Kinetic Power Tools User Guide

Version 2026.100

Disclaimer

This document is for informational purposes only and is subject to change without notice. This document and its contents, including the viewpoints, dates and functional content expressed herein are believed to be accurate as of its date of publication. However, Epicor Software Corporation makes no guarantee, representations or warranties with regard to the enclosed information and specifically disclaims any applicable implied warranties, such as fitness for a particular purpose, merchantability, satisfactory quality or reasonable skill and care. As each user of Epicor software is likely to be unique in their requirements in the use of such software and their business processes, users of this document are always advised to discuss the content of this document with their Epicor account manager. All information contained herein is subject to change without notice and changes to this document since printing and other important information about the software product are made or published in release notes, and you are urged to obtain the current release notes for the software product. We welcome user comments and reserve the right to revise this publication and/or make improvements or changes to the products or programs described in this publication at any time, without notice.

The usage of any Epicor software shall be pursuant to an Epicor end user license agreement and the performance of any consulting services by Epicor personnel shall be pursuant to Epicor's standard services terms and conditions. Usage of the solution(s) described in this document with other Epicor software or third party products may require the purchase of licenses for such other products. Where any software is expressed to be compliant with local laws or requirements in this document, such compliance is not a warranty and is based solely on Epicor's current understanding of such laws and requirements. All laws and requirements are subject to varying interpretations as well as to change and accordingly Epicor cannot guarantee that the software will be compliant and up to date with such changes. All statements of platform and product compatibility in this document shall be considered individually in relation to the products referred to in the relevant statement, i.e., where any Epicor software is stated to be compatible with one product and also stated to be compatible with another product, it should not be interpreted that such Epicor software is compatible with both of the products running at the same time on the same platform or environment. Additionally platform or product compatibility may require the application of Epicor or third-party updates, patches and/or service packs and Epicor has no responsibility for compatibility issues which may be caused by updates, patches and/or service packs released by third parties after the date of publication of this document.

Table of Contents

Kinetic Power Tools

Kinetic Power Tools are a collection of system management tools that use the Classic interface (UX). You install these tools on your local system and connect them to the server for your Kinetic environment. You then launch these tools by going to your desktop and double-clicking the Kinetic Power Tools icon. The tools display on a menu; select the tool you need to run.

Be aware that while you can access these tools from the Power Tools menu, before you can launch them your system administrator may need to grant more permissions on your user account or activate additional Kinetic licenses. Review the **Using Power Tools** section for more information.

The following tools are available on the Kinetic Power Tools menu:

- **Kinetic Software Developer Kit (SDK)** - You create User-Defined (UD) services for custom Kinetic applications with this toolset. You create the UD service and its tables through the UD Service Designer. You then use Application Studio to create the interface (UX) for the custom UD service.
- **Epicor Functions** - Create Epicor Functions that have custom, reusable logic that you then apply to services to change how they run. Run this logic both externally and internally. Do this by first creating a library and then creating functions within this library where you define your own methods.
- **Advanced Print Routing (APR)** - Use the APR toolset to create routing rules. A routing rule determines how an SSRS report generates, prints, and distributes. Routing rules help you streamline reporting for specific business needs. They can be simple rules that define an alternate report style or complex rules that divide the report run into multiple rendering workflows.
- **Data Management Tool (DMT)** - Use this utility to import data into your current Kinetic database. You typically use this tool either when you are moving data from an older Kinetic/Epicor ERP version to the current database or when you are importing legacy data from another system into your Kinetic database.

While these tools currently use the Classic UX, they may move to the Kinetic UX in an upcoming major release.

Installing Kinetic Power Tools

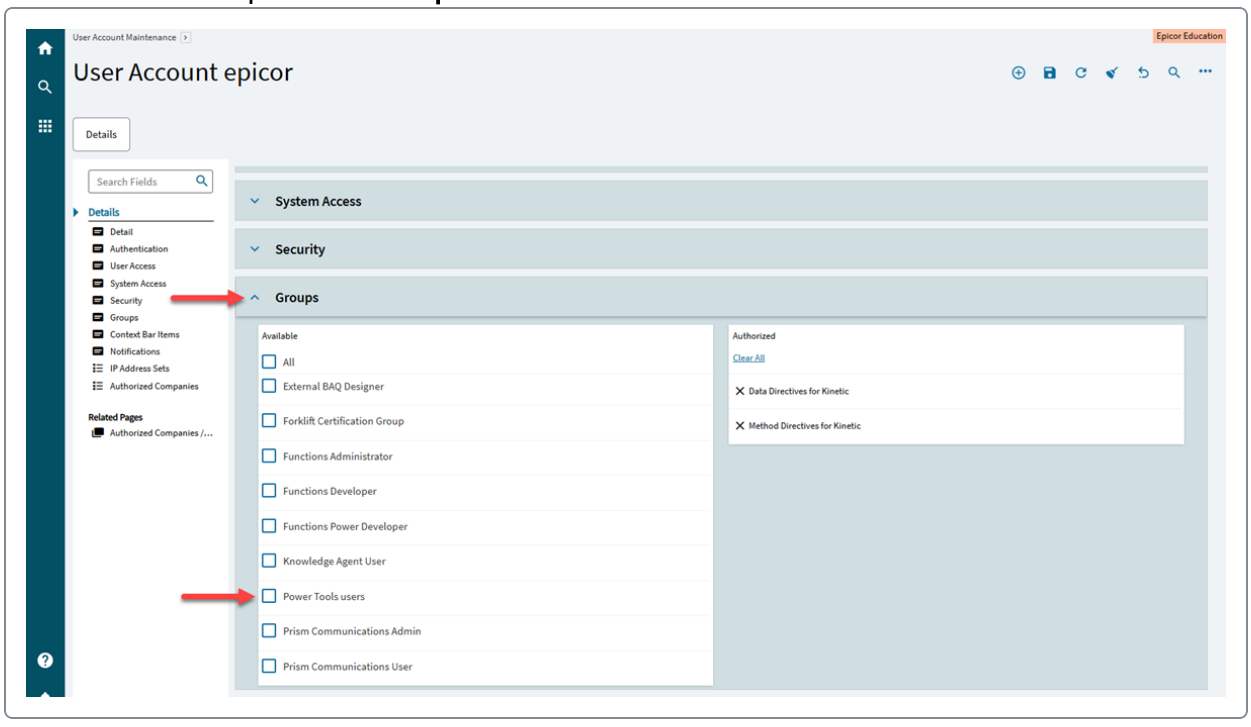
You install Kinetic Power Tools by first downloading the installer to your local machine and then running the installer. Your user account must have permission to download the installer. Once your user account has permission, log in and download the installer from the Kinetic browser client. You can then run the Power Tools installer on your local machine.

Downloading the Power Tools Installer

Begin by granting permission to your user account. You can then download the installer from the Kinetic browser.

1. Log into the Kinetic browser client.
2. Go to the left toolbar, select **Menu**, and enter **user a**. Launch **User Account Security Maintenance**.

   The **Landing Page** displays.
3. Select your account.
4. Scroll down and expand the **Groups** card.

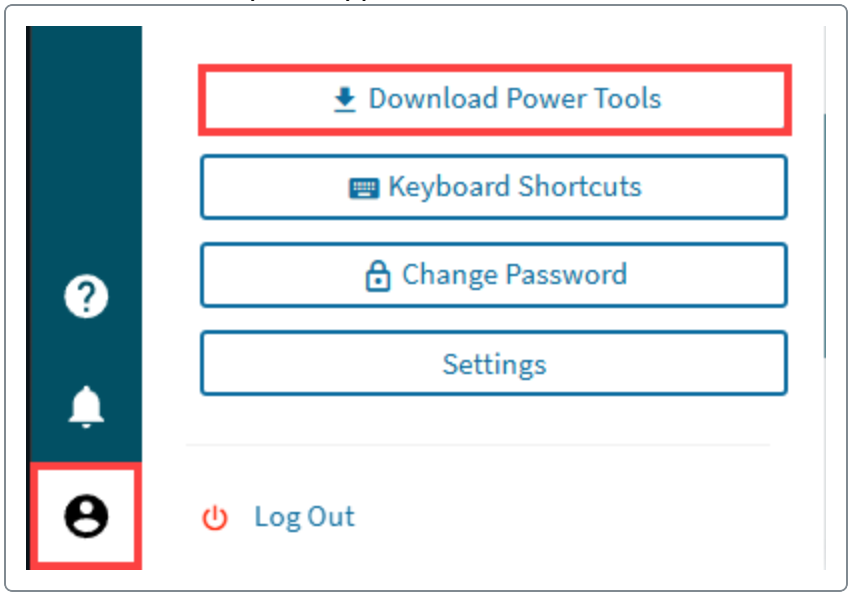


5. Go to the **Available** list and select the **Power Tools users** group.

The Power Tools users group now displays in the **Authorized** list.

6. **Save** the user account.
7. Log out of Kinetic.
8. Now log back into Kinetic with your user account.
9. Go to the left toolbar and select the **User** icon.

   The **User Account** panel appears.



10. Select the **Download Power Tools** button.

The **kineticpowertools.exe** file downloads on your local machine.

Running the Power Tools Installer

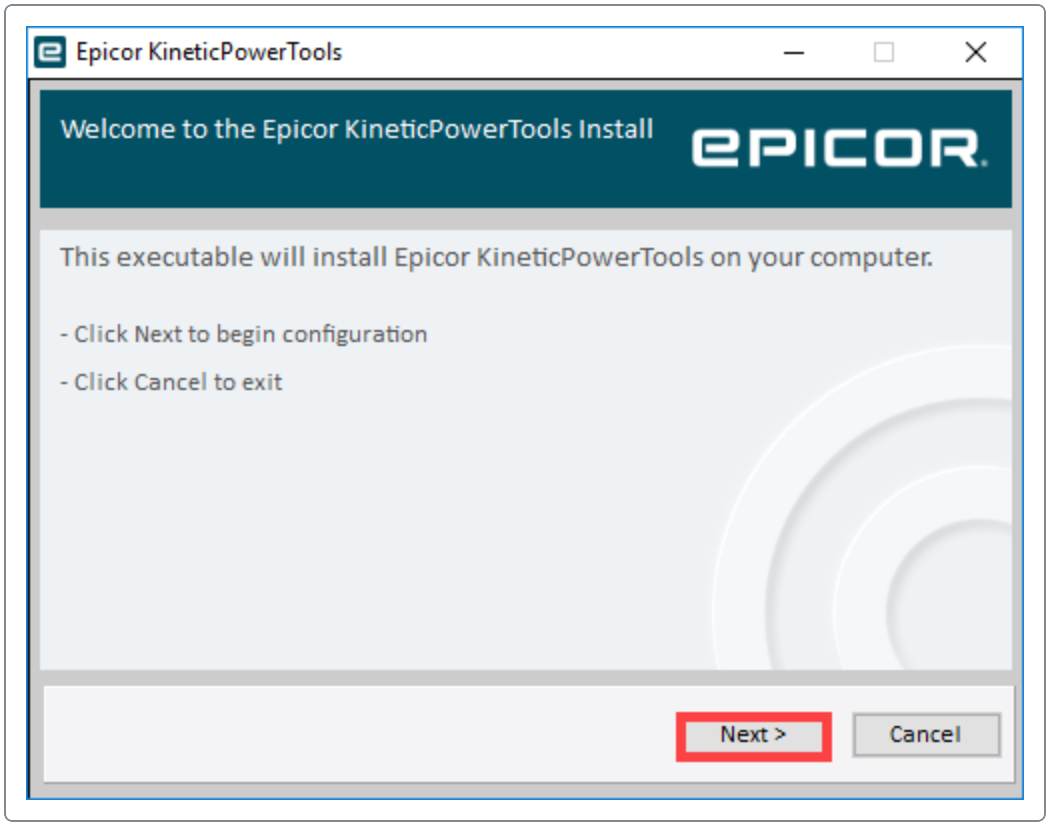
Do the following to install the Kinetic Power Tools client:

1. Go to the directory where you downloaded the installer.
2. Double-click the **kineticpowertools.exe** file.

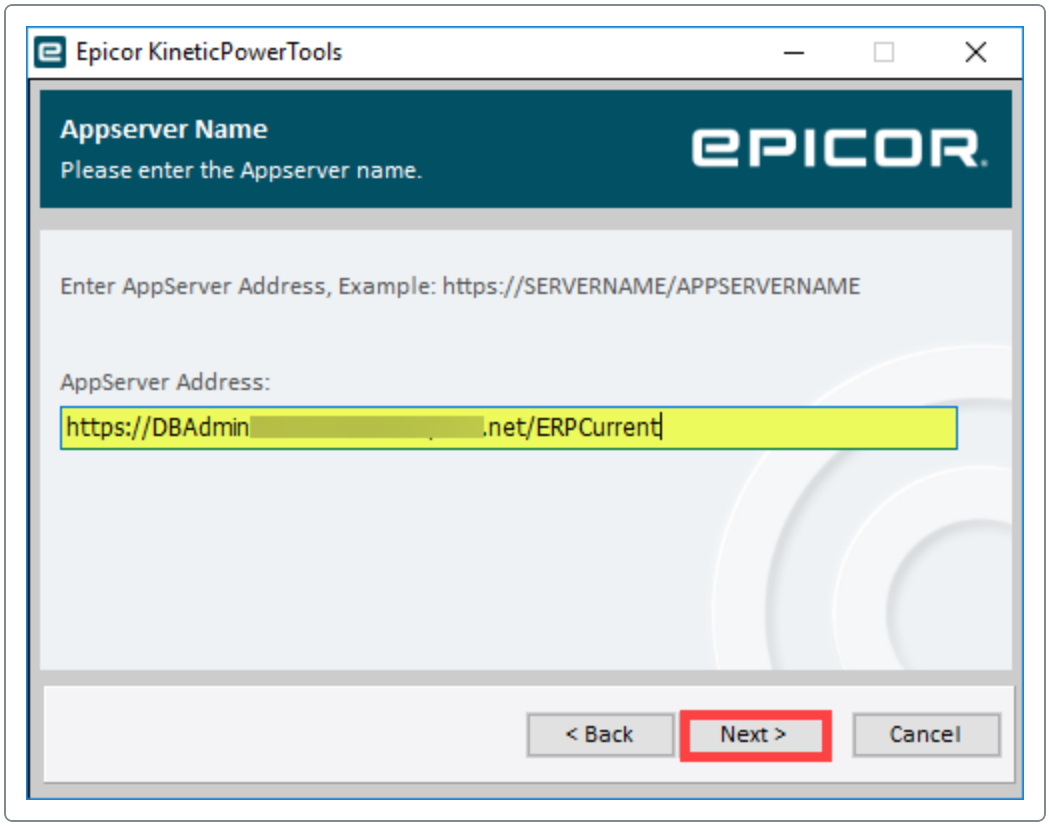
   If you get an error that the installer requires administrator permission, right-click the **kineticpowertools.exe**. From the context menu, select **Run as Administrator**.

   The **Welcome to the EpicorKinetic Power Tools Install** step appears.

3. Press **Next**.

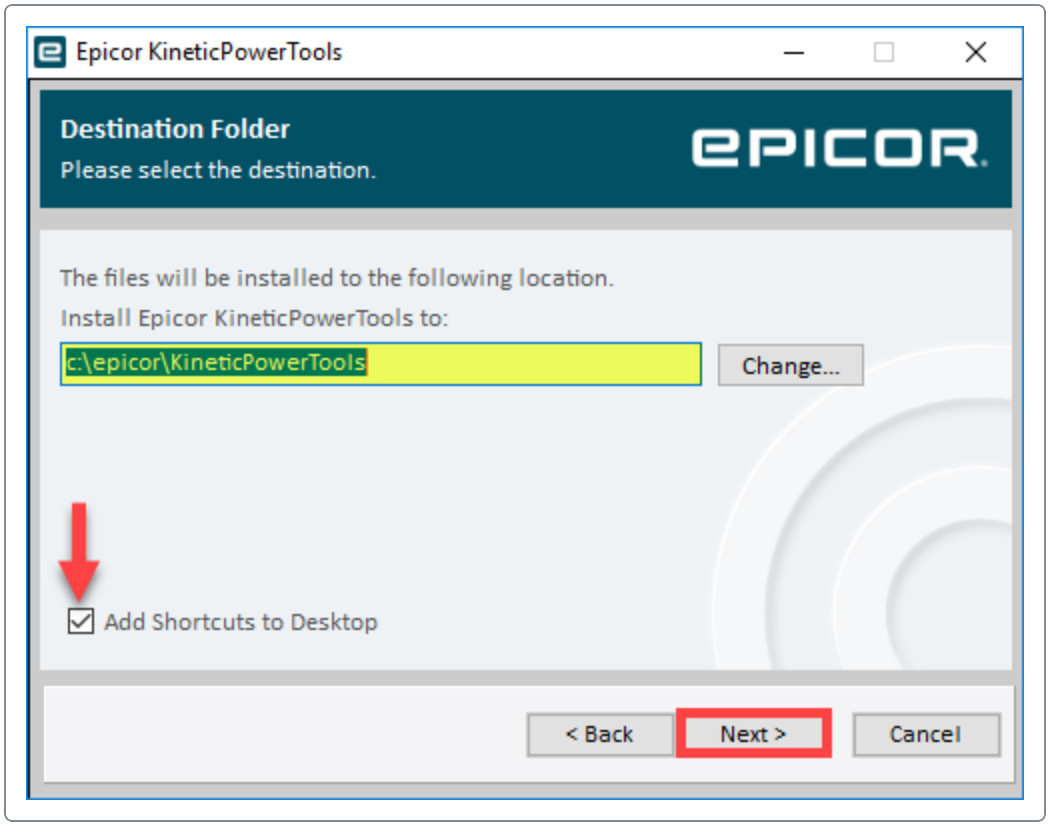


4. Now the **Appserver Name** step displays. Enter the **AppServer Address** (ServerFQDName) for your Kinetic server.



Ensure the URL refers to the server using its Fully Qualified Domain Name (FQDN).
Examples:
Correct Syntax : https://Srv-kinapp.subdomain.domain.com/Kin-Pilot
Incorrect Syntax: https://Srv-kinapp/Kin-Pilot

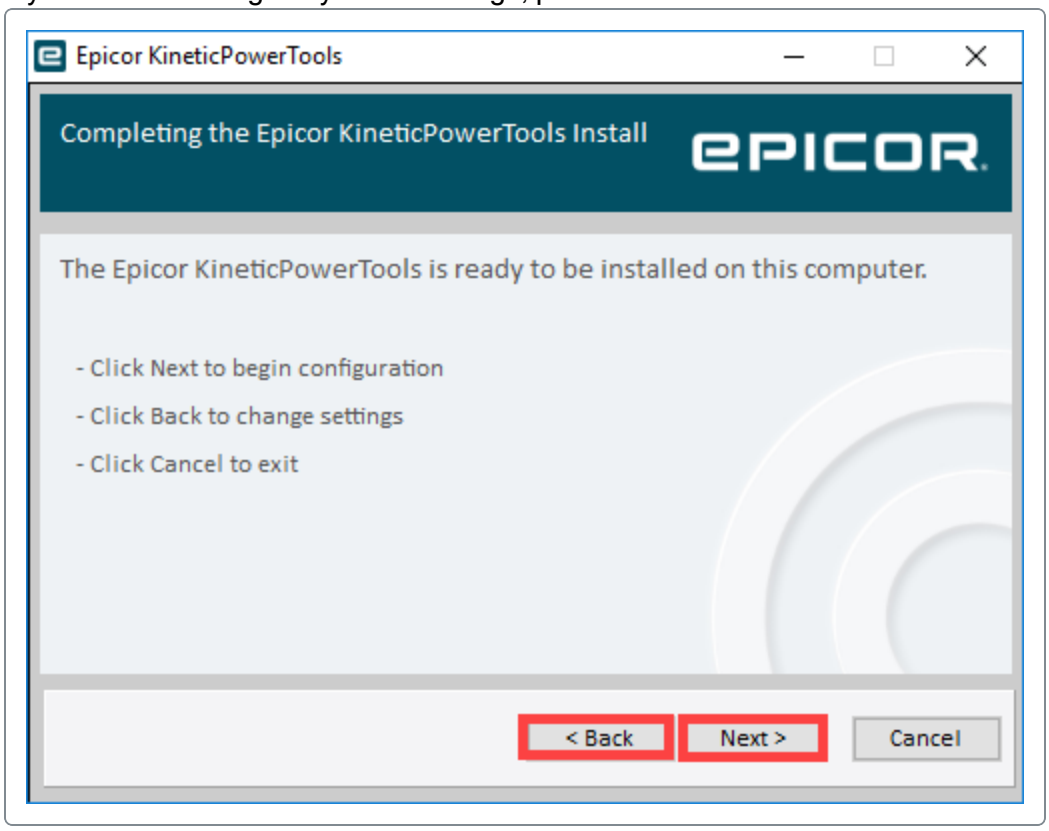
5. Press **Next**.
6. The **Destination Folder** step appears. Enter the local path where you will install the Power Tools files, such as C:\epicor\KineticPowerTools or a similar location. Typically you will use the default location.



7. If you want to add a shortcut to the Power Tools on your desktop, select the **Add Shortcuts to Desktop** check box.
8. Press **Next**.

