an element in a different model, we have to create an extension of that element. In order to be able to create an extension of an element from another model, we must ensure the model is being referenced in our new model

  The extension approach is a best practice as it cleanly puts our customizations into a separate assembly. It does not touch the standard application at all. Therefore, we do not need to compile the entire Application Suite. Also, this improves design time performance and reduces the cost associated with servicing and updating environments.

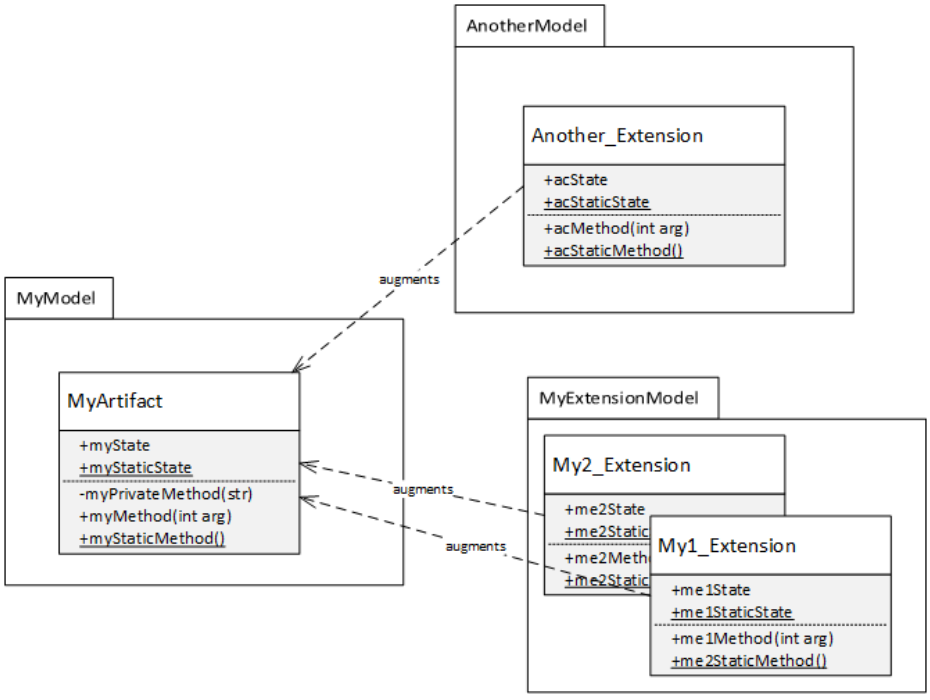
- Class Augmentation



A similar code extension mechanism already exists for X++ and is modeled after the corresponding feature in C#. Under this mechanism, a class can be designated as an extension class through a naming convention and by hosting public static methods. In the existing feature, the type of the first argument that is passed to the extension method is the type to extend. What this module describes is the next step in that direction, which offers a more capable and natural extension story.

In objected-oriented programming, the term extend has a well-defined meaning. If we say, "class B extends class A," we mean that B inherits from A, and the usual object-oriented rules are implied. In fact, this term is even used in the X++ syntax that is used in class declarations to express this relationship. At the same time, we use the term extension to talk about metadata that has contributions from several models. To avoid overloading the term extend, we will instead use the term class augmentation 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. Nevertheless, we will also continue to use the term extension class, because it's prevalent.

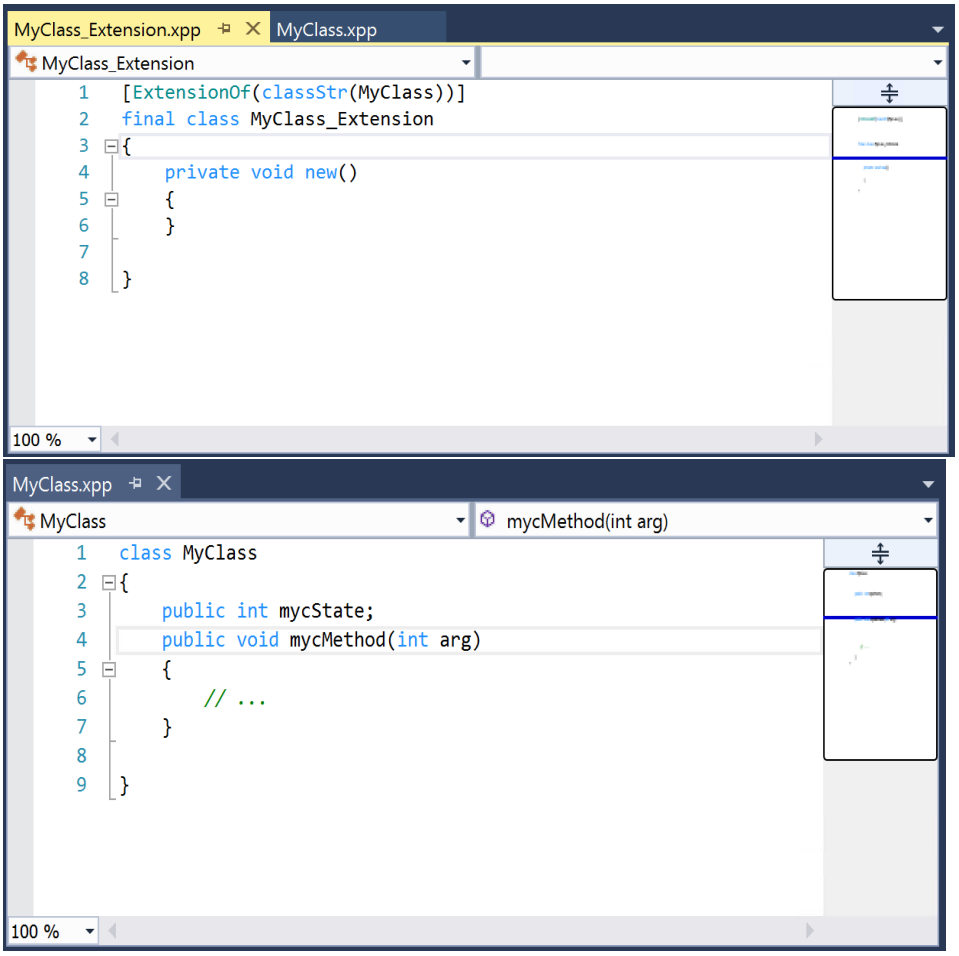
- Effective Class Concept



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.

In this example, the effective class is the class in the extension models that contains all the original methods and all the public artifacts from the extension classes. The effective class isn't the same in every model, because it includes only the class extensions that are defined in a given model. The following illustration shows the effective class of MyArtifact in the MyExtensionModel model.

- Extension Class Declarations



Extension classes are classes that are adorned with the ExtensionOf attribute and that also have a name that has the _Extension suffix. (This restriction on the naming might be removed later.) The name of the extension class is otherwise unimportant. The class augments the artifact that is specified in the ExtensionOf attribute, as shown in the following example.

Because the classes are instantiated by the runtime system, it's not meaningful to derive from the extension class. Therefore, the extension class must be marked as final. The classStr compile-time function must be used and has two purposes:

  - It produces a compilation error if the MyClass class doesn't exist.

- The compile-time function that is used tells the compiler what kind of artifact is augmented. Artifact names by themselves don't uniquely identify a given artifact to augment. For example, forms can have the same names as tables, classes, and enums.

Any number of extension classes can augment a given artifact in a particular model. Extension classes are never referenced directly by the programmer, only by the runtime system.

Any class that inherits from an augmented class also inherits the effective class. In other words, the classes that inherit from a class that has extensions inherit the methods that are defined in the extension classes.

Lesson 2: Static and Instance method

- Constructors

  Instance constructor

The instance constructor is the method that is named **new**. The instance constructor that is defined in an extension class can't have parameters. Instances of the extension classes are created, and the runtime system calls their constructors as required by the usage scenario. These constructors are never explicitly called by your code.
Constructors are useful for initializing the state of the extension objects. It's guaranteed that the constructor that is provided in an extension class will be called one time before any instance method or the instance state on the extension class is accessed. However, if no such references are made, the constructor isn't called.

Static constructors

Static constructors are the parameter-less static methods that are named **typenew**. Static constructors can be defined on extension classes. It's guaranteed that the runtime system will call the constructor before the first reference to the extension type. You can't assume any particular order of invocation for static construction among a set of extensions.

- Methods

The public methods that are defined in extension classes provide additional functionality to the augmented class in the context of the model where the extension class is defined. Only public methods are exposed in this way. You can define private methods to help implement the public methods, but those private methods aren't part of the effective class. Because extension classes are final, methods can't be marked as protected.

Instance method

```
[ExtensionOf(classStr(MyClass))]

final class MyClass_Extension

{

    private void new()

    {

    }

    public int ExtensionMethod(int arg)

    {

    }

}
```

This defines an extension method that is named ExtensionMethod and that augments MyClass.

The public instance method (ExtensionMethod) is defined in the extension class. Therefore, it's available just as if it were defined in MyClass in the context of the model where the extension class is defined. The following example shows how to call the method in the model.

MyClass c = new MyClass();

print c.ExtensionMethod(32);

Note that the instance method that is defined in the extension class is used as an instance method on the augmented artifact.

An extension method can access public members only from the artifact that it augments. This behavior is by design. No artifact should be able to interact directly with state and methods that are explicitly hidden through the private, internal, or protected keywords. Otherwise, direct interaction with explicitly hidden state and methods could cause malfunction by invalidating key implementation assumptions in those artifacts.

Methods and statements in the method body can use the this keyword. In this context, the type of this is the effective class of the augmented artifact.

Static method

```
[ExtensionOf(classStr(MyClass))]

final class MyClass_Extension

{

    private void new()

    {

    }

    public int method1(int arg)

    {

    }

    public static real
CelsiusToFahrenheit(real celsius)

    {

        return (celsius * 9.0 / 5.0)
+ 32.0;

    }

}
```

Methods that are defined as public and static in the extension class are available as static methods on the artifact that is augmented.

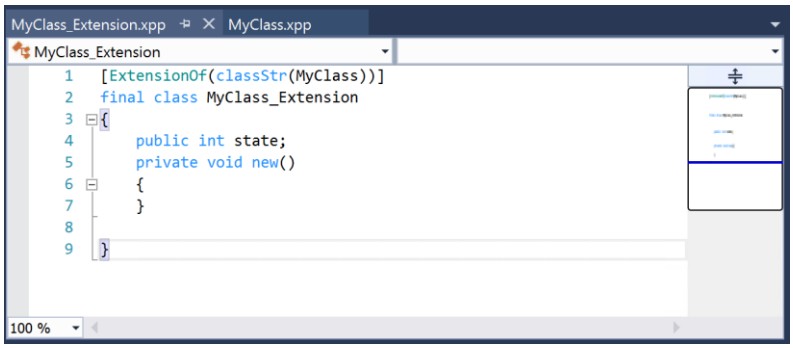
The following example shows how to call the method in the model.

var temp = MyClass::CelsiusToFahrenheit(20.0);

A static method can access the public static methods and state in the effective class of the augmented artifact.

As an interesting side effect, static extension methods on the Global class become available in the language as functions, which are available without any prefix.

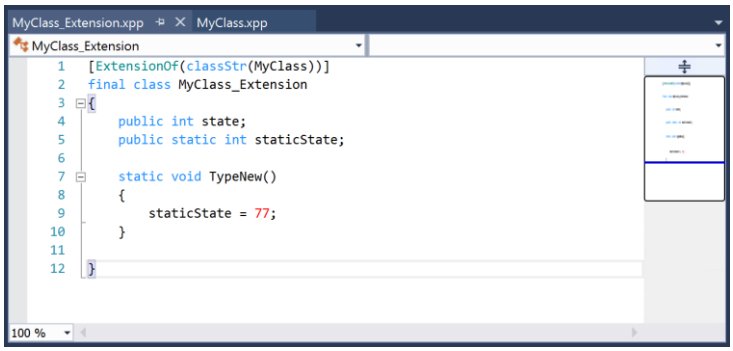
Instance state




```
//The following example shows how to use state in your code.
MyClass c = new MyClass();
c.state = 12;
```

Instance state, which is state that pertains to a particular instance of an artifact, can be specified on extension classes. The following example defines a state that is named state.

Static state



Static state applies to the type instead of an instance of the type. The following example shows a static extension state.

Lesson 3: Introduction to Chain of Command

Microsoft had introduced the Chain of Command with Platform Update 9. Chain of Command (CoC) enables strongly typed extension capabilities of public and protected methods. It is an amazing piece of development capability that allows technical consultants to extend the application avoiding over-layering.

Before we dive into the Chain of Command methods, remember that in order to use CoC you must declare your class as final, and your methods should always contain the next keyword.

Now let's check how it works with an example. Create two classes, one is the base class of the other and add a method we want to extend called 'testMe'. Add infolog calls inside the method to track the execution :

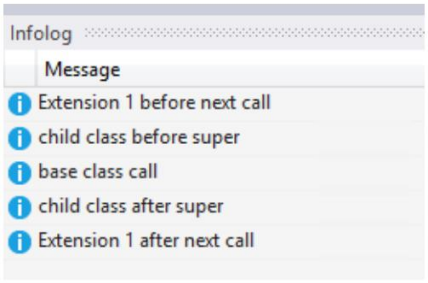
```
class COCBaseClass
{
void new(){}
protected void testMe()
{
info("base class call");
}
public void runTest()
{
    this.testMe();
}
}

class COCChildClass extends COCBaseClassclass
{
protected void testMe()
{
    info("child class before super");
    super();
    info("child class after super");
}
}
```

Create a new model, add reference for the class model we created above, and add an extension class for our child class.

```
[ExtensionOf(classStr(COCChildClass))]
final class COCChildClassCOCExt_Extension
{
protected void testMe()
{
    info("Extension 1 before next call");
    next testMe()
    info("Extension 1 after next call");
}
}
```

The method we added here is the new chain of command definition. We use exactly the same notation as the original method we are extending and add a mandatory "next" keyword to the call which will wrap our extension code around the extended method. This next keyword separates the pre and post parts of our extension and mandatory to be called inside the chain of command method. Next call cannot be placed in any kind of program block like if() block or while block. When you call next(), it checks for other extensions in the queue and runs them in random order, lastly running the base code.

Running the sample code by calling the runTest method will produce the following output.

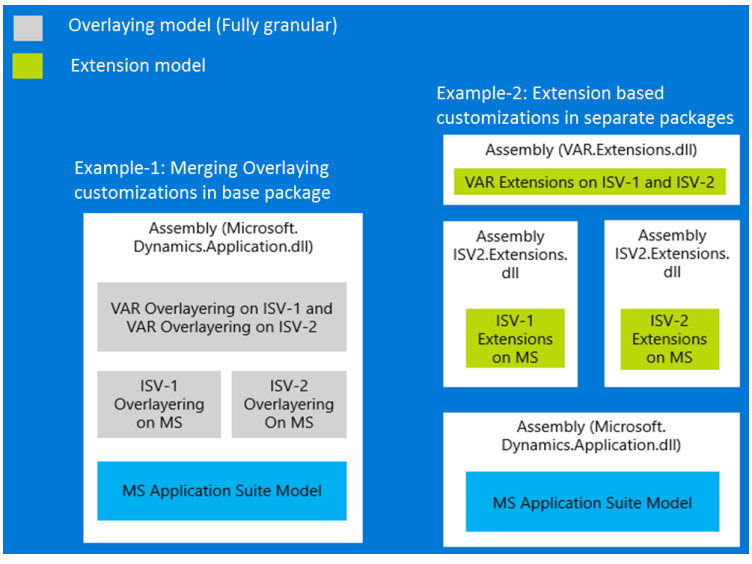


Lesson 4: Comparison to Overlaying

Overlaying is when you modify existing code by changing the system behavior.