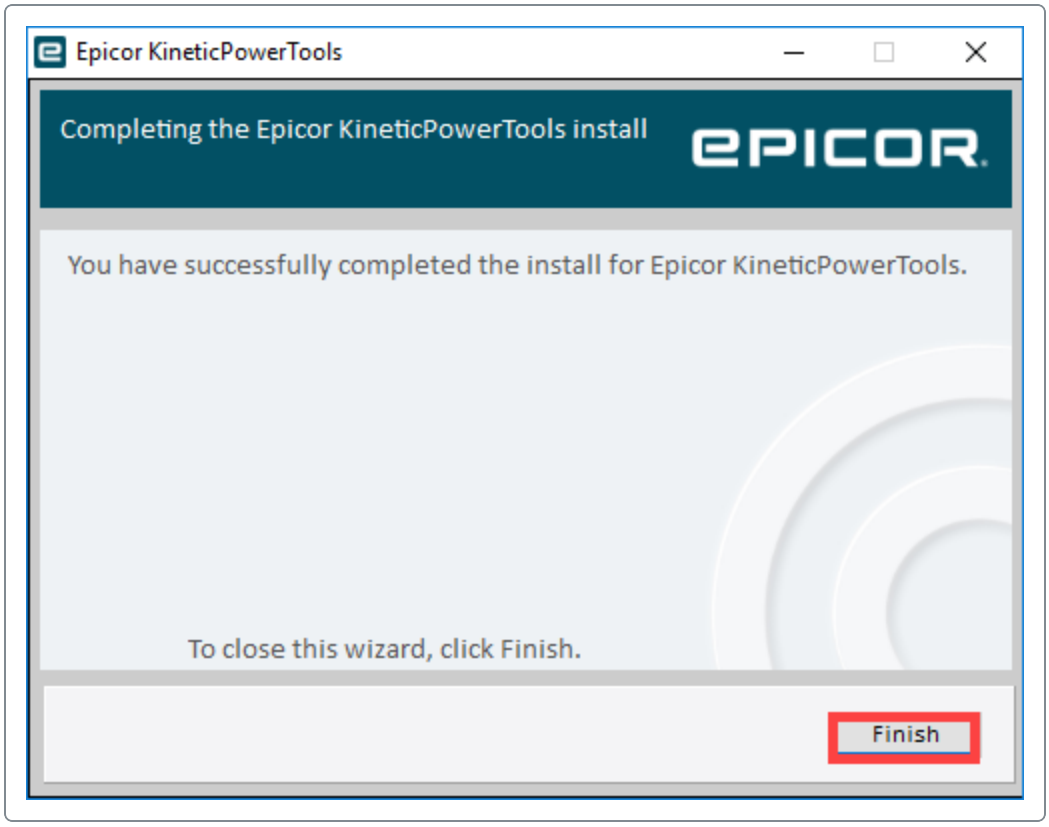
   The **Completing the Epicor Kinetic PowerTools Install** step appears.

9. If you need to change any of the settings, press **Back**.



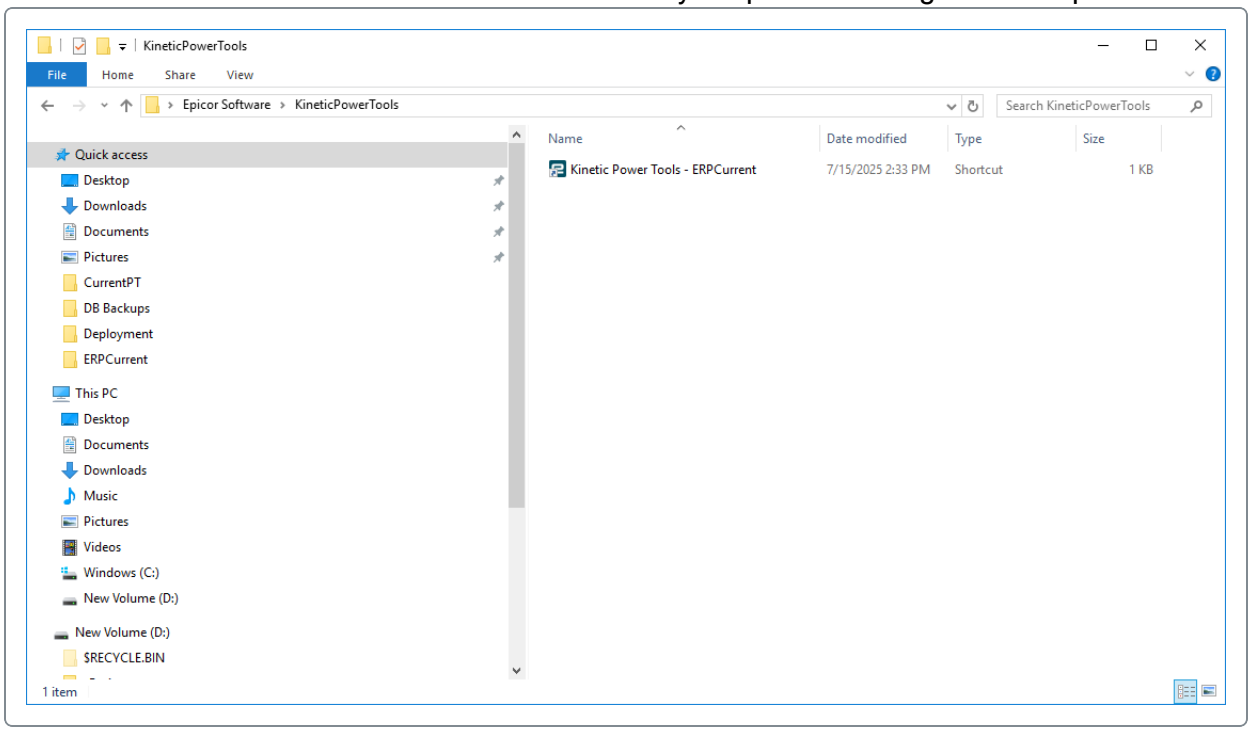
10. If you wish to continue the installation, press **Next**.

    When the installation process finishes, the **Completing the Epicor KineticPowerTools install** step appears.



11. Press **Finish**.

Kinetic Power Tools installs in the destination folder you specified during the install process:



12. You can then move the shortcut to your desktop.

The Power Tools applications are added to your Kinetic environment. While you cannot launch them in the Kinetic browser, you can still adjust security and see the menu information. Do this by launching Menu Maintenance. Go to the NavTree and select the PROCESS folder. The current power tools appear in this menu location.

Using Kinetic Power Tools

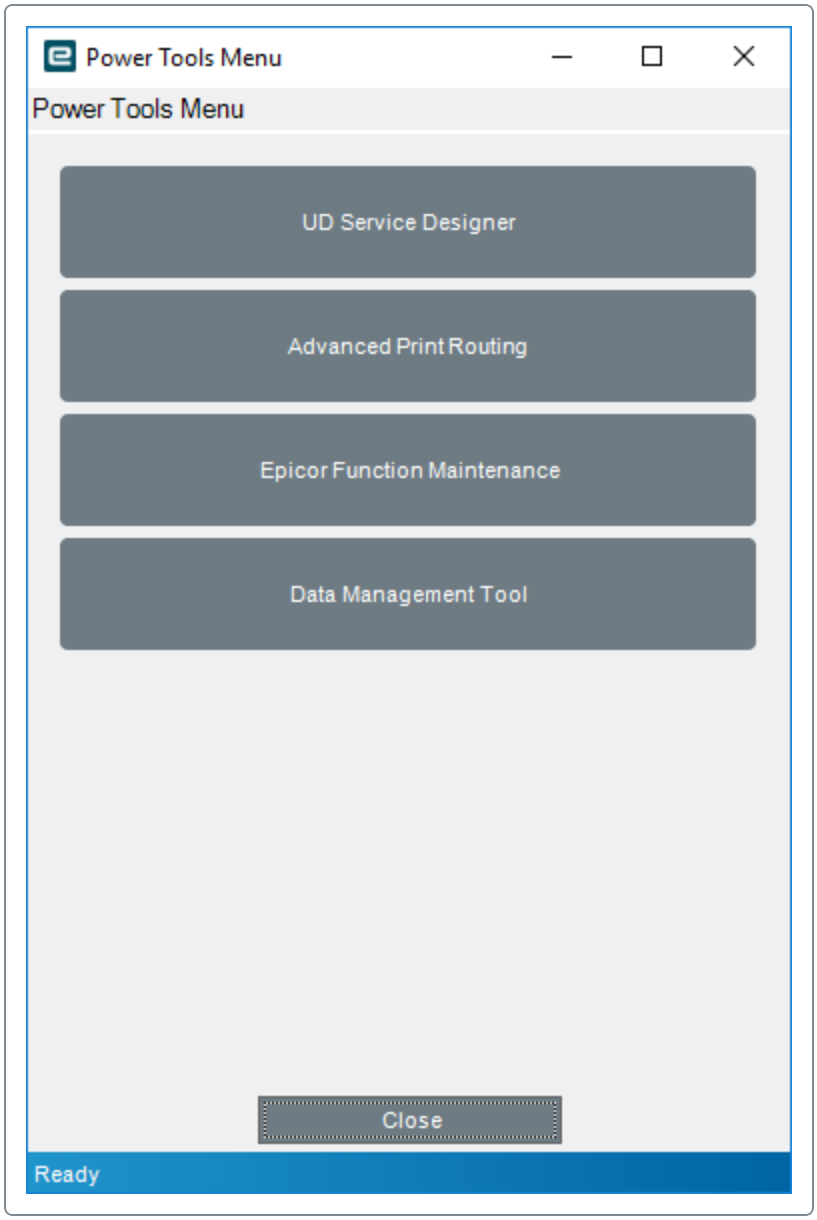
When you install Kinetic Power Tools, you then launch the Power Tools menu. While you can launch all the tools from this menu, your system administrator may need to add more permissions to your user account or activate additional Kinetic licenses. This help article details how you launch the Kinetic Power Tools client and the additional permissions and licenses you may need for each tool.

During this help article, we explore:

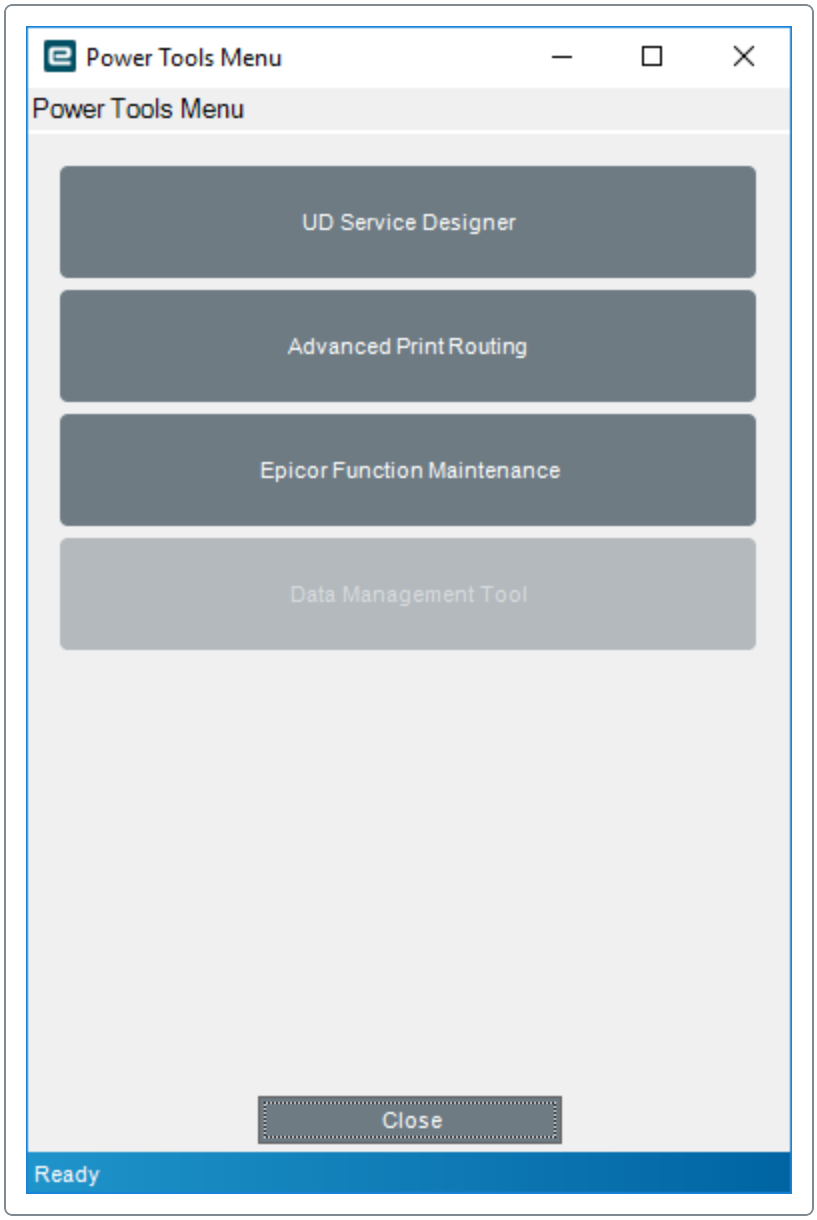
- Launching Kinetic Power Tools
- Setting Up the Kinetic Software Developer Kit (SDK)
- Setting Up Advanced Print Routing (APR)
- Setting Up Epicor Functions
- Setting Up the Data Management Tool (DMT)

Launching Kinetic Power Tools

1. Either go to the folder where you installed Kinetic Power Tools or access your desktop.
2. Double-click the **Kinetic Power Tools** icon.
3. The **Power Tools Menu** displays. If your user account has permission for a tool and its respective license is active, you can launch the tool by pressing its button.



4. However if the tool button IS NOT active, work with your system administrator to finish the setup required for the tool.



Follow the steps below if you are unable to launch Power Tool due to certificate errors:

- Right-click the Power Tools shortcut to view Properties.
- Identify the sysconfig file used (if none is specified, it uses default.sysconfig)
- Select **Open File Location**.
- From **ConfigEditor.exe** access, select **sysconfig** file and click **Open**.

- In the **Application** tab, verify if the AppServer URL refers to the Fully Qualified Domain Name.

Review the sections below for the additional setup you may need for each tool.

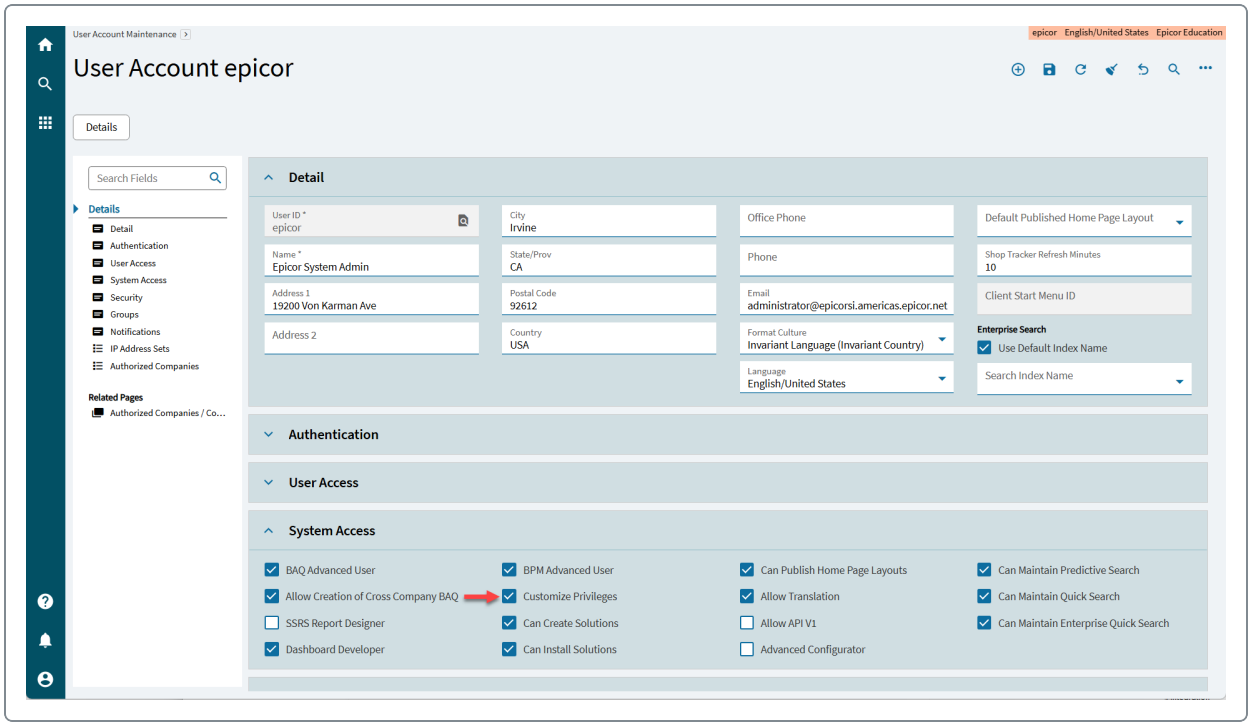
Setting Up the Kinetic Software Developer Kit (SDK)

You can run the Kinetic Software Developer Kit (SDK) when the **SDK license** is active in your Kinetic environment. When you launch the Kinetic Power Tools client, you can then access the **UD Service Designer**. Launch this tool to create a custom user-defined (UD) service.

The Kinetic SDK will launch the UD Service Designer by default; you use this application to create custom services for Kinetic. If you need to develop custom services for Classic, change a value in the Power Tools desktop icon. Right-click the icon and select **Properties**. Then go to the **Target** field. Remove the **-PT** switch and enter the **-SD** switch. Now when you launch the Power Tools icon, the Classic Service Designer displays.

When you need to access Power Tools again, remove the -SD switch and restore the -PT switch.

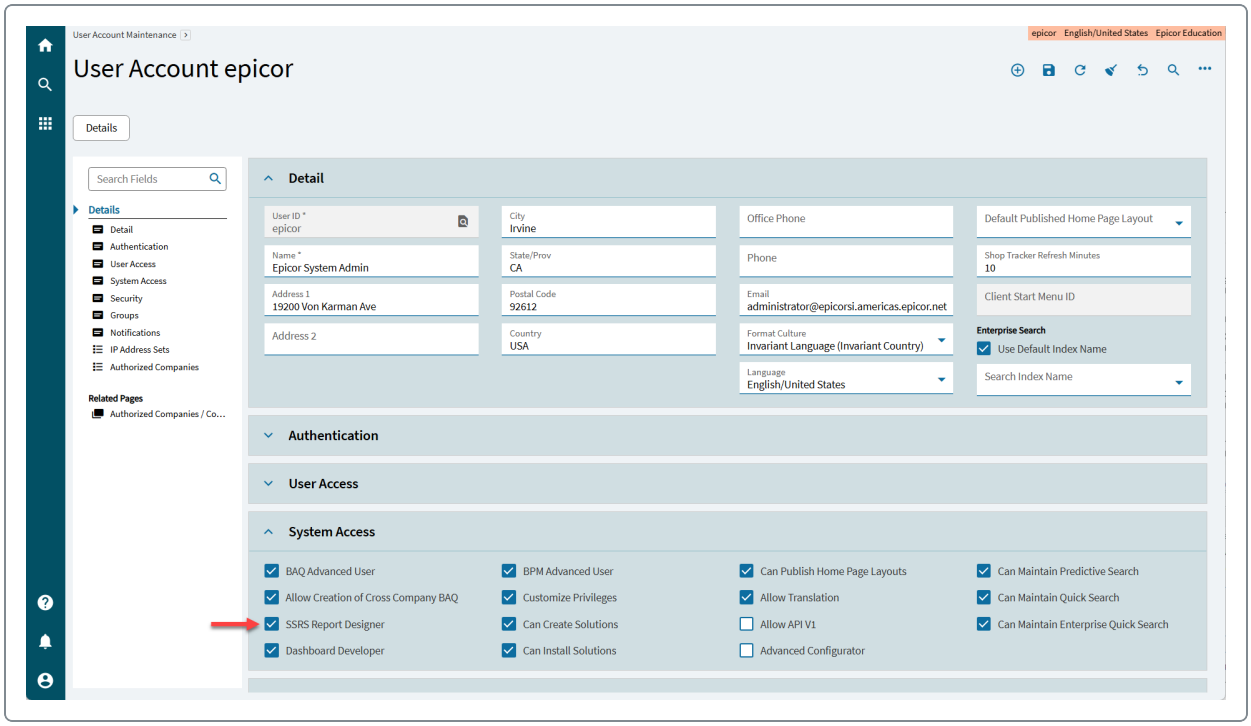
You typically will design the user interface (UX) for this UD service as well. Do this within Kinetic using **Application Studio**. You can access this toolset if your user account has **Customize Privileges**. Your system administrator grants you these rights in **User Account Security Maintenance** on the **System Access** card:



Setting Up Advanced Print Routing (APR)

The Advanced Print Routing (APR) toolset is an additional toolset that you launch from **Report Style Maintenance**. You can access this toolset when the **Advance Printing license** is active in your Kinetic environment.