Extensions are much more flexible. Extensions allow you to leave the system behavior in place while adding extra logic around it.

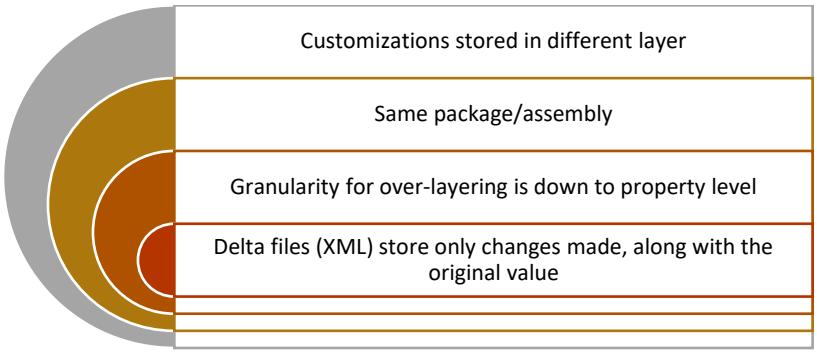
Overlaying may seem like the quicker solution, but it comes at the cost of upgrades, hotfixes, maintaining code and merging code.



Consider this diagram We can see on the left, that overlayering stores customizations in the same assembly, or DLL file. In contrast, the extension approach depicted on the right shows that each

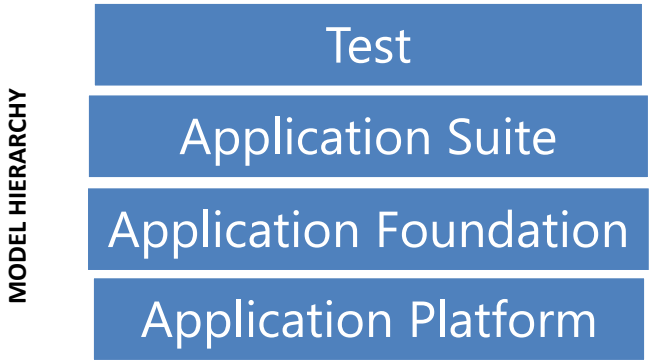
customization is stored in a separate assembly file, with separate extension elements. The ISV-1 and ISV-2 changes extend elements from the application suite and are stored in separate DLL files. Then, the VAR extension, stores customizations in elements that are extensions of the ISV-1 and ISV-2 elements.

This creates extension customizations in a separate package and does not affect the standard application at all, making a complete compile of all objects unnecessary.



- Overlaying may also be referred to as customizations.
- When we use the overlaying approach, we are storing our changes against a particular element in a higher layer, however it is within the same package, and therefore the same assembly.
- The granularity for over-layering is down the property level.
- Delta-files (XML) for the customized element store only the changes or customization, along with the original value.
- When using the extension approach, the purpose of layers changes. With the extension approach, models technically "over-layer" each other now.

Lesson 5: Using Delegates



Delegates allow us to access elements in higher models form a lower model.

With the model split, a hierarchy has been created where a higher model can take dependencies and access elements in the models below, but not in models above. In this setup, Application Suite has full access to its elements, Application Foundation's elements, and Application Platform's elements. Application Foundation can access its own elements and those of Application Platform. Finally, Application Platform can only access its own elements. We are also assuming we have created a custom model called test, which can access all elements in models below.

While the model split provides many benefits, it creates a problem when trying to access elements defined in higher models. Delegates are the recommended method for accessing elements in higher models from a lower model. Delegates are very similar to events in that when a delegate instance is invoked, a handler with compatible signature code is executed. This permits higher layer code, the handler, to be called by lower layer code, the delegate instance.

A delegate declaration must have three things:

1. The delegate keyword
2. Type void
3. Empty method

Delegate methods serve as a means for defining a contract between the delegate instance and the delegate handler. A delegate takes no action itself. This is enforced by having a void type and having no code in the method.

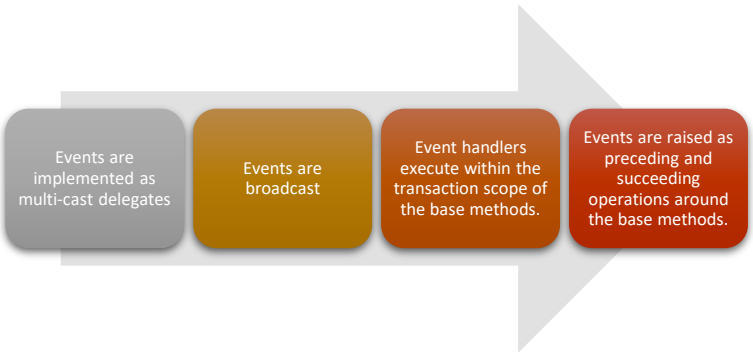
Module 19: Event Handling

- Framework Events

Tables, form data sources, form controls, and other element types that support extension events list the available events (and delegates) under an Events collection node.

For example, viewing the Events node of a table extension shows events that are defined by the framework, and delegate methods that are defined by application developers.

Events are exposed on the designer on different element and sub-element types, like table events, form events, form data source events, form control events, and others.



Events are implemented as multi-cast delegates, which means that more than one event handler can be subscribed to any particular event.

Events are broadcast; there's no sequencing of calls to event handlers.

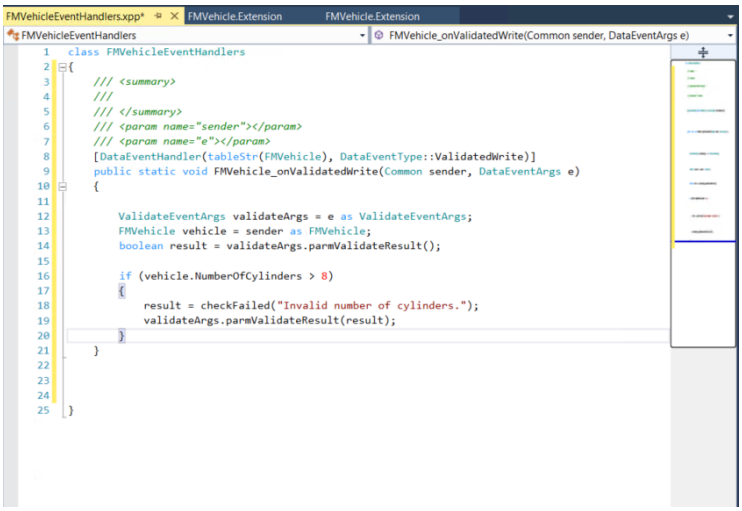
Event handlers execute within the transaction scope of the base methods.

Events are raised as preceding and succeeding operations around the base methods. This means that you have the opportunity to run code before a base method is called and after it has completed.

- Types of Events

  There are three types of events that you can subscribe to in Microsoft Dynamics 365 for finance and operations.
  - Standard events
    - Form Datasource from xFormRun
    - Get FormRun from form datasource
    - Get FromRun from form control
  - Delegates
  - Pre/Post Events

- Demonstration- Table events



1. Open Visual Studio as an administrator
2. Ensure that you Extension Testing model is open with the Extension Project Test.
3. Select the FMVehicle.Extension table.
4. Expand the **Events** node. The **Events** node lists all events that the table exposes. This includes events that are defined by the framework, and delegate methods that are defined by application developers.

   Note: Different framework events are exposed on the designers of many types of element and sub-elements, like table events, form events, form data source events, and form control events.
5. Right-click onValidatedWrite, and then select **Copy event handler method**. This step copies the event handler method signature to the clipboard
6. Add a new class named **FMVehicleEventHandlers** to the project. (Add > New item > Code > Class)
7. In **Solution Explorer**, double-click **FEVehicleEventHandlers** to open the code editor.
8. Right-click and paste the event handler method that you copied
9. Add the following code:

```
ValidateEventArgs validateArgs = e as ValidateEventArgs;
    FMVehicle vehicle = sender as FMVehicle;
    boolean result = validateArgs.parmValidateResult();
    if (vehicle.NumberOfCylinders > 8)
    {
       result = checkFailed("Invalid number of cylinders.");
       validateArgs.parmValidateResult(result);
    }
```

10. Save FMVehicleEventHandlers class and Build.
    Tip: You can paste and define your event handlers in any class of your model. The class FMVehicleEventHandlers is used only as an example.

- Demonstration- Form events
    1. Open Visual Studio as an administrator
    2. Ensure that you Extension Testing model is open with the Extension Project Test.
    3. Select the FMVehicleEventHandlers class, if not already open.
    4. Open
    5. Select the FMVehicle.Extension form.
    6. Right click to open.
    7. Expand Events
    8. Right click OnIntialized
    9. Copy event handler method
    10. Paste in to class

    - Form data source event handler

```
[FormDataSourceEventHandler(formDataSourceStr(EcoResProductDetailsExtended,
InventTable), FormDataSourceEventType::Written)]
public static void InventTable_OnWritten(FormDataSource sender,
FormDataSourceEventArgs e){
    FormRun                 form          = sender.formRun();
    FormDataSource          InventTable_ds =
 form.dataSource(formDataSourceStr(EcoResProductDetailsExtended,InventTable))
as FormDataSource;
   InventTable              inventTable    = InventTable_ds.cursor();
}
```

    - Form event handler Table Buffer on form closing event

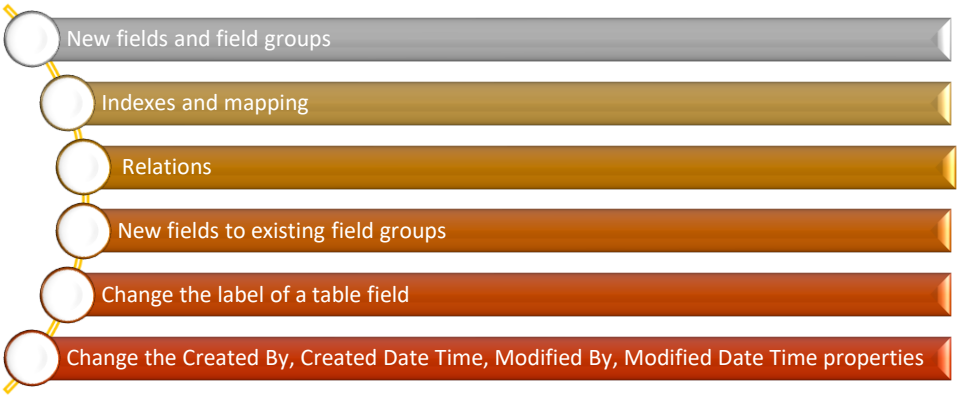
```
[FormEventHandler(formStr(EcoResAttributeValue), FormEventType::Closing)]
public static void EcoResAttributeValue_OnClosing(xFormRun sender,
FormEventArgs e)
{
    FormDataSource ecoResProduct_ds   =
sender.dataSource(formDataSourceStr(EcoResAttributeValue,
EcoResProductAttributeValue));
     EcoResProductAttributeValue      ecoResAttributeValue =
ecoResProduct_ds.cursor();
}
```

    - Control value and form event level for which auto declaration must be set true

```
[FormControlEventHandler(formControlStr(EcoResProductCreate, OKButton),
FormControlEventType::Clicked)]
public static void OKButton_OnClicked(FormControl sender, FormControlEventArgs
e)
{
    FormRun          element      = sender.formRun();
    //form control
    FormControl      modelGroupRef =
element.design(0).controlName("ModelGroupId");
     Info(strfmt("Model Group %1", modelGroupRef.valueStr()));
    //form parameter
    ItemId           itemId       = element.parmItemId();
}
```

Module 20: Table Extension Classes

- Table Extension



- Table Methods

  In Dynamics 365 for Finance and Operations, you use extensions to implement event handlers that are called from the base implementations of the table methods. The following tables list each table method and its events.

Published Table method	Preceding event	Succeeding event
validateWrite	ValidatingWrite	ValidatedWrite
validateDelete	ValidatingDelete	ValidatedDelete
validateField	ValidatingField	ValidatedField
validateFieldValue	ValidatingFieldValue	ValidatedFieldValue
modifiedField	ModifyingField	ModifiedField
modifiedFieldValue	ModifyingFieldValue	ModifiedFieldValue
Insert	Inserting	Inserted
Update	Updating	Updated
Delete	Deleting	Deleted
Initvalue	InitializingRecord	InitializedRecord

Validation events capture and return results by using the **DataEventArgs** parameter. The display and edit method modifiers are supported on table extensions.

Published Table method	Preceding event	Succeeding event
FinalDeleteValidation	Executed when a delete operation is performed on a table object, before the operation is committed to the underlying database table	N/A
FinalInsertValidation	Executed when an insert operation is performed on a table object, before the operation is committed to the underlying database table	N/A
FinalReadValidation	Executed when a read operation is performed on a table object.	N/A
FinalUpdateValidation	Executed when an update operation is performed on a table object, before the operation is committed to the underlying database table.	N/A

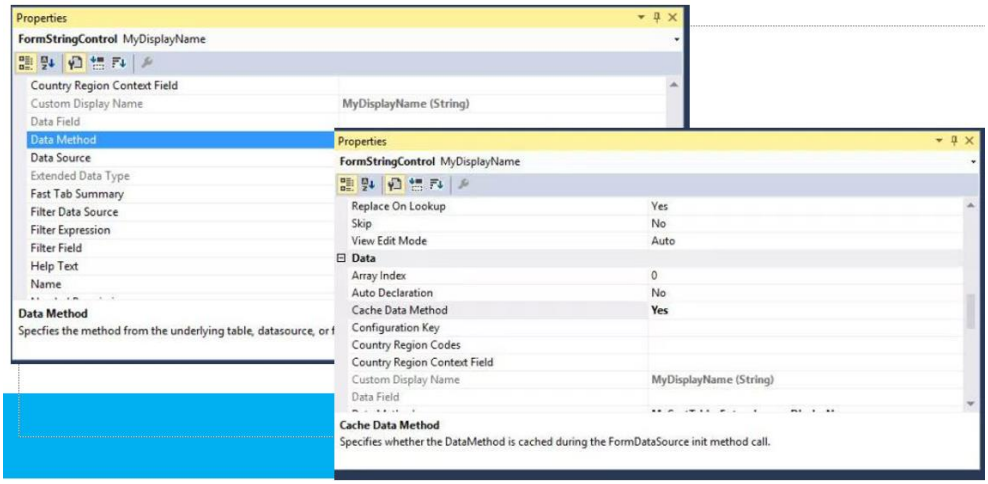
- Query Extensions

  You can extend a Query element to achieve the following:

  1. Add ranges to an existing data source.
  2. Add new (embedded) data sources to an existing data source.
  3. Add new fields to an existing data source.

- Add Display Method

  A **Display method** is used to display a calculation or data from another table. The method modifier display is added to the beginning of the method definition line to allow the return value to be displayed on a form. Display methods can be created either on a table or on a form DataSource. If it is created on the form DataSource, then it takes the table as a parameter. If possible, a display method should be created on the table itself, but there are situations where it has to be on the form DataSource, for example, if it uses an **unbound control on the form**. We don't have the option to create methods on table extensions, so we should use the extension class to do that. Extension methods enable you to extend tables by creating new display and edit methods on these.



Demonstration: Display Method on a Table Extension

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

a. **Note: <Class-name> - can be any name, but it is preferred to give the table name for which the extension is being created.*
b. *postfix <_Extension> is must.*

3. Click **Add**.
4. First create your table extension class and your display method following the example below:
   a. Code to be applied to the class:

```
**********************************************************

[extensionof(tablestr(MyCustTable))]
public static class MyCustTable_Extension
{
[SysClientCacheDataMethodAttribute(true)] //This attribute will cache your display method.
public static display Name myDisplayName(CustTable _this)
{
return _this.nameAlias() + "myDisplayName"; //Do here your display method as usual.
}

}
**********************************************************
```

5. To use your display method on the form, create your new field on the form extension and use the property as shown above of **Data Method** set to **MyCustTable_Extension::MyDisplayName** (Image 2).
6. To cache your display method on the form set on the field the property **"Cache Data Method" = yes** if you don't want to use the Attribute above (Image 3).
7. Build/Rebuild the project/solution and check the output in the URL.