You typically will create custom SSRS reports to design alternate styles and workflows. To access all the SSRS features in Kinetic, your user account needs **SSRS Report Designer** rights. Your system administrator grants you these rights in **User Account Security Maintenance** on the **System Access** card:

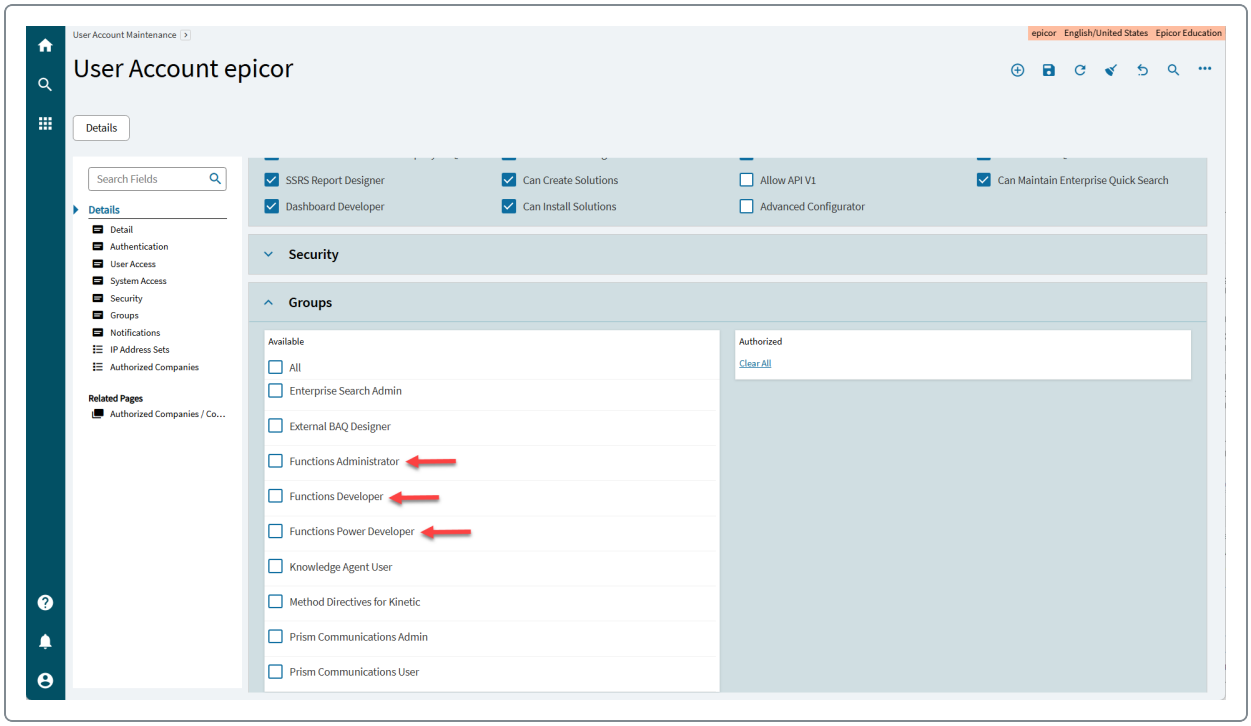


Setting Up Epicor Functions

Epicor Functions Maintenance is included with Kinetic. However if you user account IS NOT authorized to use one of the Functions security groups, you cannot launch this application. Each Functions security group grants a specific level of access to this toolset:

- **Functions Administrator** - Can maintain existing Epicor Function libraries and edit some library properties. These users CANNOT add new Epicor Function libraries.
- **Functions Developer** - Can create new Epicor Function libraries and edit existing libraries. These users can access the Epicor Functions Designer and create workflows that use existing widget options.
- **Functions Power Developer** - Have all Functions Developer rights and can also add custom code widgets to workflows and custom code functions to Epicor Functions libraries. These users can also create custom code that accesses the Kinetic database.

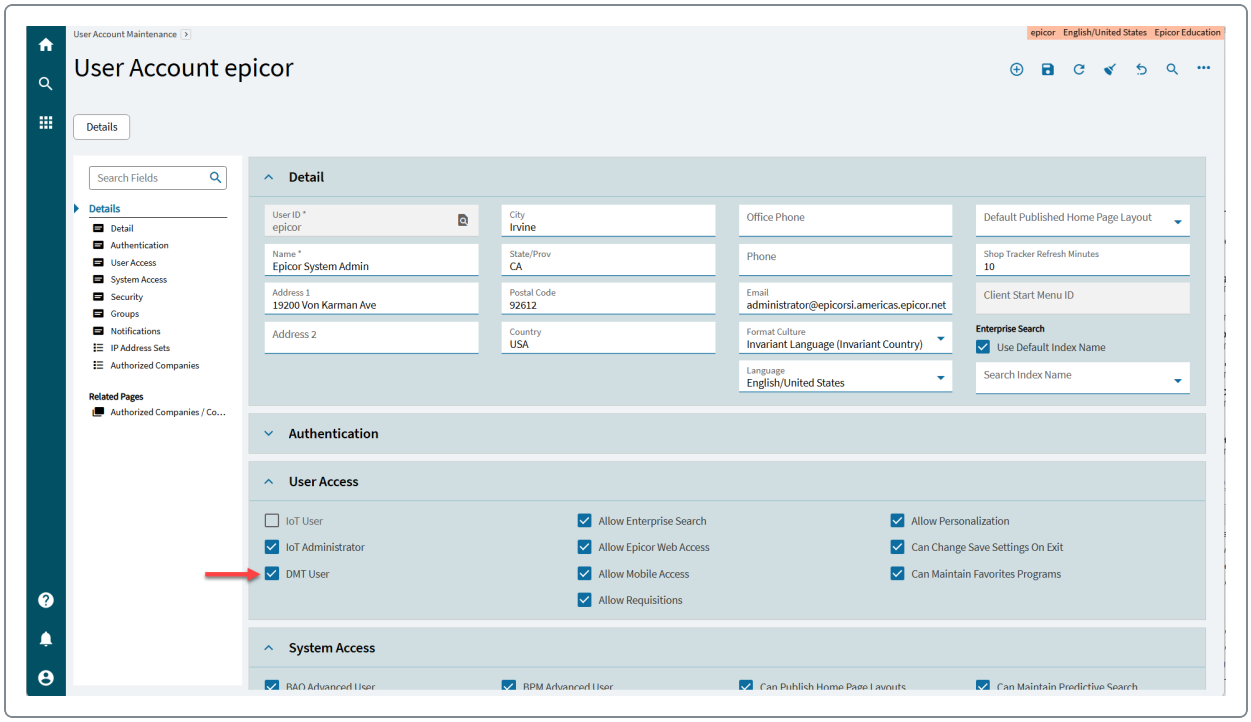
Your system administrator authorizes your appropriate security group in **User Account Security Maintenance** on the **Groups** card:



Setting Up the Data Management Tool (DMT)

You can run the Data Management Tool (DMT) when the **DataManagementTool license** is active in your Kinetic environment.

Your user account also needs **DMT User** rights. Your system administrator grants you these rights in **User Account Security Maintenance** on the **User Access** card:


User Account Maintenance
epicor English/United States Epicor Education
User Account epicor
Details
Search Fields
Details
Detail
Authentication
User Access
System Access
Security
Groups
Notifications
IP Address Sets
Authorized Companies
Related Pages
Authorized Companies / Co...
Detail
User ID * epicor
City Irvine
Office Phone
Default Published Home Page Layout
Name * Epicor System Admin
State/Prov CA
Phone
Shop Tracker Refresh Minutes 10
Address 1 19200 Von Karman Ave
Postal Code 92612
Email administrator@epicorsi.americas.epicor.net
Client Start Menu ID
Address 2
Country USA
Format Culture Invariant Language (Invariant Country)
Enterprise Search
Use Default Index Name
Language English/United States
Search Index Name
Authentication
User Access
IoT User
IoT Administrator
DMT User
Allow Enterprise Search
Allow Epicor Web Access
Allow Mobile Access
Allow Requisitions
Allow Personalization
Can Change Save Settings On Exit
Can Maintain Favorites Programs
System Access
BAQ Advanced User
BPM Advanced User
Can Publish Home Page Layouts
Can Maintain Predictive Search

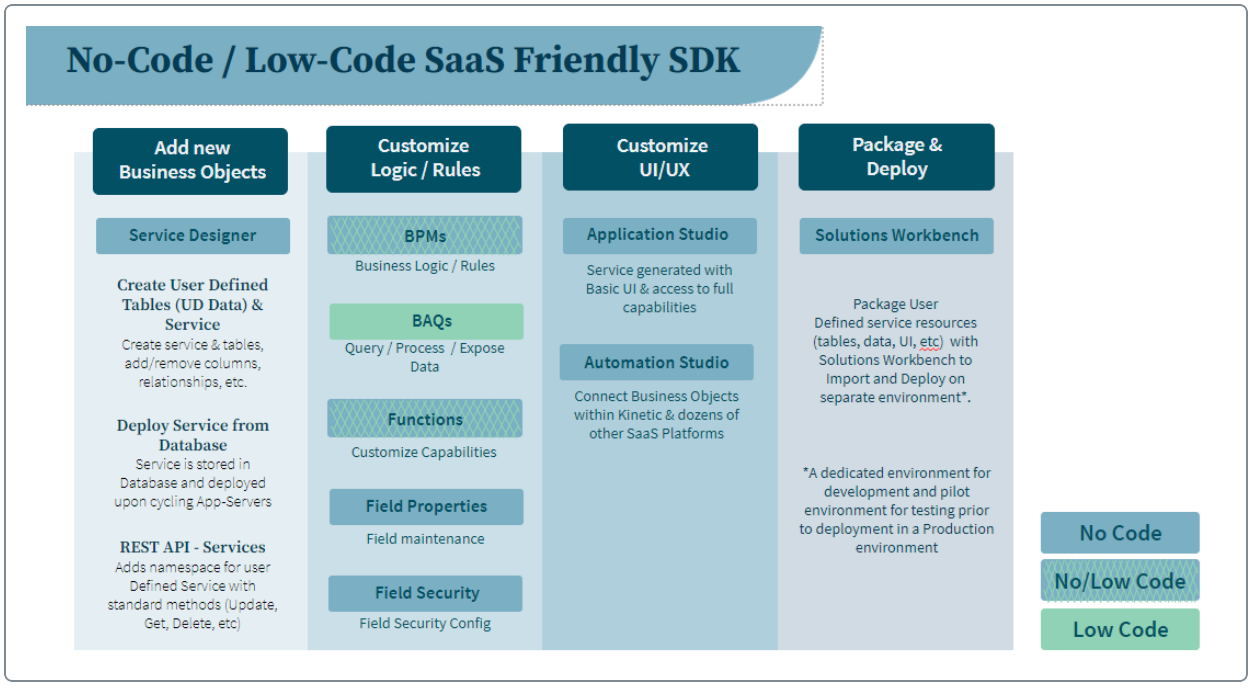
Kinetic Software Developer Kit (SDK)

The **Kinetic Software Developer Kit** (SDK) is available if you purchase an SDK license. Typically, Epicor partners and customers with more complex customization projects use the Kinetic SDK. This toolset is included with Kinetic Power Tools. You launch the SDK from your desktop using the Power Tools Menu.

You can only use this toolset within on-premises and public cloud environments. You cannot use it within multi-tenant cloud environments.

This SDK toolset broadens your ability to customize the Kinetic system. You expand it by creating innovative solutions for your unique business needs. Do this by creating and then adding a User-Defined (UD) service to your system. This service has its own namespace within the REST API, and it also has its own set of tables and basic methods.

If you run Kinetic in a cloud environment, you work with Epicor Support or use the Cloud Management Portal to integrate your UD service. If you run Kinetic in an on-premise environment, you access the server and integrate the service using the Epicor Administration Console. In either environment, you regenerate the data model so the UD service links with your database. It is a no-code/low-code extension that you can then modify using other system tools.



The UD service can interact with all the system logic and rules. Use Business Process Management (BPM) directives to customize actions the service runs. You can also extend its methods by creating Functions that handle unique tasks. Create Business Activity Queries (BAQs) that pull UD service data for display in custom dashboards and BAQ reports.

Launch Application Studio to modify the user interface for the UD service, as well create events and data rules that define how users interact with the custom application. Create recipes in Automation Studio that run automatic tasks that use and/or update data in the UD service. For example, if a currency exchange rate changes in an external application, an Automation Studio recipe can then update this currency exchange rate within the UD Service data.