Module 21: Form Extension Classes

Areas of Form Extension

In a form extension, you can:

- Add a new control.
- Enable or disable a control.
- Change the text or label property of a control.
- Change a control's visibility.
- Change a form's help text.
- Change a form's caption.
- Add a new data source.
- Change properties at the data-source level.
- Add a form part.

Other ways to customize a form, such as reordering controls in the form or subscribing to form or control events, are planned to be included in a future release.

Form Methods

In the current version, you use extensions to implement event handlers that are called from the base implementations of form methods.
The following tables list each method and its associated events.

Published form DataSource method	Preceding event	Succeeding event
active	N/A	Activated
delete	Deleting	Deleted
validateWrite	ValidatingWriting	ValidatedWrite
write	Writing	Written
create	Creating	Created
executeQuery	N/A	QueryExecuted
linkActive	N/A	PostLinkActive
init	N/A	Initialized
validateDelete	ValidatingDelete	ValidatedDelete
reread	N/A	Reread
selectionChanged	N/A	SelectionChanged
markChanged	N/A	MarkChanged
leaveRecord	LeavingRecord	LeftRecord

Form Object methods

Published form Object method	Preceding event	Succeeding event
init	Initializing	Initialized
close	Closing	N/A
run	N/A	PostRun
activate	N/A	Activated

Form Control methods

Published form Control method	Preceding event	Succeeding event
modified	N/A	Modified
validate	Validating	Validated
leave	Leaving	LostFocus
enter	N/A	Enter
gotFocus	N/A	GotFocus
clicked	N/A	Clicked
selectionChange	SelectionChanging	N/A
pageActivated	N/A	PageActivated
allowPageDeactivate	AllowPageDeactivate	N/A
expand	Expanding	Expanded
tabChanged	N/A	TabChanged
dialogClosed	N/A	DialogClosed

Demonstration: Extend the FMVehicle Form

1. Open Visual Studio as an administrator
2. Ensure that you Extension Testing model is open with the Extension Project Test.
3. Use **Application Explorer** to find the form named **FMVehicle**, and in the **Application Explorer** filter bar, enter the *FMVehicle type:form*.
4. Right-click the form, and then click **Create extension**.
5. Right click the new extension in your project and open
6. Add a new integer control named **NumberOfCylinders** to the **Attributes2** group control. You can find this control by expanding **Design > Tab > TabPageDetails > TabHeader > DetailsDetails > Attributes2**.
7. Right click Attributes 2, right click New and select Integer
8. Bind the new control to the **NumberOfCylinders** data field in the properties by setting the data field to NumberOfCylinders and Datasource to FMVehicle. (Easiest to type values) [Best practice would include setting label]
9. Save and Build the project.

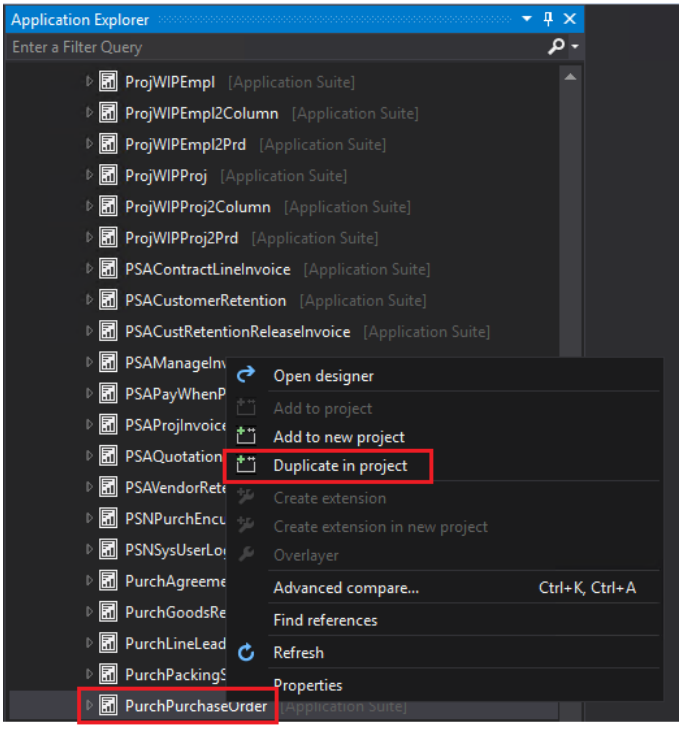
View in MsDyn365.

1. Open Dynamics 365 for Operations
2. Select Show navigation pane button
3. Select Fleet Management under Modules
4. Select Vehicles under Vehicles group
5. Select a Vehicle.
6. Expand the Details fasttab if not already
7. Observe new field under GPS enabled and before Doors.

Module 22: Reporting

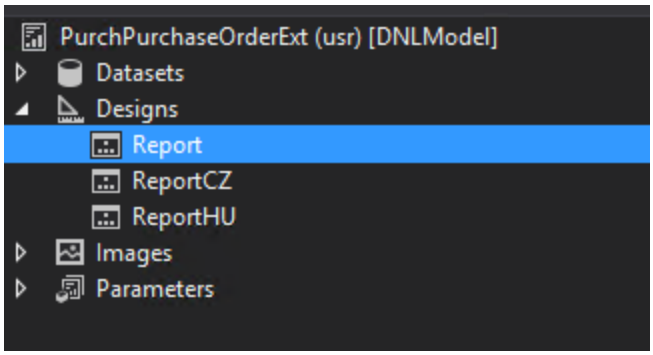
Lesson 1: Customize Standard Reports using Extension

1. In this sample, we'll customize Purchase Order. First, find PurchPurchaseOrder report in the AOT then right click → Select Duplicate in project (to automatically add report on your project).

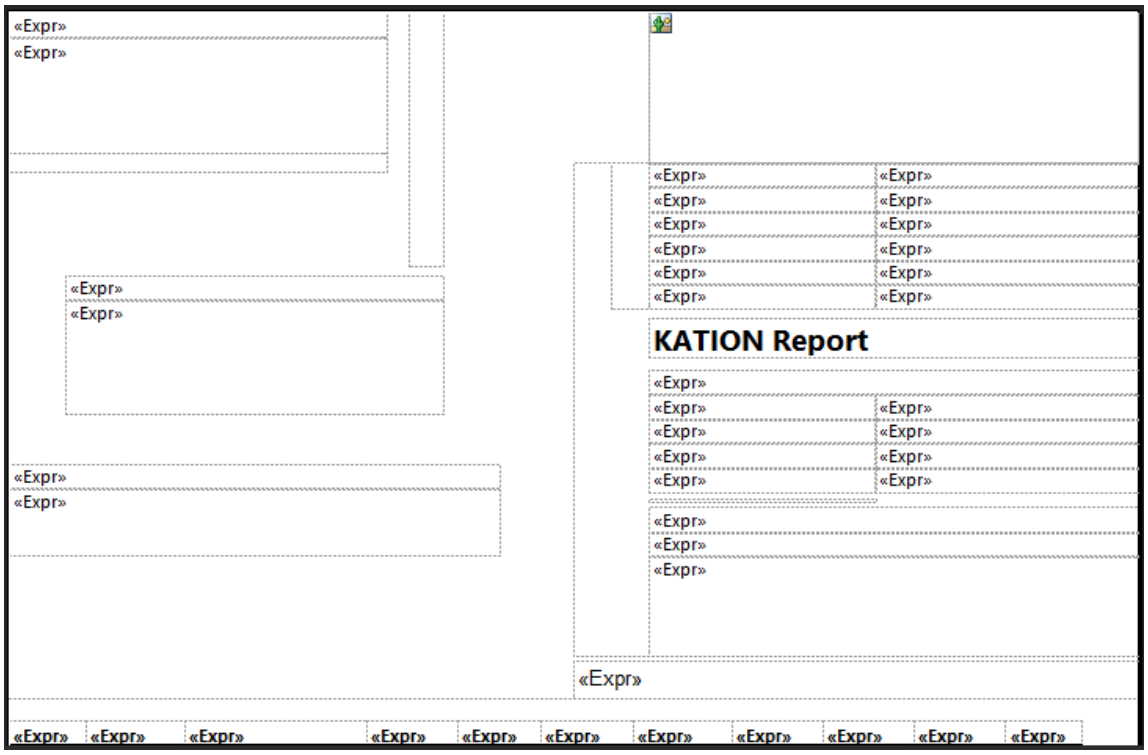


2. Rename the report to **PurchPurchaseOrderExt** to easily identify that this is an extension.

3. You can add designs to the existing report or customize the standard designs.
Note that this won't affect the original report. And we still need to set this up to use our new design.



4. Right click report design → Click edit using designer → Modify report design based on requirements.



5. Create a new class which extends PurchPurchaseOrderController, name the class as **PurchPurchaseOrderControllerExt**

```
class PurchPurchaseOrderControllerExt extends PurchPurchaseOrderController
{
}
```

- Add a constructor class

```
public static PurchPurchaseOrderControllerExt construct()
    {
        return new PurchPurchaseOrderControllerExt();
    }
```

- Copy **main method** of the original code and overwrite the initialization of report name.

Note: Codes in green color are overwritten.

```
public static void main(Args _args)
    {
       SrsReportRunController          formLetterController =
       PurchPurchaseOrderControllerExt::construct();
       PurchPurchaseOrderControllerExt   controller;

       if (TradeFormHelper::isCalledFromForm(_args,
formStr(VendPurchOrderJournalListPage)))
        {
            _args.record(VendPurchOrderJour::findRecId(_args.record().RecId));
```

```
        }

        controller = formLetterController;
        controller.initArgs(_args, ssrsReportStr(PurchPurchaseOrderExt, Report));

//controller.initArgs(_args,
PrintMgmtDocType::construct(PrintMgmtDocumentType::PurchaseOrderConfirmationRequest).getD
efaultReportFormat());

        if (classIdGet(_args.caller()) == classNum(PurchPurchOrderJournalPrint))
        {
            formLetterController.renderingCompleted +=
eventhandler(purchPurchOrderJournalPrint::renderingCompleted);
        }

        formLetterController.startOperation();
    }
```

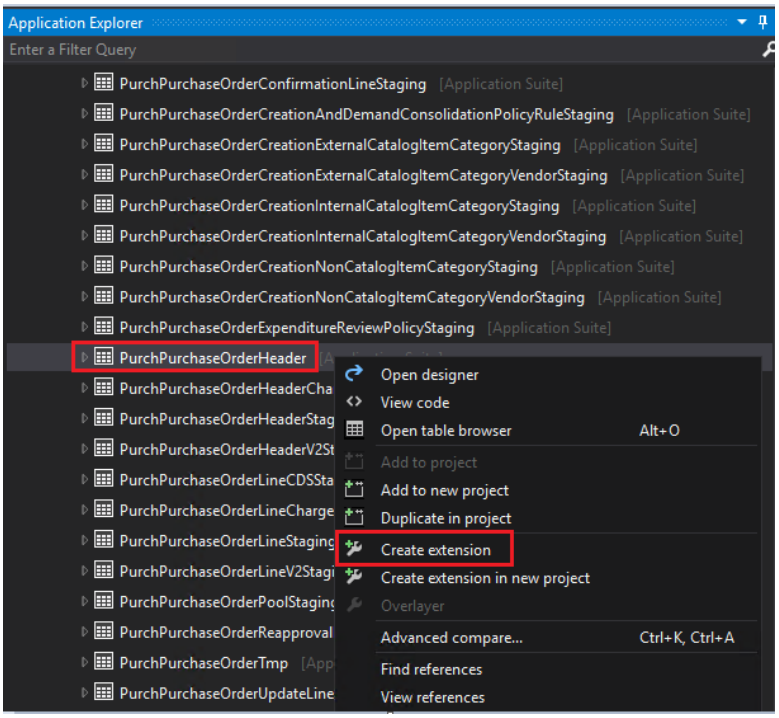
- Lastly, add an **output report** method. This will redirect the original report setting to your customized report design.

```
protected void outputReport()
    {
        SRSCatalogItemName  reportDesign;
        reportDesign = ssrsReportStr(PurchPurchaseOrderExt,Report);

        this.parmReportName(reportDesign);
        this.parmReportContract().parmReportName(reportDesign);


formletterReport.parmReportRun().settingDetail().parmReportFormatName(reportDesign);

        super();
    }
```

6. Next we must create a class for Print management handler. In AX 2012, we can easily add our customized reports under PrintMgmtReportFormat table → populate method. But in D365 we have to create a class and specify customized reports design and document type.

```
class PrintMgmtDocTypeHandlerExt
{
    [SubscribesTo(classstr(PrintMgmtDocType), delegatestr(PrintMgmtDocType,
getDefaultReportFormatDelegate))]
    public static void getDefaultReportFormatDelegate(PrintMgmtDocumentType _docType,
EventHandlerResult _result)
    {
        switch (_docType)
        {
            case PrintMgmtDocumentType::PurchaseOrderRequisition:
                _result.result(ssrsReportStr(PurchPurchaseOrderExt,Report));
                break;
        }
    }
}
```

6. If there's an additional field you need to add, just find the table → right click → Select create extension → rename table to **PurchPurchaseOrderHeader.Extension.** Then just add your required fields.



7. To populate your additional field, create an extension class of **PurchPurchaseOrderDP.**

Note:

- You always have to use the keyword [**ExtensionOf**())]
- Declare class as **Final**
- Add **_Extension** postfix on your newly created class name

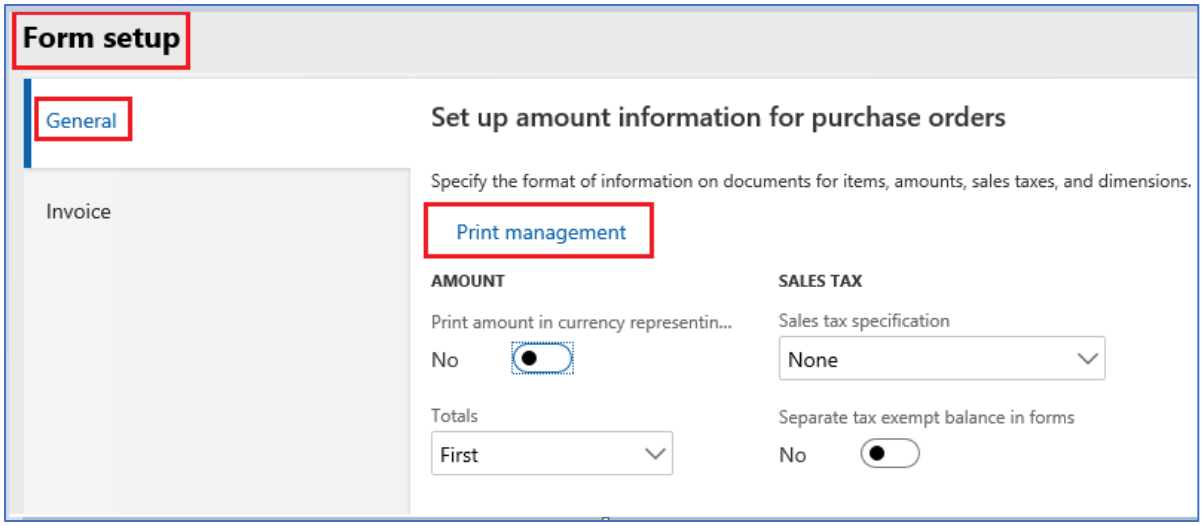
Sample code:

```
[ExtensionOf(classStr(PurchPurchaseOrderDP))]
final class KTI_PurchPurchaseOrderDP_Extension
{
    protected PurchPurchaseOrderHeader initializePurchaseOrderHeader(VendPurchOrderJour
_vendPurchOrderJour)
    {
        next initializePurchaseOrderHeader(_vendPurchOrderJour);

        PurchPurchaseOrderHeader purchPurchaseOrderHeader;
        PurchTable  purchTable;
        Name        docRef;

        this.setTransactionConnection(purchPurchaseOrderHeader);

        purchTable = _vendPurchOrderJour.purchTable();

        docRef = strFmt("%1 - %2 - %3", purchTable.PurchId, purchTable.OrderAccount,
purchTable.DeliveryDate);
```

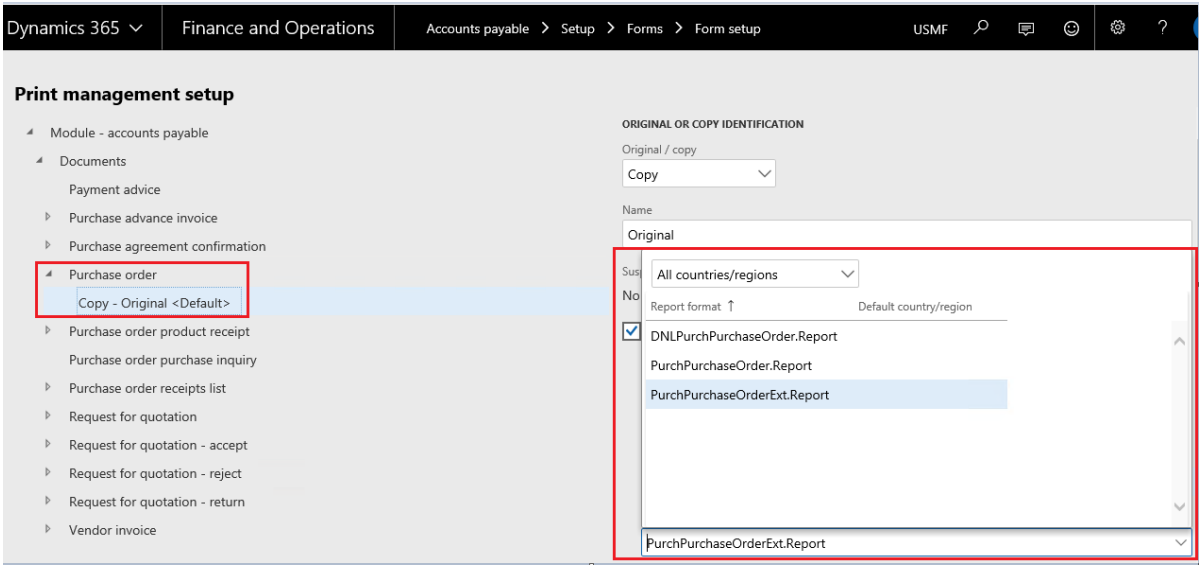
```
            purchPurchaseOrderHeader.DNL_DocRef = docRef;

            return purchPurchaseOrderHeader;
        }

}
```

8. Build and Sync your solution.

9. Next step is to setup our customized report on Print Management setup. Since we customized Purchase order report, we have to navigate to Accounts Payable > Setup > Forms > Click Form Setup



When Form Setup window opened, navigate to General > Click Print Management



In Print Management setup window, navigate to Purchase order fastTab > Click Copy – Original [Default] > Then change the value of Report Format field. Select your customized report.

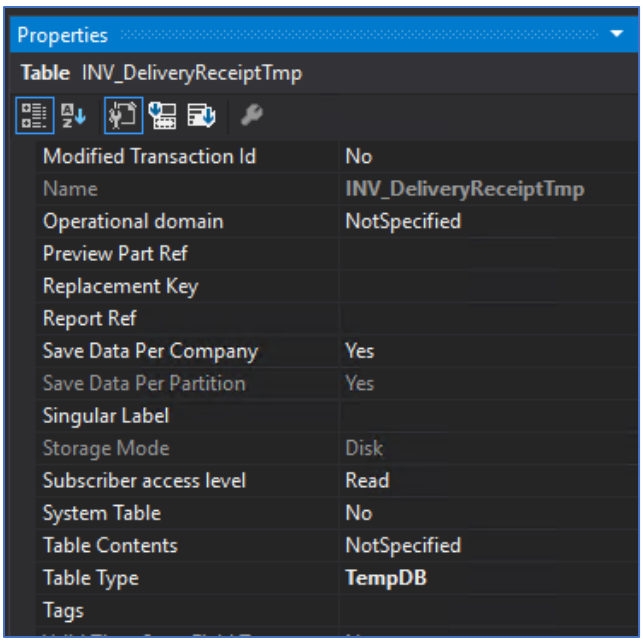
Lesson 2: Create Reports from Scratch

In creating a report from scratch, you'll have to consider page layout, page size, margins, if it is pre-printed or free form.

- Pre-printed – reports which already have a printed form layout. This one needs form shooting.
- Free form – reports which is from scratch, but all the design details will be made by developer.

Our example will be Delivery Receipt report generated from Movement Journal.

1. Create a temp table which will hold data upon generating the report.
Identify table name and table type (recommended table type for DP class is TempDB).
Then add fields needed on your report.



2. Data Provider (DP) class – all logics will be defined in this class including data insertion.

SAMPLE DP CLASS:

- Class Declaration

```
/// <summary>
/// SSRS sample Delivery Receipt
/// </summary>
[
    SRSReportParameterAttribute(classStr(INV_InventDeliveryContract_DNL))
  ]
    public class INV_InventDeliveryDP_DNL extends SRSReportDataProviderBase
{
    INV_DeliveryReceiptTmp  reportTmp;
```

- Process report method – where filters for queries or data will be applied. And contract initialization.

```
    //Process report
    [SysEntryPointAttribute]
    public void processReport()
    {
        INV_InventDeliveryContract_DNL  contract =   this.parmDataContract() as
INV_InventDeliveryContract_DNL;
        JournalId journalId;

        journalId = contract.parmJournalId();

        this.insertLedgerJournalTrans(journalId);
    }
```

- Insert data on temporary table

```
    //// <summary>
    /// Inserts journal entries from <LedgerJournalTrans> table
    /// </summary>
    /// <param name = "_journalNum"></param>
    /// <param name = "_voucher"></param>
    private void insertLedgerJournalTrans(JournalId  _journalNum)
    {
        InventJournalTable      inventJournalTable;
        InventJournalTrans      inventJournalTrans;
        CustTable               custTable;
        CustTrans               custTrans;
        CompanyInfo             companyInfo = CompanyInfo::find();

        inventJournalTable = InventJournalTable::find(_journalNum);

        reportTmp.CustomerName   = "CREATIVE DISHES & RECIPE INC.";
        reportTmp.CompanyPhone   = CompanyInfo.phone();
        reportTmp.DeliveryNotes  = "SAMPLE ONLY";
        reportTmp.CompanyAddress = companyInfo.postalAddress().Address;
        //reportTmp.Logo = FormLetter::CompanyLogo();

        while select inventJournalTrans
            where inventJournalTrans.JournalId == _journalNum
        {
            reportTmp.JournalId   = _journalNum;
            reportTmp.Unit        = inventJournalTrans.unitId();
            reportTmp.TransDate   = InventJournalTrans.TransDate;
            reportTmp.Qty         = InventJournalTrans.Qty;
            reportTmp.Description = inventJournalTrans.itemName();
            reportTmp.Voucher     =  InventJournalTrans.Voucher;
            reportTmp.insert();
        }

    }
```

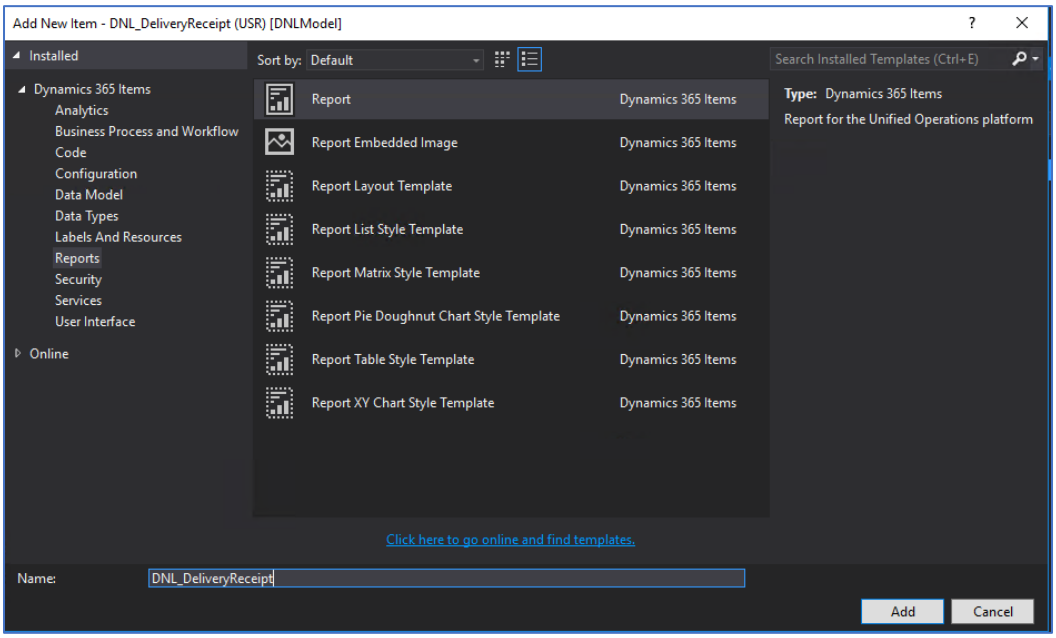
- Getter of data that will be used on SSRS report.

```
    //// <summary>
    /// Gets the full data from the <c>GFP_DisbursementVoucherTmp_FIN</c> temp
table.
```

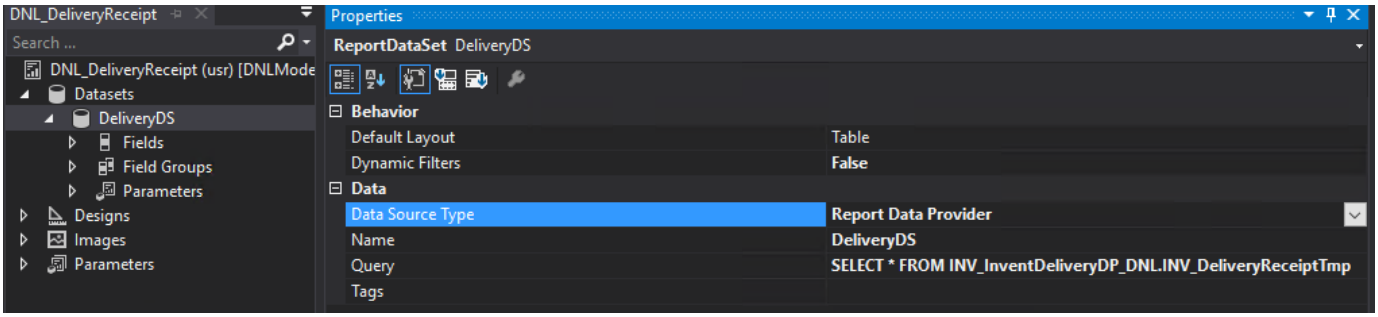
```
    /// </summary>
    /// <returns>
    /// The <c>GFP_DisbursementVoucherTmp_FIN</c> temporary table.
    /// </returns>
    [
        SRSReportDataSetAttribute(tableStr(INV_DeliveryReceiptTmp))
    ]
    public INV_DeliveryReceiptTmp getDeliveryReceiptTmp()
    {
        select reportTmp;
        return reportTmp;
    }

}
```

3. Create SSRS report design

- Right click project > Add > New item > Under Dynamics 365 Items, click Reports > Select Report > Specify report name
- Then click Add button to add this report on your solution.

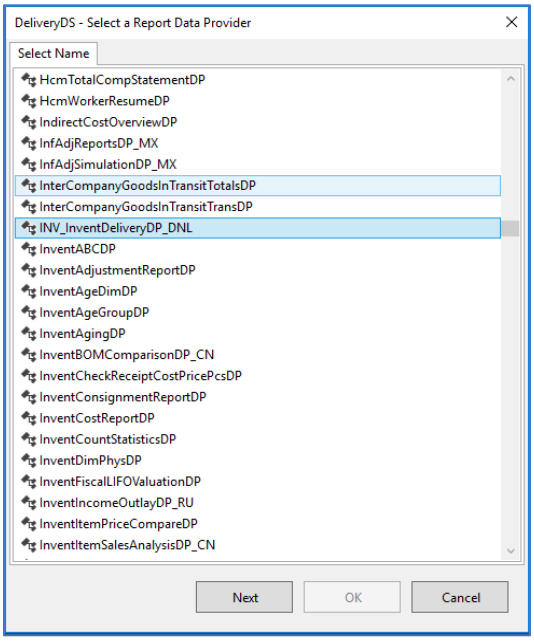


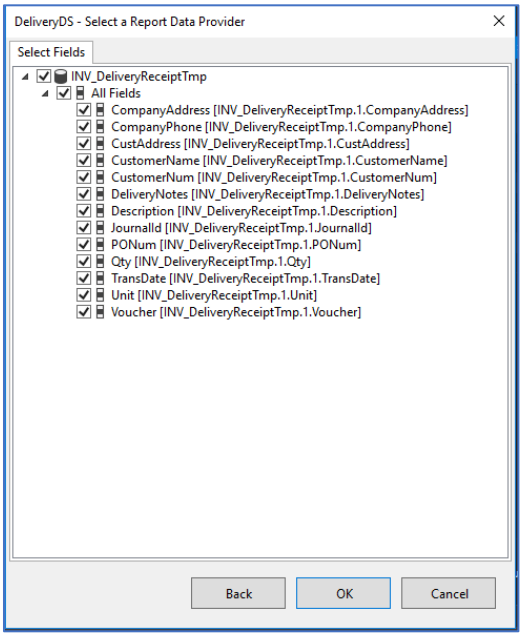
- Right click report > Click Open



Under Datasets node, click New Dataset > Rename Dataset as **DeliveryDS** then set the following properties:
Data source type – Report Data Provider

- **Query** – click the ellipses on this field, this will open a new form which list all of DP classes. Select DP Class you created > Click next > Then select fields applicable on your report > Click Ok.





- Back on your report, right click Designs node > Select Precision Design. And start designing the report based on user requirements.

- Rename report design as Report.

4. Let's create a contract class.

Identify parameters to be used in the report like date range, filters etc.

In **delivery receipt**, our only parameter will be **Journal Id** which will be used to filter records and display related records of the selected journal **only**.

SAMPLE CONTRACT CLASS:

```
[DataContractAttribute]
class INV_InventDeliveryContract_DNL
{
    //Declare parameters
    JournalId journalId;

    //parm methods
    [
        DataMemberAttribute('JournalId')
    ]
    public JournalId parmJournalId(JournalId _journalId = journalId)
    {
        journalId = _journalId;
        return journalId;
    }

}
```

5. Controller class – here we need to specify the report design to be used, initialize controller class and parameter values.