Before you run the UD service in your live environment, thoroughly test it within a pilot or development environment. This environment must mirror your live environment. When the UD service runs as you expect, use Solution Workbench to package the UD service. You can then install this package within your live environment.

Using the UD Service Designer

You create User-Defined (UD) services for new Kinetic applications within the UD Service Designer. When you purchase the Software Developer Kit license, this tool is available by launching the **Kinetic Power Tools** menu.

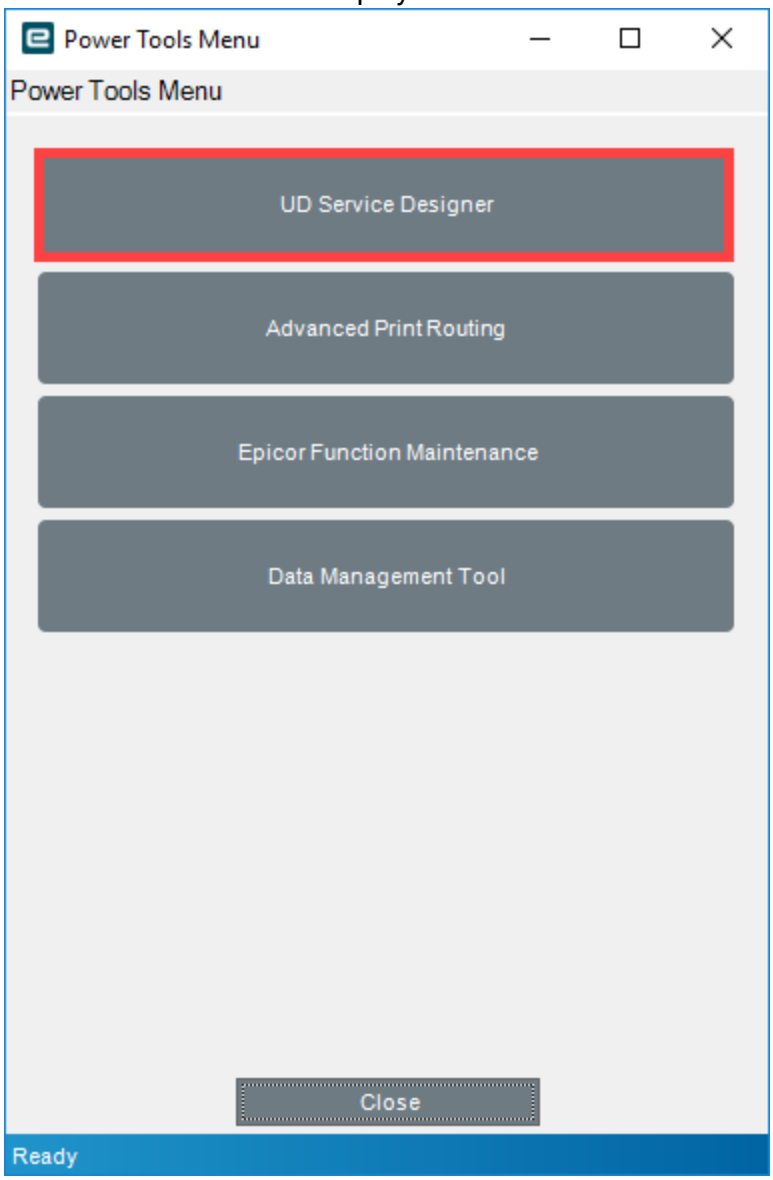
The UD Service Designer is a flexible tool you use to maintain all service components. Use it to define tables, fields, datasets, and so on; each component is independent from the other components. Each component has its own tab on the interface.

Launching the UD Service Designer

You run the UD Service Designer from the Kinetic Power Tools client:

1. Launch the **Kinetic Power Tools** client.
2. Log in with your Kinetic user account.

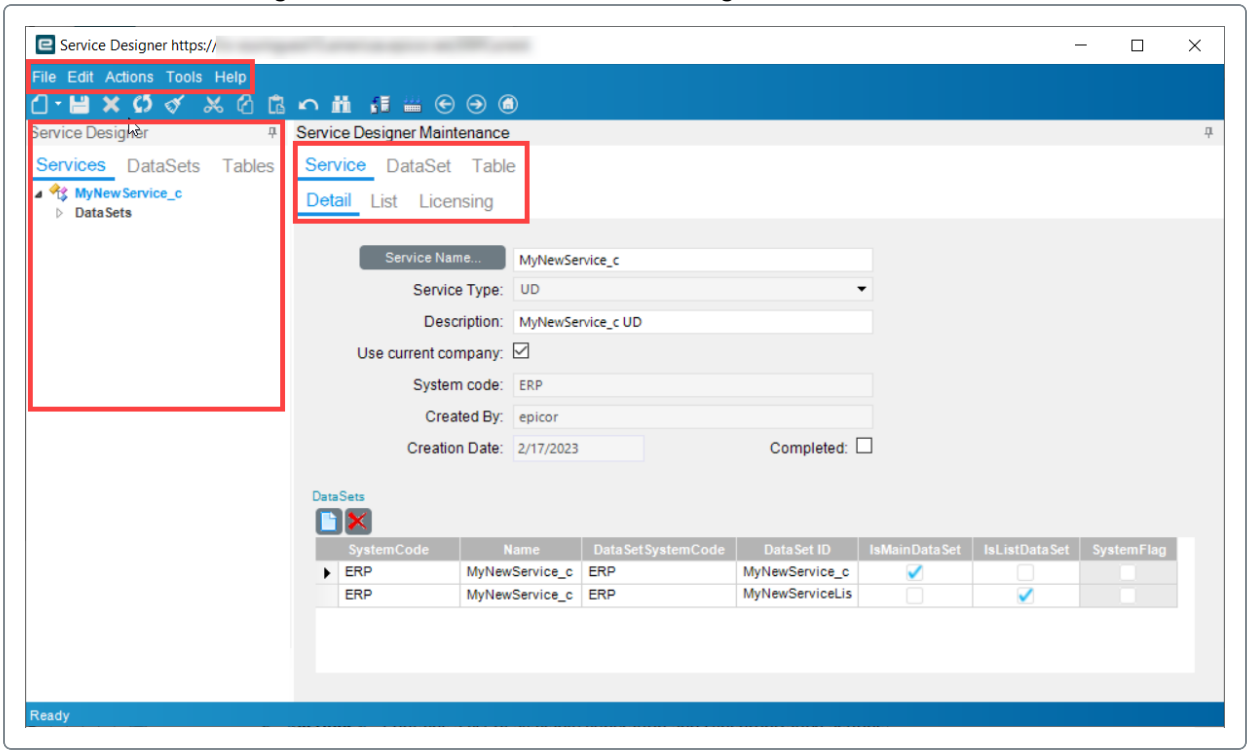
3. The **Power Tools Menu** displays. Press the **UD Service Designer** button.



If this button IS NOT active, your system administrator needs to activate the **SDK license** for your Kinetic environment. You should also make sure that your user account has customization rights.

Reviewing the UD Service Designer Interface

The UD Service Designer interface contains the following elements:



1. The Main Menu bar:
    a. **File** - Use this menu item to create, save or delete services and service components, or to exit the program.
    b. **Edit** - Provides various editing options.
    c. **Actions** - Contains a list of generation and synchronization actions.
    d. **Tools** - Use these tools to personalize the Service Designer interface.
    e. **Help** - Contains links to the application help and EPICWeb.
2. Service Designer Maintenance Panel:
    a. **Service tab** - Contains Detail, List, and Licensing sheets.
    b. **DataSet tab** - Contains Detail, List, and Relation sheets.
    c. **Table tab** - Contains Detail, List, Column, Link, and Key sheets.
3. Service Designer Tree View:

a. **Services**, **DataSets** and **Tables** tabs

b. Displays the tree view of the service component structure.

Interface Tips

- As you work on a service, it is easier to use the Service sheets (Detail, List, Licensing) and then use the Tree View to navigate to the other components.
- Only have one service and its components open within the UD Service Designer at a time. This is especially true if you are updating an existing service. Use the Clear button on the toolbar to keep the number of open services to a minimum.

It is recommended that you install the SDK license in a development or pilot environment. Create and then thoroughly test your customizations in this environment. After you finish testing, use the Solution Workbench to package your customizations into a solutions file. You can then use the Solution Workbench to import your customizations into your live environment.

An important part of the interface is the **Tree View**, a panel found on the left side of the UD Service Designer. How the Tree View refreshes depends on how you load in a service. If users populate the Tree View using the Search program, the selected service and its components load in, or overlay, the Tree View nodes. If users enter components directly through a field, all the components and their nodes appear, or append, within the Tree View.

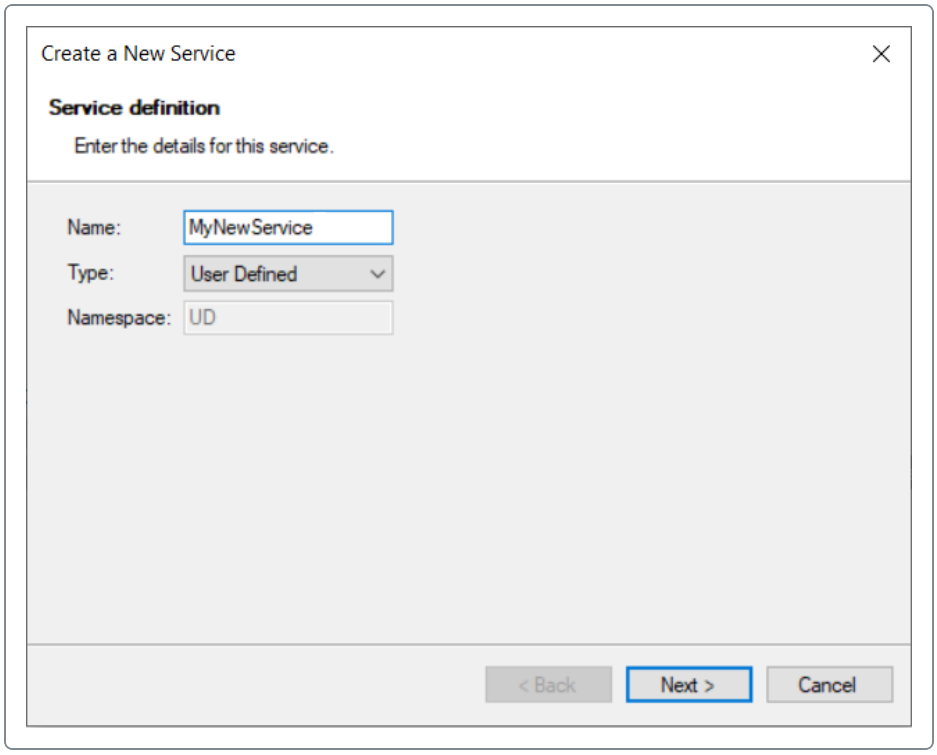
Example: Select the Tables tab and, within the Tree View, the Tables node. Within the DataTable ID, enter ShipVia and press <Tab>. Now click File > New Table, within the DataTable ID, enter Part and press <Tab>. Both the ShipVia and the Part tables appear in the Tree View. Now click the DataTableID button and select the ABCCode table. Only the ABC Code table appears in the Tree View, and its higher-level objects populate the Tree View with its lower-level components.