SAMPLE CONTROLLER CLASS:

- Class declaration:

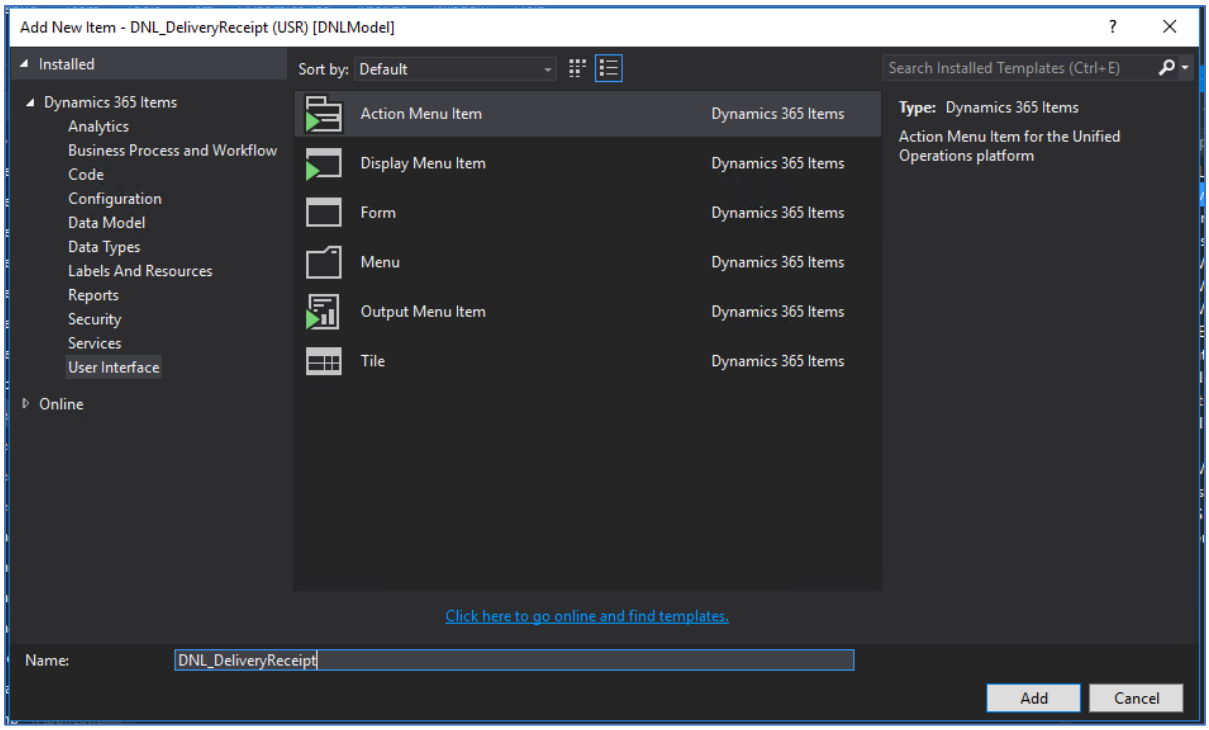
```
class INV_InventDeliveryController_DNL extends SrsReportRunController
{
    #define.reportName('DNL_DeliveryReceipt.Report')
```

- Add Main method

```
public static void main(Args _args)
{
   INV_InventDeliveryController_DNL controller = new
   INV_InventDeliveryController_DNL();
   INV_InventDeliveryContract_DNL contract = new INV_InventDeliveryContract_DNL();
   InventJournalTable  inventJournalTable;

    controller.parmReportName(#reportName);
    controller.parmArgs(_args);
    contract = controller.parmReportContract().parmRdpContract();
    controller.parmShowDialog(false);
    controller.parmReportContract().parmRdpContract(contract);
    inventJournalTable = _args.record();

    contract.parmJournalId(inventJournalTable.JournalId);

    if (controller.prompt())
    {
        controller.run();
    }
    }

   }
```

After creating table, classes and report designs next step would be creating a menu item that will enable users to generate report at the front end.

1. Right click project > Add > New item > Under Dynamics 365 Items, Click User Interface > Select Output Menu Item (since we're creating menu item for a report) > Specify menu item name on Name field > Then click Add button.

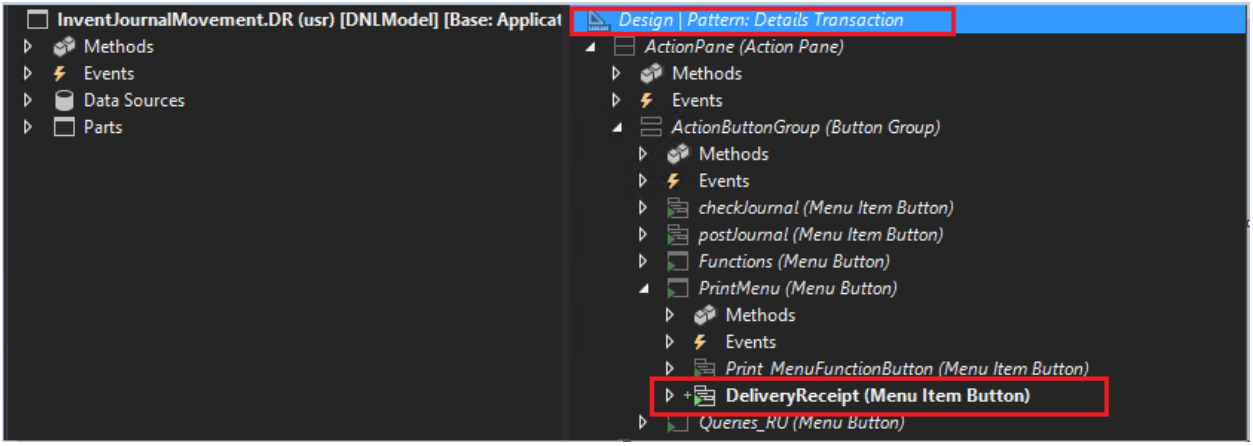


2. Open menu item. Then set the following properties:

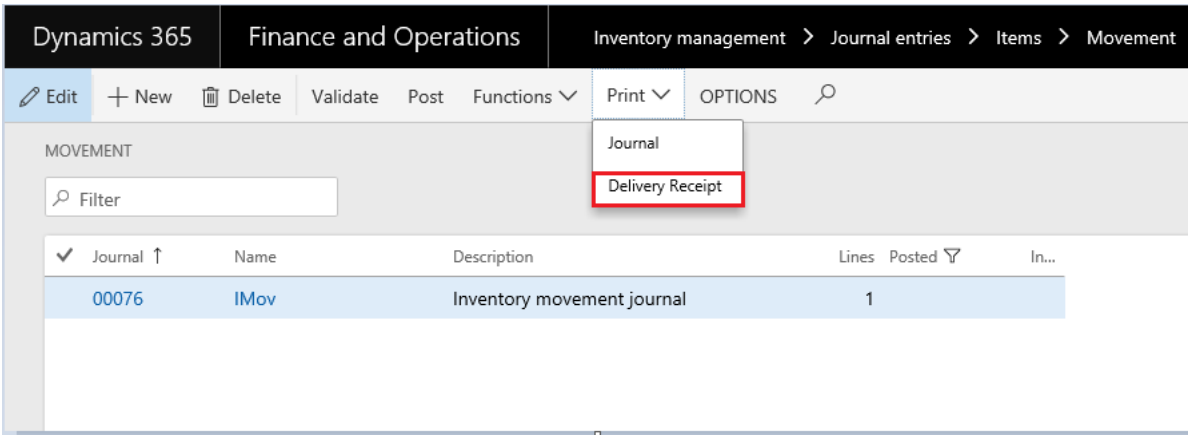
Label: Delivery Receipt
Object: INV_InventDeliveryController_DNL (Controller class)
ObjectType: Class

3. Add menu item on the form where you need to generate the report.
In this sample, we'll add the menu item on **InventJournalMovement** form.

- First, find **InventJournalMovement** form on AOT > User Interface > Forms > right click form > Click create extension
- Rename form **InventJournalMovement.DR** to easily identify that this is an extended form.
- Open the from and drag menu item under Design node > Action Pane > ActionButtonGroup > PrintMenu



4. Build and Sync your solution.
5. After build and sync, you may now generate report on the frontend.
   Under Inventory Management > Journal entries > Items > Movement



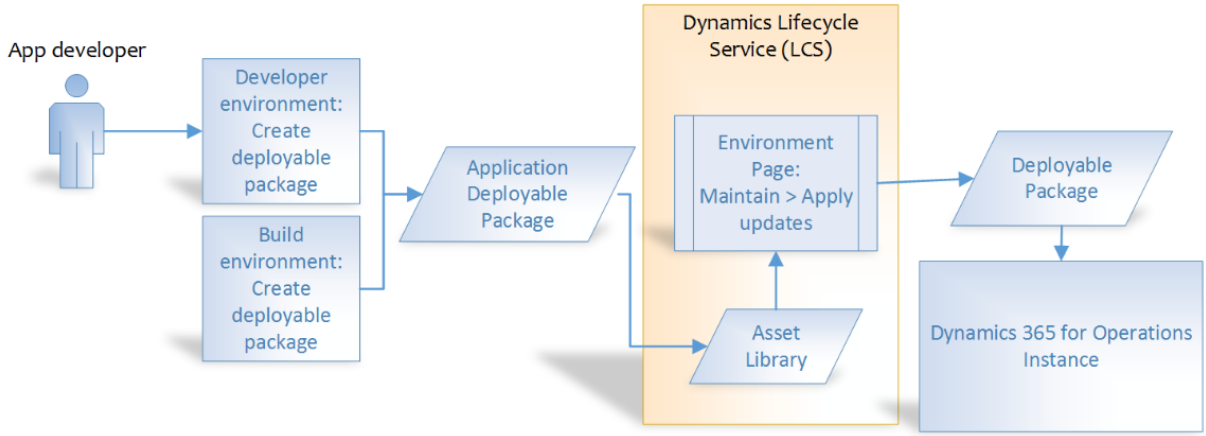
6. Dropdown Print button > Click Delivery receipt to generate report.

Module 23: Project Deployment

Lesson 1: Create deployable packages of models

- Overview of the process

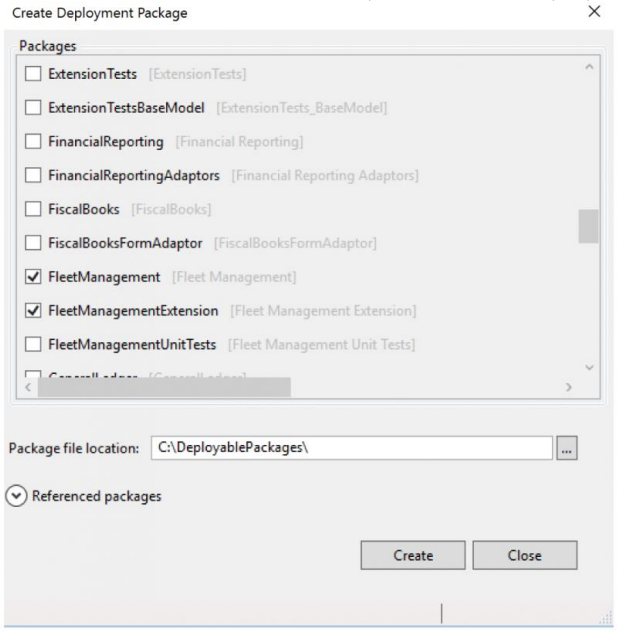
  In order to deploy your code and customizations to a runtime environment (Demo, Sandbox or Production), you must create deployable packages of your solution or implementation. Deployable packages can be created using the Visual Studio dev tools, or by the build automation process that are available on build environments. These deployable packages are referred to as Application Deployable Packages or AOT Deployable Packages. The image below is an overview of the process. Once a deployable package is created, it must be uploaded to the LCS project's asset library. An administrator can then go to the LCS environment page and apply the package to a runtime environment using the Maintain > Apply updates tool.



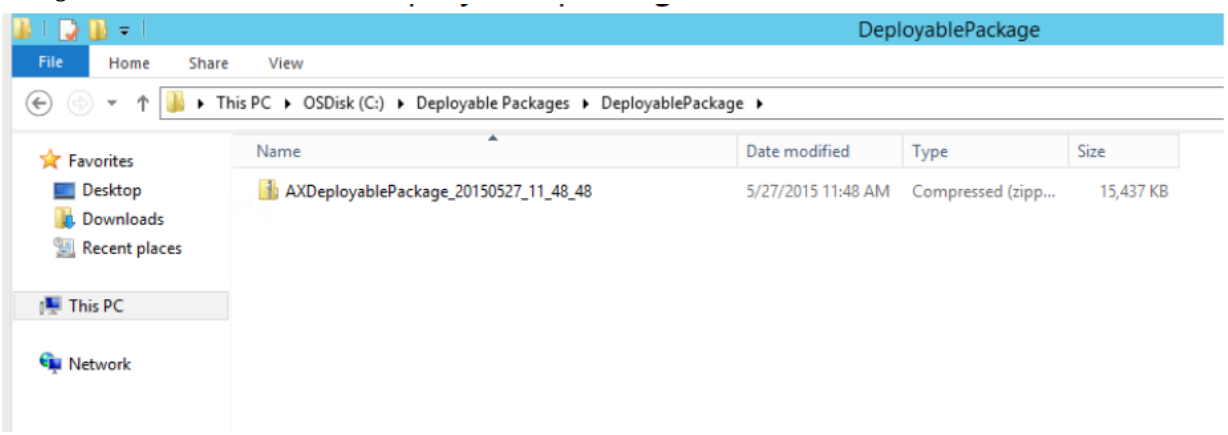
- Create a deployable package

  After you have completed the development stage, follow these steps to create a deployable package from Visual Studio.

  1. In Microsoft Visual Studio, select **Dynamics 365** > **Deploy** > **Create Deployment Package**.



  2. Select the packages that contain your models, and then select a location in which to create the deployable package.



  3. After a deployable package is created, sign in to Microsoft Dynamics Lifecycle Services (LCS), and then, in your LCS project, click the **Asset Library** tile.
  4. Upload the deployable package that you created earlier.

Resource: https://docs.microsoft.com/en-us/dynamics365/unified-operations/dev-itpro/deployment/create-apply-deployable-package

Lesson 2: Apply updates to cloud environments

- Supported environments

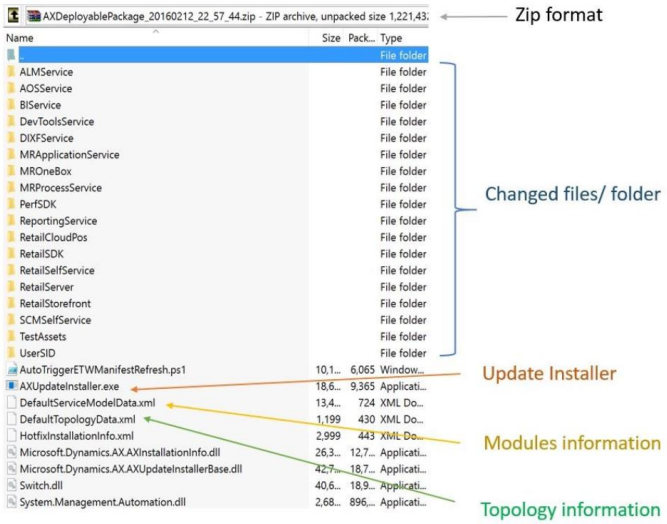
  The following topologies support package deployment that uses automated flows in LCS:

  LCS Implementation Project – All environment types are supported. Automated package application is a self-service operation in all environments except production environments. For production environments, customers must use LCS to submit a request to apply packages.

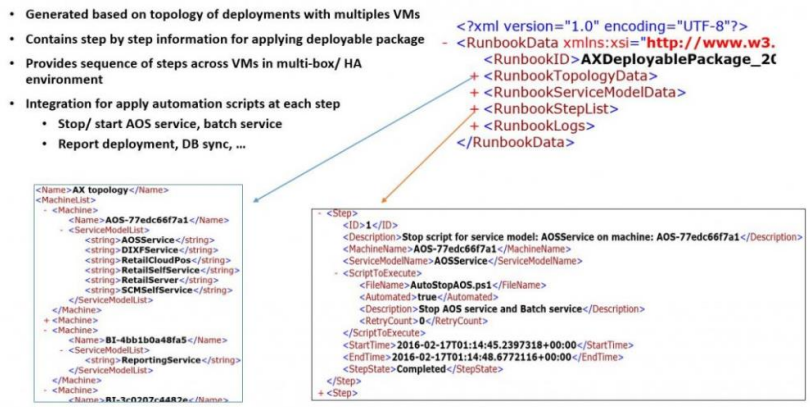
  LCS Partner and Trial Projects – All environment types are supported, except multi-box dev/test topologies.

- Key concepts
  Deployable packages, runbooks, and the AXUpdateInstaller are the tools you use to apply updates.

  Deployable package – A deployable package is a unit of deployment that can be applied in any Finance and Operations or Retail environment. A deployable package can be a binary update to the platform or other runtime components, an updated application (AOT) package, or a new application (AOT) package. Deployable packages downloaded from LCS or created in a development environment cannot be applied across product types. That is, a Finance and Operations deployable package cannot be applied in a Retail environment, and vice versa. If you have an existing customization for Finance and Operations that is compatible with Retail, and would like to apply it to a Retail environment, you will need to re-package your source code in a Retail development environment, and conversely if moving in the other direction.



  Runbook – The deployment runbook is a series of steps that are generated in order to apply the deployable package to the target environment. Some steps are automated, and some steps are manual. AXUpdateInstaller lets you run these steps one at a time and in the correct order.



AXUpdateInstaller – When you create a customization package from Microsoft Visual Studio or a Microsoft binary update, the installer executable is bundled together with the deployable package. The installer generates the runbook for the specified topology. The installer can also run steps in order, according to the runbook for a specific topology.

- Supported package types

  AOT deployable package – A deployable package that is generated from application metadata and source code. This deployable package is created in a development or build environment.

  Application and Platsform Binary update package – A deployable package that contains dynamic-link libraries (DLLs) and other binaries and metadata that the platform and application depend on. This is a package released by Microsoft. This is available from the **All binary updates** tile from LCS.

  Platform update package – A deployable package that contains dynamic-link libraries (DLLs) and other binaries and metadata that the platform depend on. This is a package released by Microsoft. This is available from the **Platform binary updates** tile from LCS.

  Retail deployable package – A combination of various Retail packages that are generated after the Retail code is combined.

  Merged package – A package that is created by combining one package of each type. For example, you can merge one binary update package and one AOT package, or one AOT package and one Retail deployable package. The packages are merged in the Asset library for the project in LCS.

- Prerequisite steps

  Make sure that the package that should be applied is valid. When a package is uploaded to the Asset library, it isn't analyzed. If you select the package, the package status appears in the right pane as **Not Validated**. A package must pass validation before it can be applied in an environment by using the following procedures. The status of the package will be updated in the Asset library to indicate whether the package is valid. We require validation to help guarantee that production environments aren't affected by packages that don't meet the guidelines.

  There are three types of validations:
  1. Basic package format validations
  2. Platform version checks
  3. Types of packages

Make sure that the package is applied in a sandbox environment before it's applied in the production environment. To help guarantee that the production environment is always in a good state, we want to make sure that the package is tested in a sandbox environment before it's applied in the production environment. Therefore, before you request that the package be applied in your production environment, make sure that it has been applied in your sandbox environment by using the automated flows.

If you want to apply multiple packages, create a merged package that can be applied first in a sandbox environment and then in the production environment. Application of a single package in an average environment requires about 5 hours of downtime. To avoid additional hours of downtime when you must apply multiple packages, you can create a single combined package that contains one package of each type. If you select a binary package and an application deployable package in the Asset library, a **Merge** button becomes available on the toolbar. By clicking this button, you can merge the two packages into a single package and therefore reduce the total downtime by half.

Make sure that the Application binary update package is applied to your dev/build environment before it is applied to your sandbox and production environment - If the application binary package is applied directly to your Tier 2+ sandbox environment but is not applied on your dev/build environment, the next time you move an AOT package from dev/build box (which does not have the same application binaries as your sandbox environment) to sandbox, some of the application binaries will be overwritten with what is in your dev/build environment. This could result in a regression of the version of your sandbox environment.

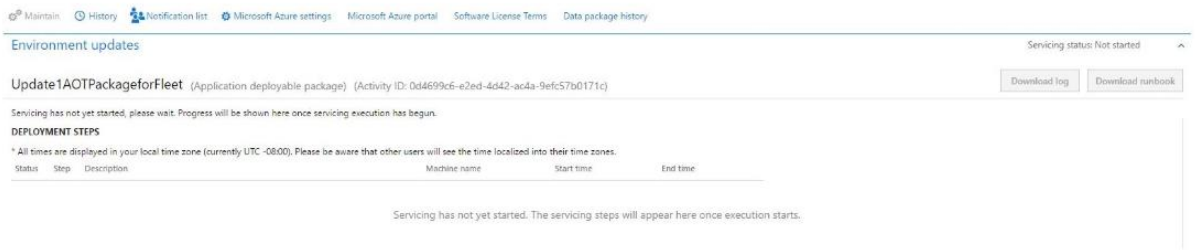
- Apply a package to a non-production environment by using LCS

  Before you begin, verify that the deployable package has been uploaded to the Asset library in LCS.

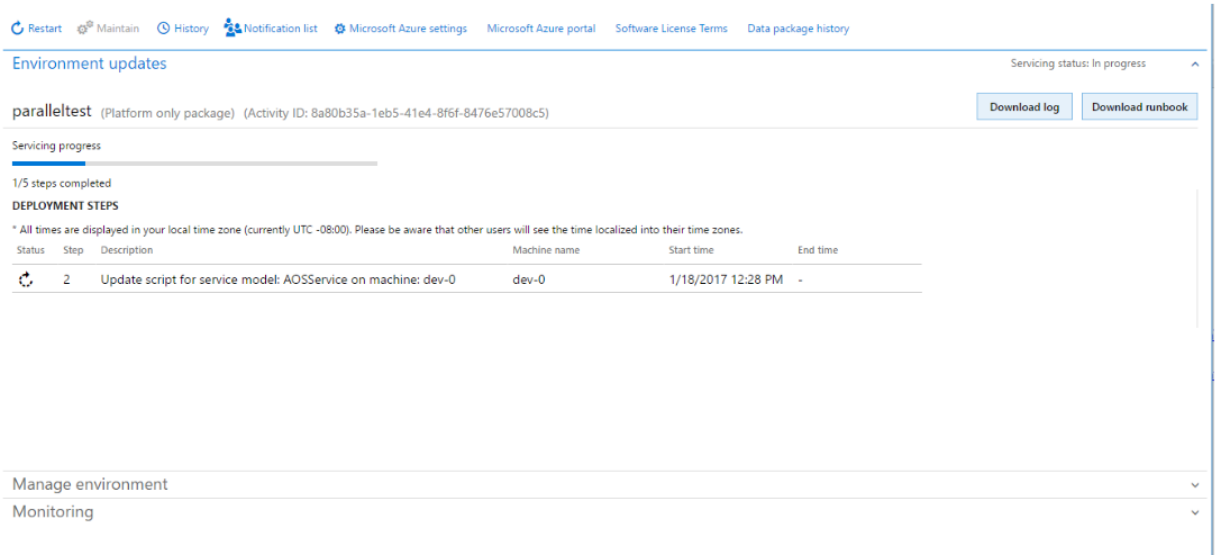
  1. For a binary update, upload the package directly to the Asset library. For information about how to download an update from LCS, see Download updates from Lifecycle Services.

     For an application (AOT) deployable package that results from an X++ hotfix, or from application customizations and extensions, create the deployable package in your development or build environment, and then upload it to the Asset library.

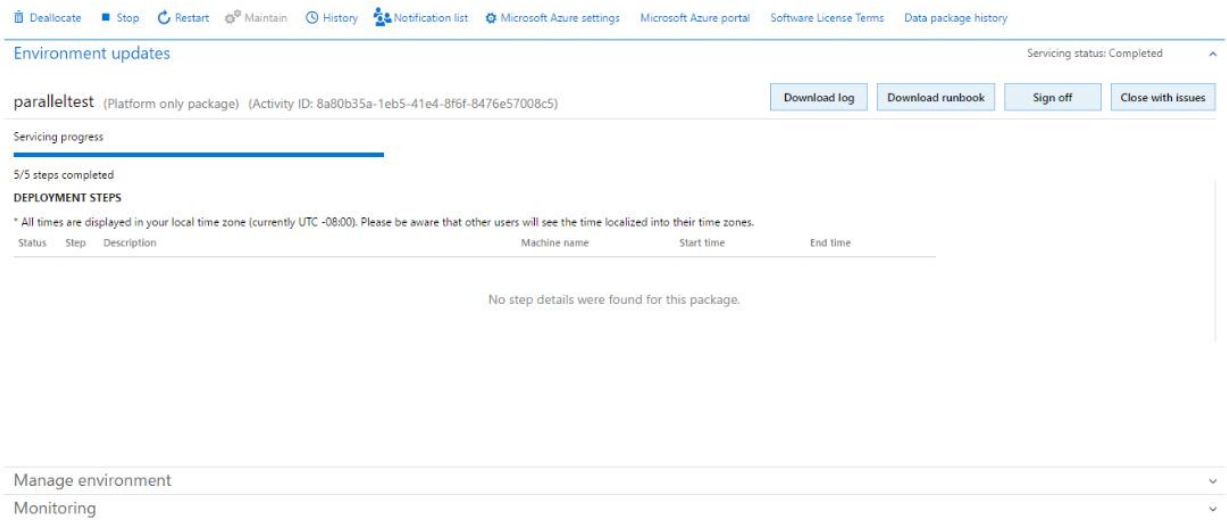
  2. Open the **Environment details** view for the environment where you want to apply the package.
  3. Click **Maintain** > **Apply updates** to apply an update.
  4. Select the package to apply. Use the filter at the top to find your package.
  5. Click **Apply**. Notice that the status in the upper-right corner of the **Environment details** view changes to **Queued**, and that an **Environment updates** section now shows the progress of the package.



6. Refresh the page to see the progress of the package application. Notice that the servicing status is **In Progress**, and that the environment status is **Servicing**.



7. Continue to refresh the page to see the status updates for the package application request. When the package has been applied, the environment status changes to **Deployed**, and the servicing status changes to **Completed**.

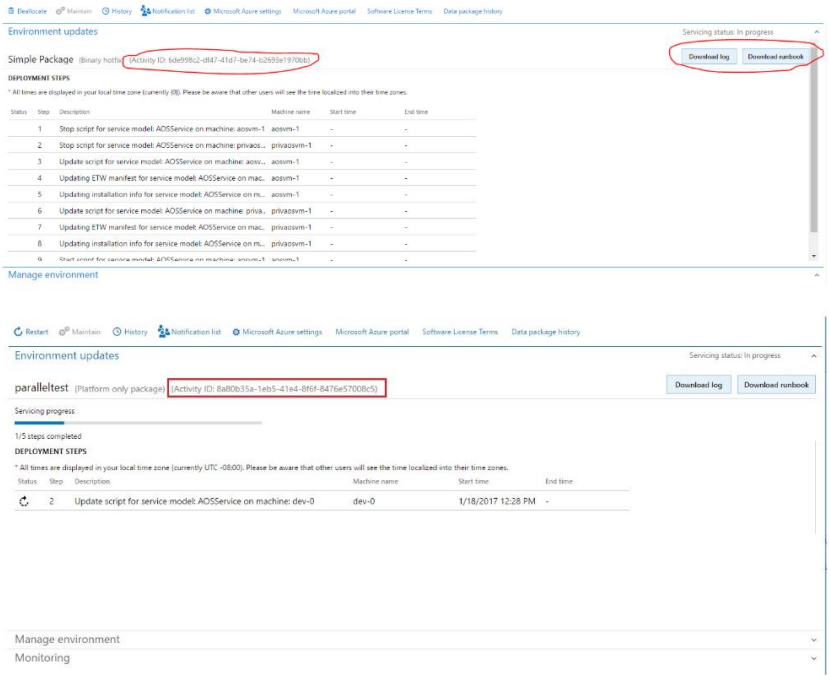


8. To sign off on package application, click **Sign off** if there are no issues. If issues occurred when you applied the package, click **Sign off with issues**.

- Troubleshooting
  General troubleshooting/diagnostics

  If package application isn't successful, you can download either the logs or the runbook to see the detailed logs. You can also use RDP to connect to an environment so that you can fix issues. If you must report the

issue to Microsoft, be sure to include the activity ID that is reported in the **Environment updates** section.



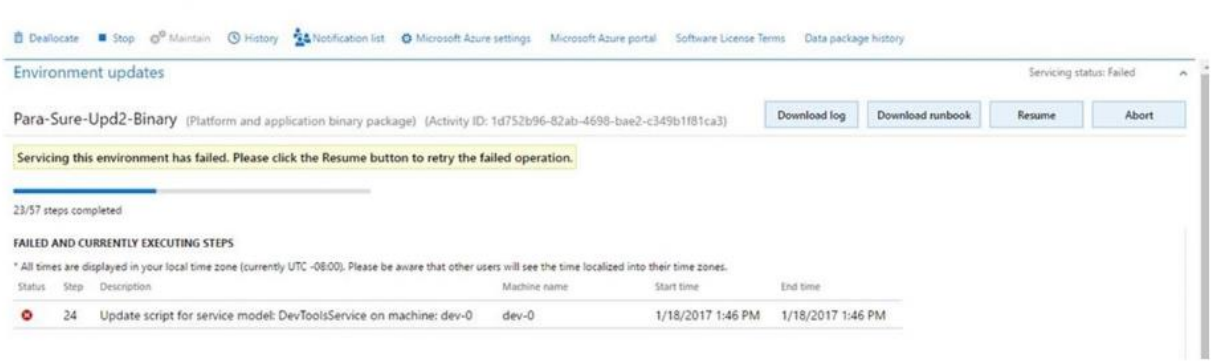
Using the logs

1. Download the logs.
2. Unzip the log files.
3. Select the role that a step failed for, such as **AOS** or **BI**.
4. Select the VM where the step failed. This information appears in the **Machine name** column in the **Environment updates** section.
5. In the logs for the VM, select the folder that corresponds to the step where the issue occurred. The folder name identifies the step that each folder corresponds to. For example, if the issue occurred in the executing of a step, select the **ExecuteRunbook** folder.
   For example, if the folder name is ExecuteRunbook-b0c5c413-dae3-4a7a-a0c4-d558614f7e98-1_I0_R0, the step number is highlighted and is the number after the globally unique identifier (GUID).

Package failure

If package application isn't successful, you have two options:

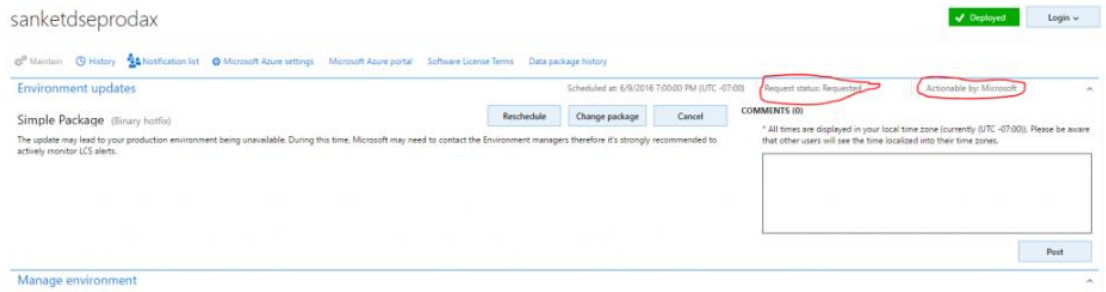
1. Click **Resume** to retry the operation that failed.