When you select a service through a search, the service first populates the DataSet and Tables views. A DataSet next populates the Tables views.

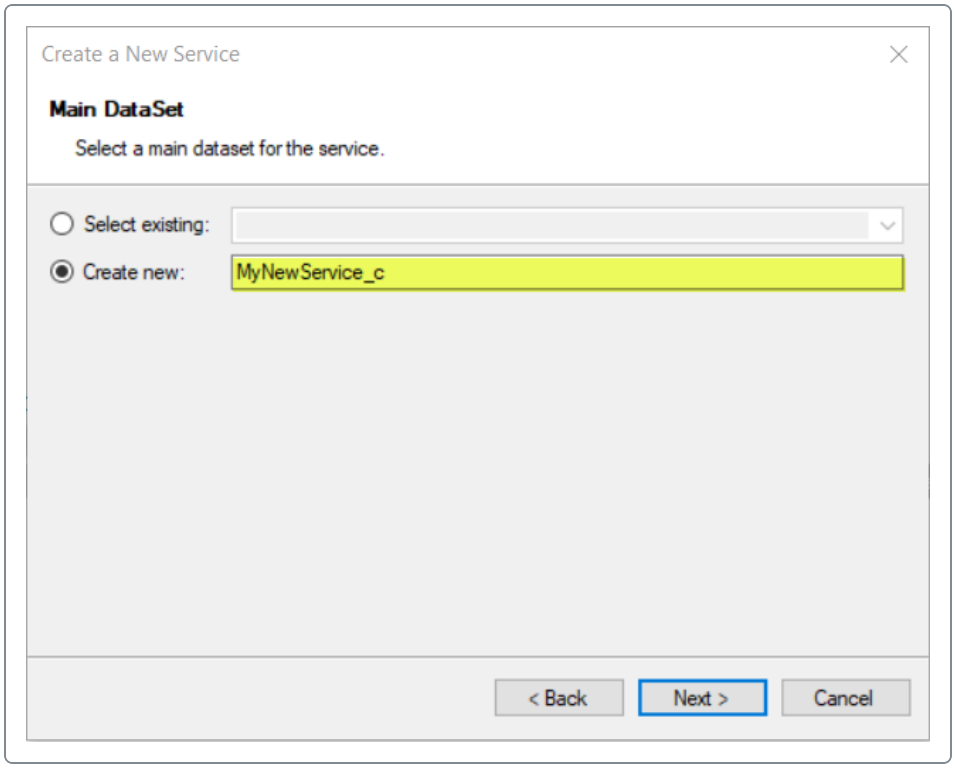
Creating New Services

Use the Service Creation Wizard to build new UD services in the UD Service Designer.

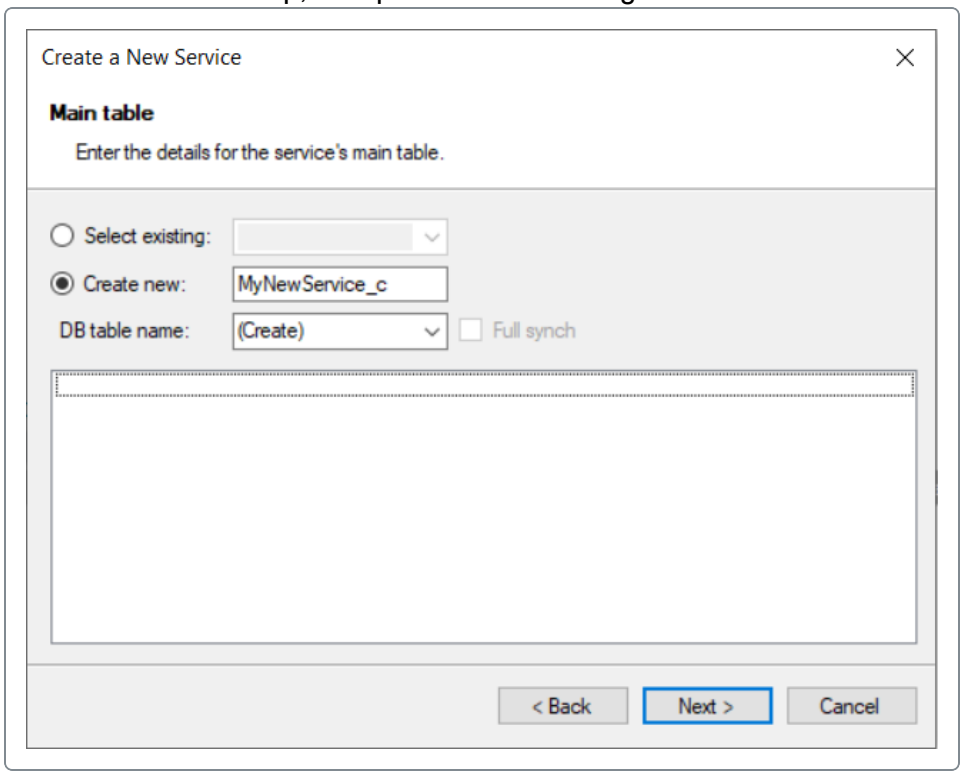
1. In the main toolbar, select the **Down Arrow** next to the **New** icon and select **New Service**.
2. The **Create a New Class** window displays. On the **Service Definition** step in the **Service Name** field, enter the name of your service.



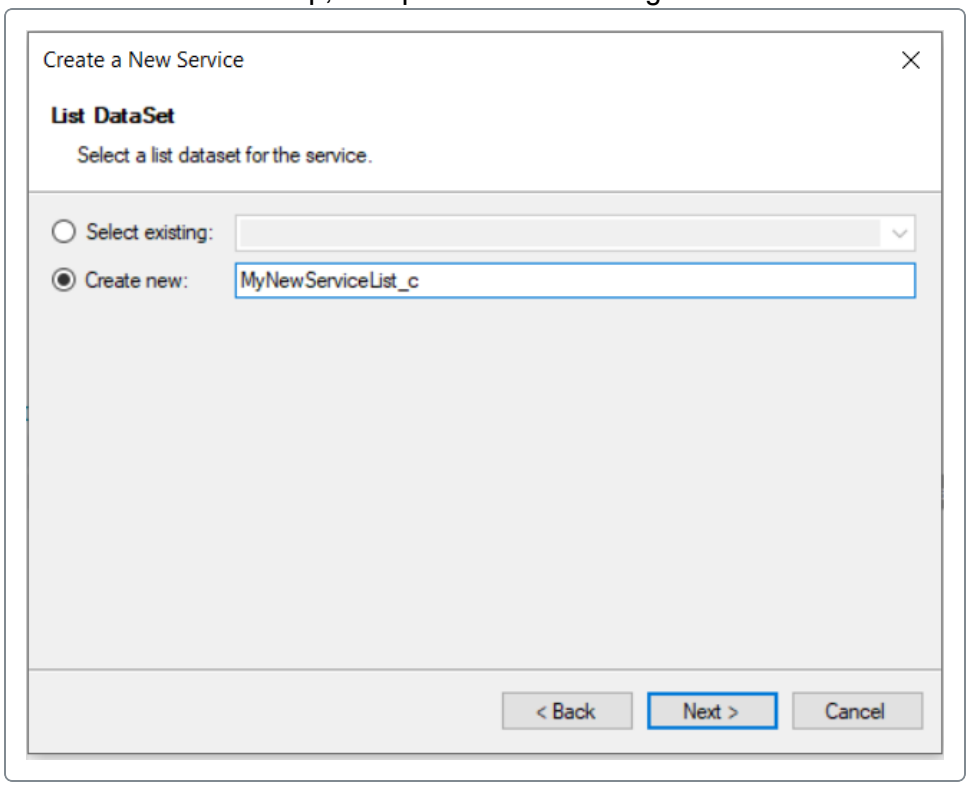
3. The **Type** drop-down list selects the **User Defined** service type by default.
4. Click **Next**.
5. On the **Main DataSet** step, leave the default value in the **Create new** field. The UD Services Designer automatically adds a "_c" suffix to the dataset. This indicates its a custom table and makes sure it does not conflict with other tables in the database. Select **Next**.



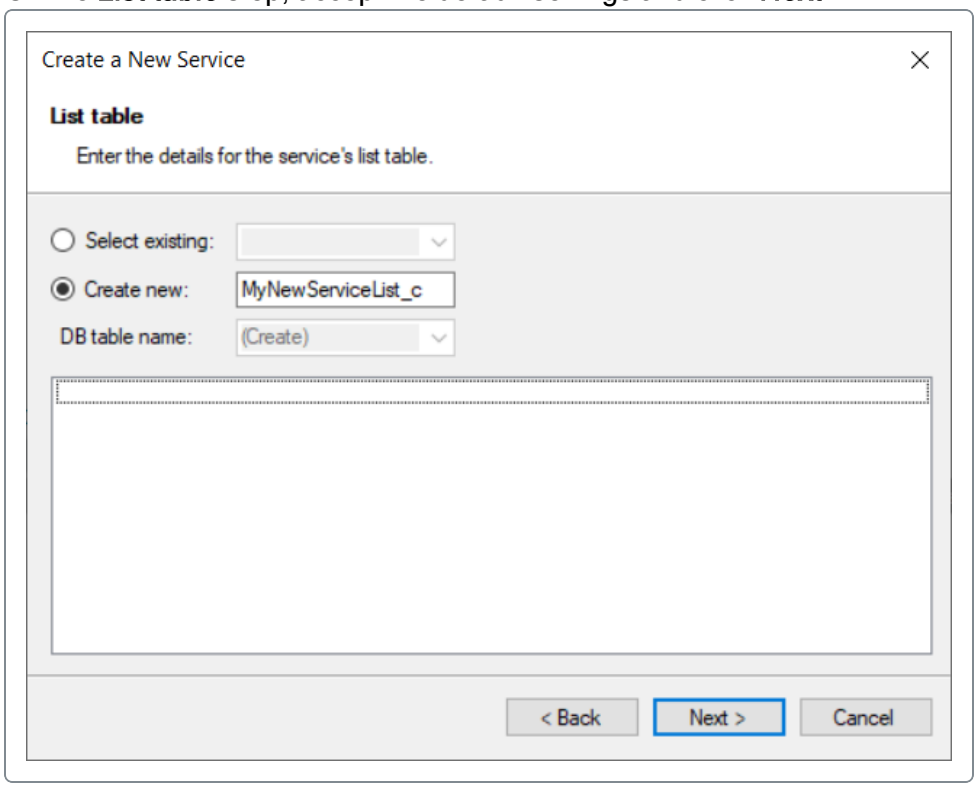
6. On the **Main table** step, accept the default settings for the table and select **Next**.