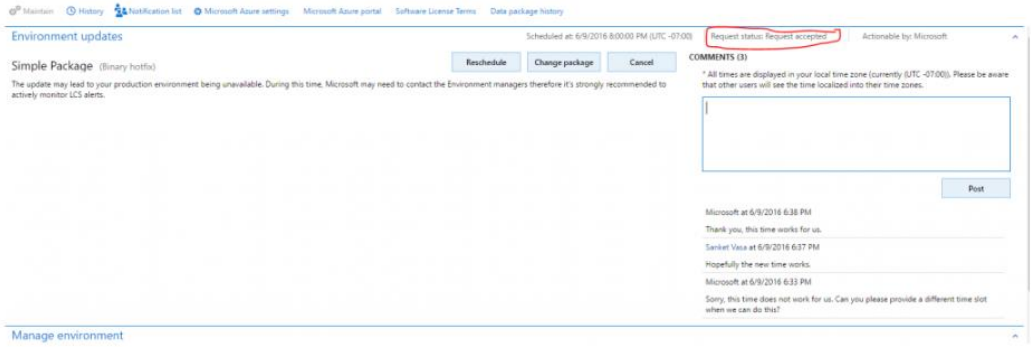
2. Click **Abort** to stop package application.

- Apply a package to a production environment by using LCS
  In a production environment, unlike in a sandbox environment or other types of environments, package application through LCS isn't self-serve. Customers and partners must submit a request to Microsoft to apply a package when the customer is ready for the downtime.

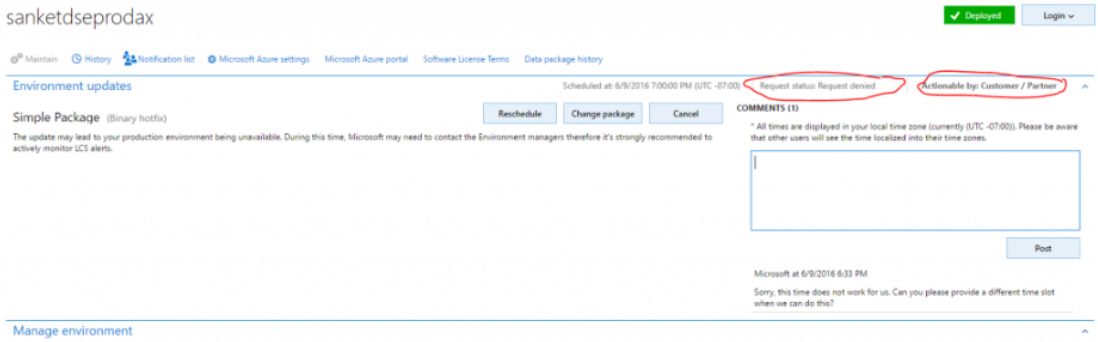
  1. Download an update from LCS. For information about how to download an update from LCS, see Download updates from Lifecycle Services.
     - For a binary update, upload the update deployable package directly to the Asset library.
     - For an application/X++ update, apply the package in a development environment. After you resolve any conflicts, generate a deployable package from Visual Studio, and upload the package to the Asset library. For information about how to upload to the Asset library and create a deployable package, see Create deployable packages of models.
  2. On the **Asset library** page in LCS, on the tab that corresponds to the asset type (**Software deployable package**), select a package, and then click **Release candidate**.
  3. Apply the package in a sandbox environment by using the instructions earlier in this topic.
  4. After the package is successfully applied and signed off in the sandbox environment, open the Asset Library, and mark the package as **Release Candidate**.
  5. Open the **Environment details** view for the production environment where you want to apply the package.
  6. Click **Maintain** > **Apply updates** to apply the package.
  7. Select the type of package to apply.
  8. Select the package to apply in your production environment, and then click **Schedule** to submit a request to apply it.
  9. Specify the date and time to schedule package application for, click **Submit**, and then click **OK** to confirm. Note that your environments will be down and unavailable to perform business while the package is being applied.
  10. Refresh the page. Two fields on the page indicate the status of the request.
      1. **Request status** – This field indicates the status of the request that you submitted to Microsoft.
      2. **Actionable by** – This field indicates who must take action.



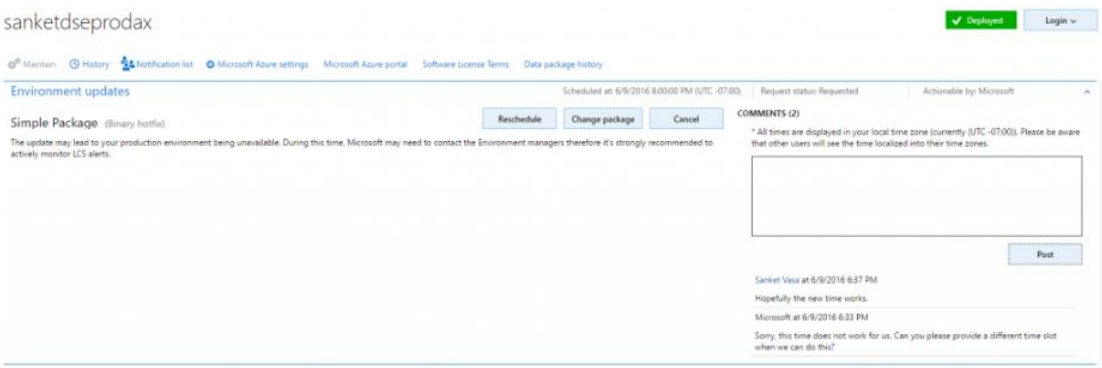
11. Microsoft either accepts or denies the request.
    - If the request is accepted, Microsoft begins to update the environment.



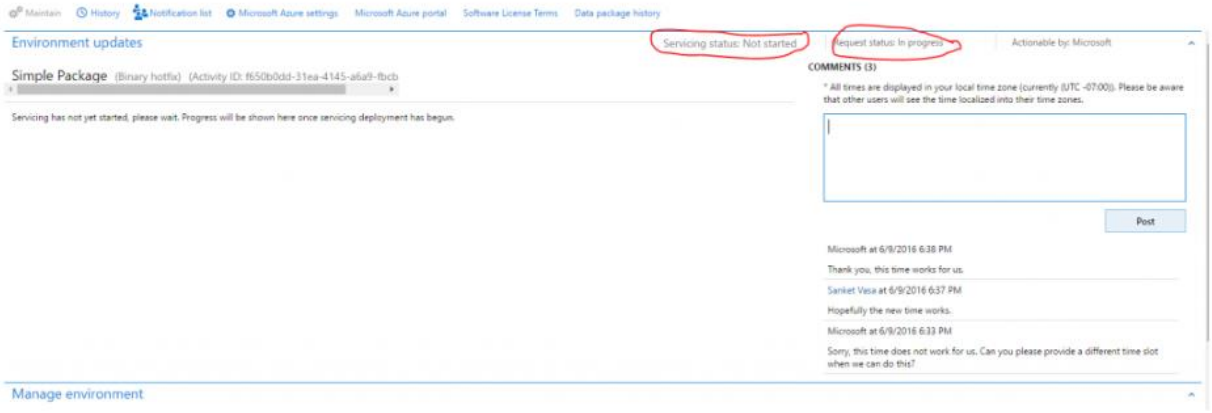
    - If the request is denied, Microsoft informs the customer about the reason for denial and the action that the customer must take. The customer can then reschedule the request, change the package, or cancel the request.



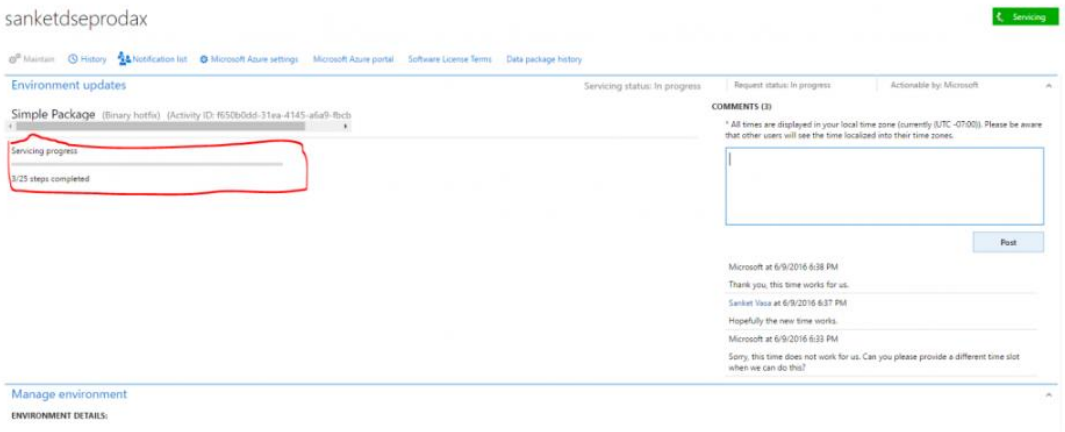
At any time, the customer can use the **Comments** field to post comments to the request.



12. After the environment is serviced, you can monitor the status. The **Servicing status** field indicates the status of package application.

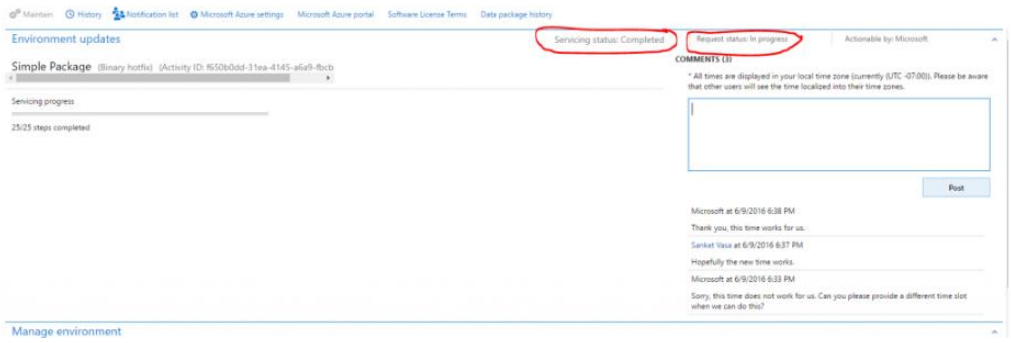


Additionally, a progress indicator shows the number of steps that have been run, out of the total number of steps that are available.



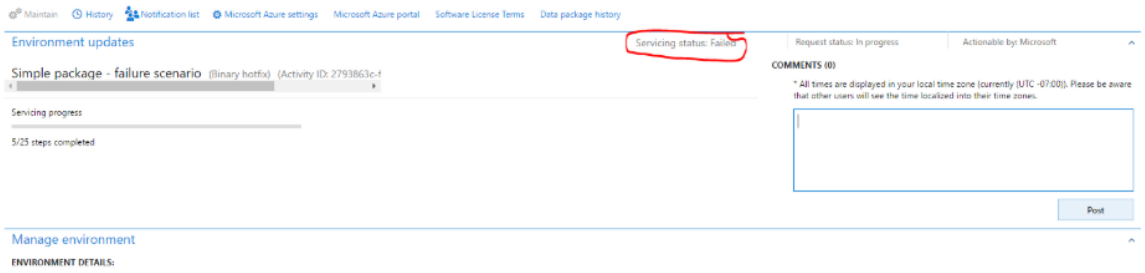
- Successful package application

  - After the deployment is successfully completed, the **Servicing status** field is set to **Completed**, but the **Request status** field is still set to **In progress** because the request hasn't yet been closed.

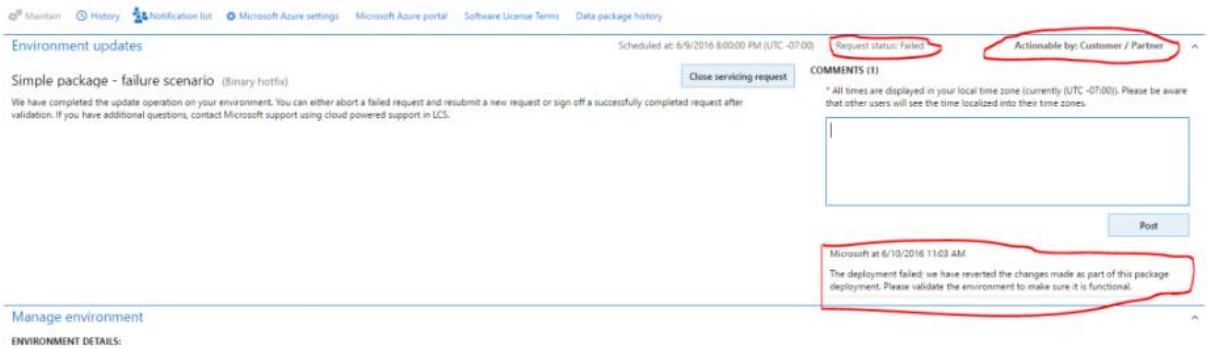


- After Microsoft has finished applying the request, you must close the request by clicking **Close servicing request**.
- When you close a successful request, in the **Edit work item details** dialog box, set the **Service request status** field to **Succeeded**, and then click **Submit**.

- Unsuccessful package application
  - If package application isn't successfully completed, Microsoft will investigate the issue. The **Servicing status** field will indicate that package application has failed.



  - When deployment fails, Microsoft can abort the package, revert the environment to a good state, and send the request back to the customer, so that the customer can validate the environment and close the request. If there is an issue in the package, the customer must submit a new request that includes the new package.



  - When you close a failed request, in the **Edit work item details** dialog box, set the **Service request status** field to **Aborted**.

Resource: https://docs.microsoft.com/en-us/dynamics365/unified-operations/dev-itpro/deployment/apply-deployable-package-system

Lesson 3: Remove a deployable package

Occasionally, you might have to uninstall a deployable package. For example, you might be reorganizing your source code. Alternatively, you no longer require an independent software vendor (ISV) product and haven't renewed the license. Therefore, you must remove the package.

- Remove a model
  A model is a design-time concept that is part of a package. When a model isn't the only model in a module, you can just remove it from the source code. No other steps are required, because when you deploy the updated module, the old module is overwritten. All overlayer models fall into this category.

- Prerequisites
  - If any models reference the module that will be removed, the references must be removed from them. For information about how to find the references that must be removed, see Viewing model dependencies.
  - Build and deploy any modules that references were removed from.
  - All references to and from the modules must be removed before you begin to uninstall the module. To remove all a module's references, add a single class to the model. This class should contain no code. It should contain only a reference to the application platform.

- Uninstall a package
  1. Create a file that is named **ModuleToRemove.txt**.
  2. In the file, put the name of each module that you want to remove on a separate line. Make sure that you've completed the prerequisites for each module that you're removing.
  3. Create a valid deployable package, and put the ModuleToRemove.txt file in the **package\AOSService\Scripts** folder.
  4. Install the deployable package. For more information about how to install deployable packages, see Apply updates to a cloud deployment.
  5. Verify that the package was uninstalled before you complete this procedure in a production environment.

Resource: https://docs.microsoft.com/en-us/dynamics365/unified-operations/dev-itpro/deployment/uninstall-deployable-package

Lesson 4: Install Deployable package through runbook

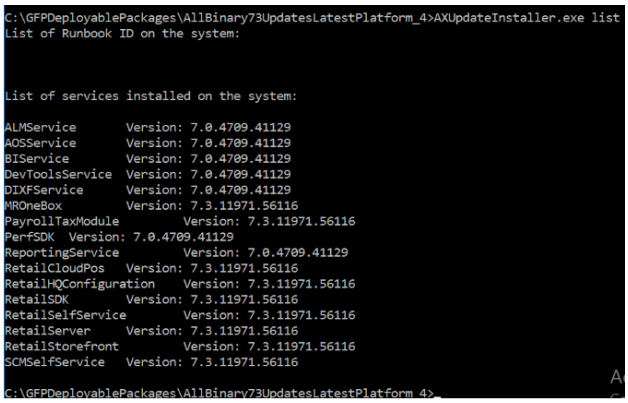
1. Create a folder on a non-user directory(C:)
2. Extract zip file to the created folder
3. Collect topology configuration data. In the folder where you extracted the deployable package, find and open the file that is named **DefaultTopologyData.xml**. You must specify the **VM name** and **the installed components** in this file.
   a) To specify the VM name, follow these steps:
      - In File Explorer, right-click **This PC**, and then select **Properties**.
      - In the system properties, find and make a note of the computer name (for example, **AOS-950ed2c3e7b**).
      - In the DefaultTopologyData.xml file, replace the machine name with the computer name that you found in the previous step.

   b) To specify the installed components, follow these steps:

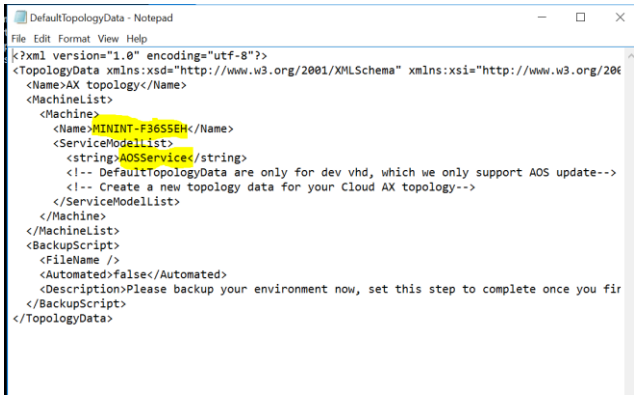
      - Open a Command Prompt window as an administrator.

- Go to the extracted folder and run the following command to see a list of all the components that are installed on the computer.

  AXUpdateInstaller.exe list



c) the DefaultTopologyData.xml file should resemble the following illustration.



d) Repeat steps a and b for every other VM that is listed on the **Environment** page

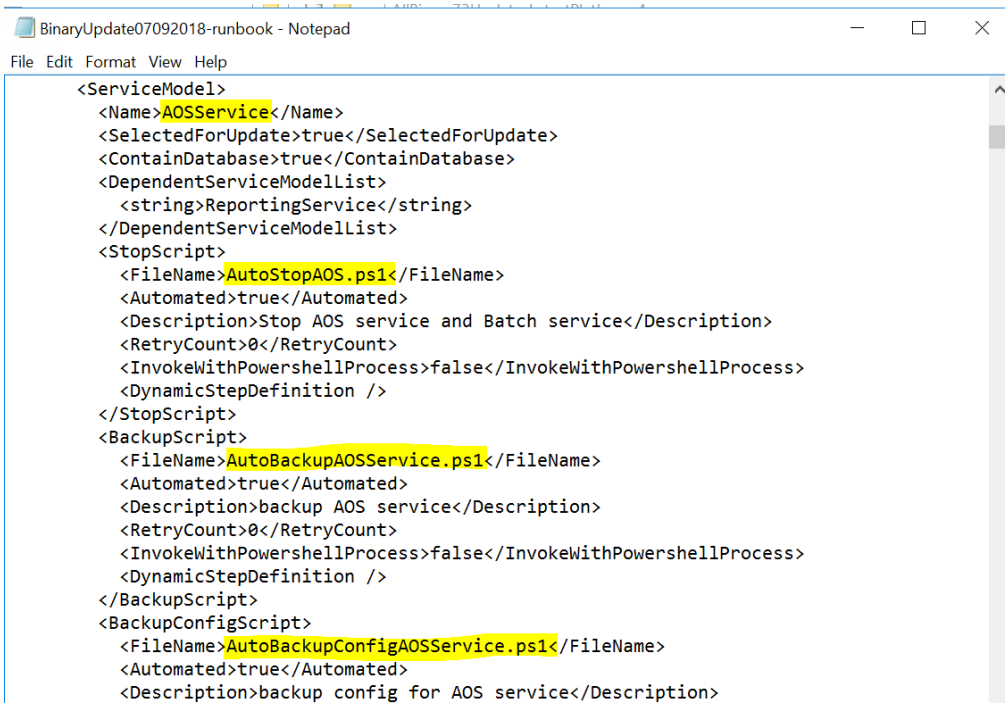
4. Generate runbook from the topology.
   a) Run the following command to generate the runbook.

   AXUpdateInstaller.exe generate -runbookid=[**runbookID**] -topologyfile=[topologyFile] -servicemodelfile=[**serviceModelFile**] -runbookfile=[**runbookFile**]

   - **[runbookID]**– A parameter that is specified by the developer who applies the deployable package. Developer may use any file name for runbook id.
   - **[topologyFile]**– The path and name of the **DefaultTopologyData.xml** file.
   - **[serviceModelFile]**– The path of the **DefaultServiceModelData.xml** file.
   - **[runbookFile]**– The name of the runbook file to generate (for example, **AOSRunbook.xml**).

```
C:\GFPDeployablePackages\AXDeployablePackage_20180706_11_26_24>AXUpdateInstaller.exe generate -runbookid="GFPCustomizations07092018-runbook"
-runbookfile="GFPCustomizations07092018-runbook.xml" -topologyfile="DefaultTopologyData.xml" -servicemodelfile="DefaultServiceModelData.xml"
```

The runbook provides the sequence of steps that must be run to update the environment. The following illustration shows an example of a runbook file. Each step in a runbook is associated with an ID, a machine name, and step execution details.



5. Install deployable package

   a) Import the runbook by running the following command:

      AXUpdateInstaller.exe import -runbookfile=[runbookFile]

```
C:\GFPDeployablePackages\AllBinary73UpdatesLatestPlatform_4>AXUpdateInstaller.exe import -runbookfile="BinaryUpdate07092
018-runbook.xml"
```

   b) Verify the runbook

      AXUpdateInstaller.exe list

```
C:\GFPDeployablePackages\AllBinary73UpdatesLatestPlatform_4>AXUpdateInstaller.exe list
List of Runbook ID on the system:

BinaryUpdate07092018-runbook


List of services installed on the system:

ALMService       Version: 7.0.4709.41129
AOSService       Version: 7.0.4709.41129
BIService        Version: 7.0.4709.41129
DevToolsService  Version: 7.0.4709.41129
DIXFService      Version: 7.0.4709.41129
MROneBox         Version: 7.3.11971.56116
PayrollTaxModule         Version: 7.3.11971.56116
PerfSDK  Version: 7.0.4709.41129
ReportingService         Version: 7.0.4709.41129
RetailCloudPos   Version: 7.3.11971.56116
RetailHQConfiguration    Version: 7.3.11971.56116
RetailSDK        Version: 7.3.11971.56116
RetailSelfService        Version: 7.3.11971.56116
RetailServer     Version: 7.3.11971.56116
RetailStorefront         Version: 7.3.11971.56116
SCMSelfService   Version: 7.3.11971.56116
```

c) Execute runbook

AXUpdateInstaller.exe execute -runbookid=[runbookID]

```
C:\GFPDeployablePackages\AllBinary73UpdatesLatestPlatform_4>AXUpdateInstaller.exe execute -runbookid="BinaryUpdate070920
18-runbook"
Start executing runbook : BinaryUpdate07092018-runbook
```

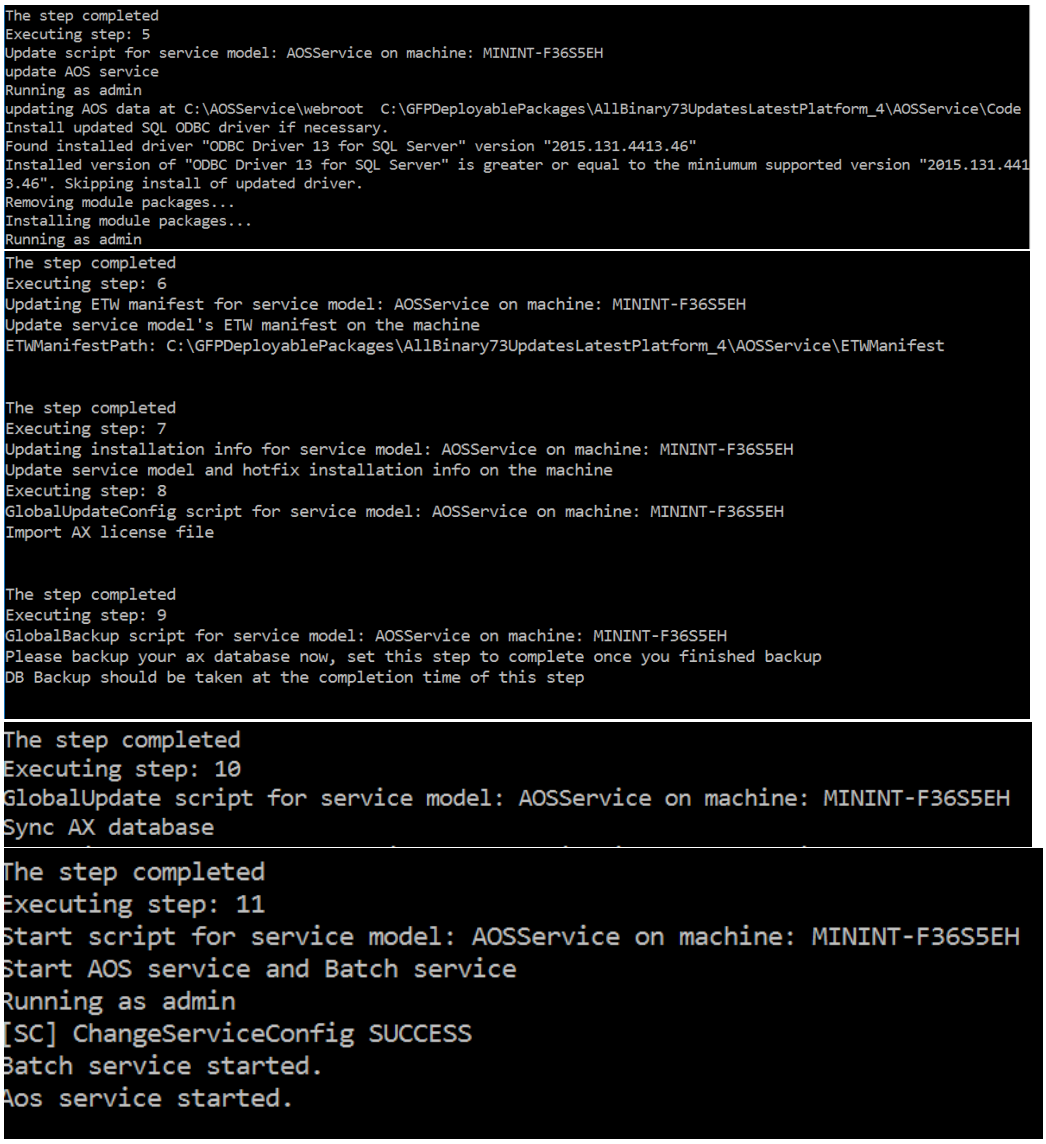
Make sure all steps are completed and error free

```
C:\GFPDeployablePackages\AllBinary73UpdatesLatestPlatform_4>AXUpdateInstaller.exe execute -runbookid="BinaryUpdate070920
18-runbook"
Start executing runbook : BinaryUpdate07092018-runbook
Executing step: 1
Stop script for service model: AOSService on machine: MININT-F36S5EH
Stop AOS service and Batch service
Running as admin
DynamicsAxBatch stopped.
Aos service stopped.
```

```
The step completed
Executing step: 2
BackupConfig script for service model: AOSService on machine: MININT-F36S5EH
backup config for AOS service


    Directory: C:\GFPDeployablePackages\AllBinary73UpdatesLatestPlatform_4\RunbookWorkingFolder\BinaryUpdate07092018-run

    book\MININT-F36S5EH\AOSService\2\Backup


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         7/9/2018  12:11 AM                webroot
backup AOS config data at C:\AOSService\webroot
```

```
Executing step: 3
UpdateConfig script for service model: AOSService on machine: MININT-F36S5EH
upgrade config for AOS service
Creating the log file 'C:\GFPDeployablePackages\AllBinary73UpdatesLatestPlatform_4\AOSService\scripts\UpdateConfigFiles.
log'.
```

```
The step completed
Executing step: 4
Backup script for service model: AOSService on machine: MININT-F36S5EH
backup AOS service
```



6. For manual steps, follow the instructions, and then run the following command to mark the step as completed in the runbook.

   AXUpdateInstaller.exe execute -runbookID=[runbookID] -setstepcomplete=[stepID]

7. Export and back up runbook file by running the following

   AXUpdateInstaller.exe export -runbookid=[runbookID] -runbookfile=[runbookFile]

8. After all the steps in the runbook are completed and you've exported the runbook, save the file outside the computer for future reference. For example, you might have to use the runbook file in these situations:
   a) You must analyze the downtime requirements for production, and so on.
   b) You must send the file to Microsoft because a deployable package can't be installed.

IF THERE ARE ERRORS:

9. If errors occur during any step, debug the script or the instructions in the step, and update accordingly.
10. If any step in the runbook fails, you can rerun it by running the following command.

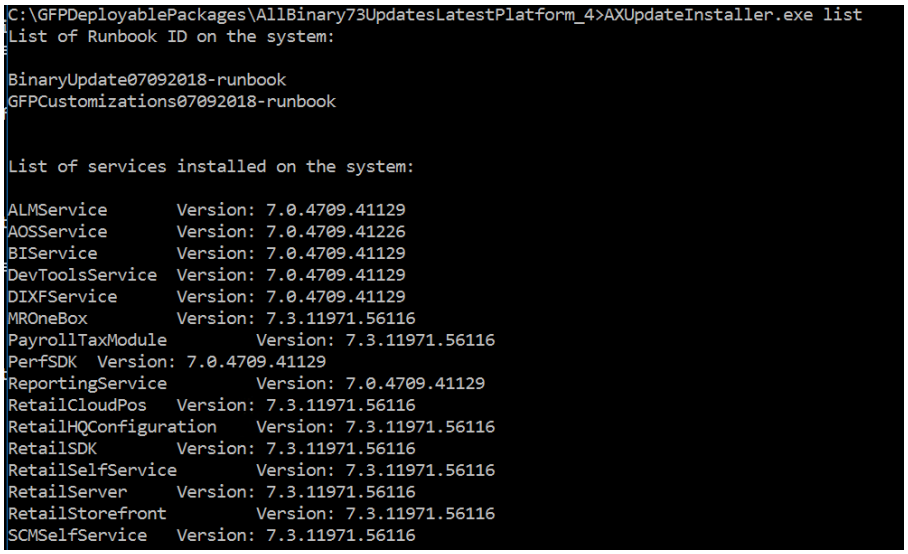
    AXUpdateInstaller.exe execute -runbookid=[runbookID] -rerunstep=[stepID]

11. If the steps has been completed, run the command to set the step to completed

    AXUpdateInstaller.exe execute -runbookID=[runbookID] -setstepcomplete=[stepID]

12. Verify installation by running the following:

    AXUpdateInstaller.exe list



13. Open D365 and check deployed packages.

Resource: https://docs.microsoft.com/en-us/dynamics365/unified-operations/dev-itpro/deployment/install-deployable-package