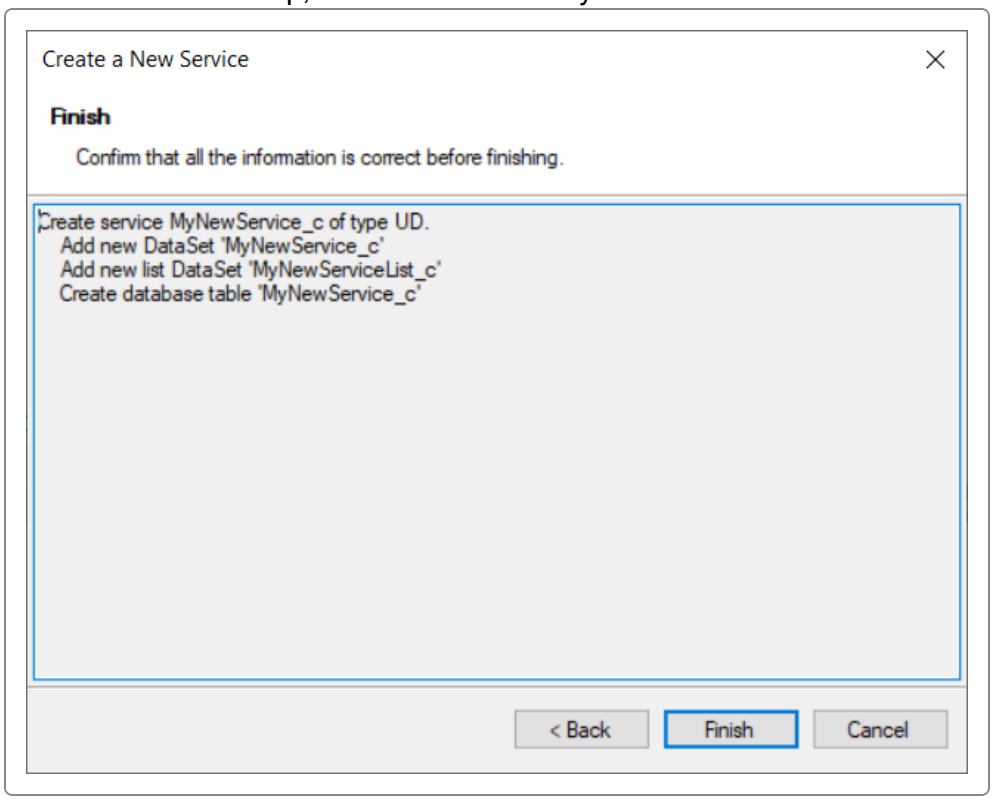
7. On the **List DataSet** step, accept the default settings and click **Next**.



8. On the **List table** step, accept the default settings and click **Next**.



9. On the final **Finish** step, review the details of your service and click **Finish**.



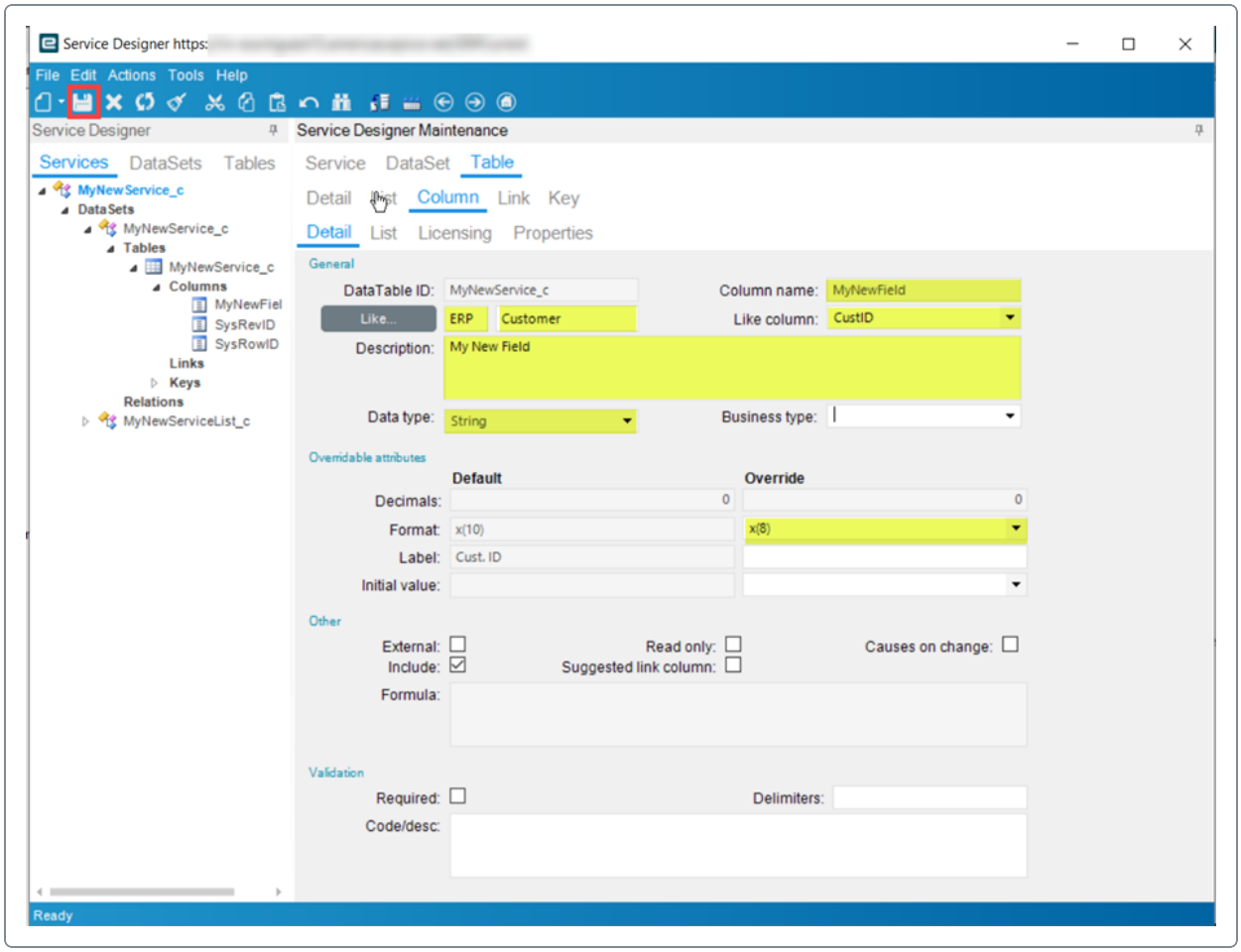
Adding Fields

You can add additional fields to your new service table. They append to the table. When users enter data in these fields, the framework automatically populates them with this entered data for display.

Each UD service can have up to three (3) indexes, and then each index can have up to four (4) key columns. This prevents UD services from slowing database performance.

To add a field:

1. On the main toolbar, select the **Down Arrow** next to the **New** icon and select **New Column**.
2. The **Table > Column > Detail** sheet displays. Enter the **Column name**. This is the value that identifies this field in the database, such as MyNewService_c.MyNewField. You use this value to display the column data by adding it to the interface in Application Studio.



3. Enter the **Like** field and **Description** to define the key field this column uses in the database. Any tables that share this Like field can then display the data from this new column (field).

4. The **Like column** defines what key field outside this table you will use to link this field to data in a related table, such as CustID.

5. Select the **Data type** for this field. Available options include String, Numeric, Integer, and so on.

6. After you select the Data type, the default **Format** for this type appears, like x(10). This indicates the new field holds an alphanumeric value ten characters long. You can change this default format in the **Override** column.

7. **Save** the field.

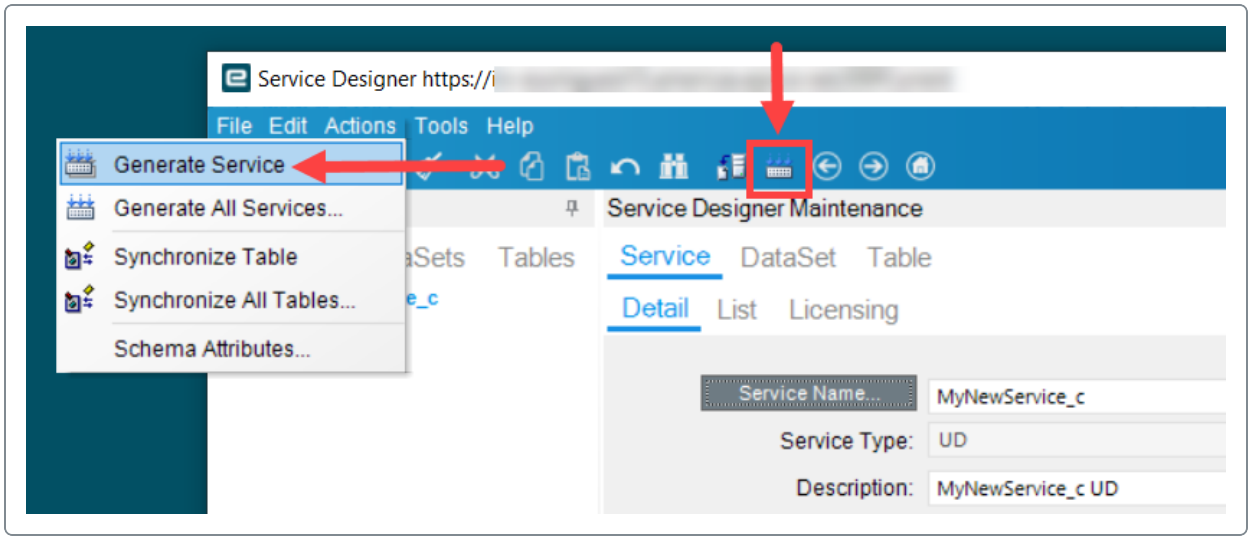
Continue adding the fields you need for the UD service.

Deploying UD Services

After you finish creating the UD service and adding fields to it, you are ready to deploy it. This makes the service available to customize and use within Kinetic.

> If you deploy the UD service and then later add more fields to it in the UD Service Designer, you will need to redeploy it following the steps in this section.

1. Regenerate the data model for your environment. This builds the table structure. You can do this in two ways:
    - Contact **Epicor Support** to submit a support case. Explain you are adding a UD service to your Kinetic environment. Epicor Support will then regenerate the data model.
    - Log into the **Cloud Management Portal**. You then select the instance for your environment and regenerate the data model.
2. Now return to the Service Designer to generate the UD service. This synchronizes the service with the table structure in your Kinetic database.
3. Go to **Actions > Generate Service**. You can also select the **Generate** button on the toolbar.



4. The **Generate Service** dialog box displays. Select **Yes**. Your UD service integrates with the current table structure.
5. Stop and restart the application pool. This refreshes your Kinetic environment with the generated assemblies for your UD service. You can do this in two ways:
    - Contact **Epicor Support** to submit a second support case. Explain you have generated a new UD service for your Kinetic environment. Epicor Support will then stop and restart

the application pool for your environment.

- Log into the **Cloud Management Portal**. You then select the instance for your environment and stop and restart the site.

The name of service looks like this: Ext.Services.BO.MyNewService

The Ext prefix means your new service generated in your own development environment.

You can now access the UD service in Application Studio, Business Activity Query (BAQ) Designer, and other tools. Learn how to do this in the Reviewing the UD Service help article.

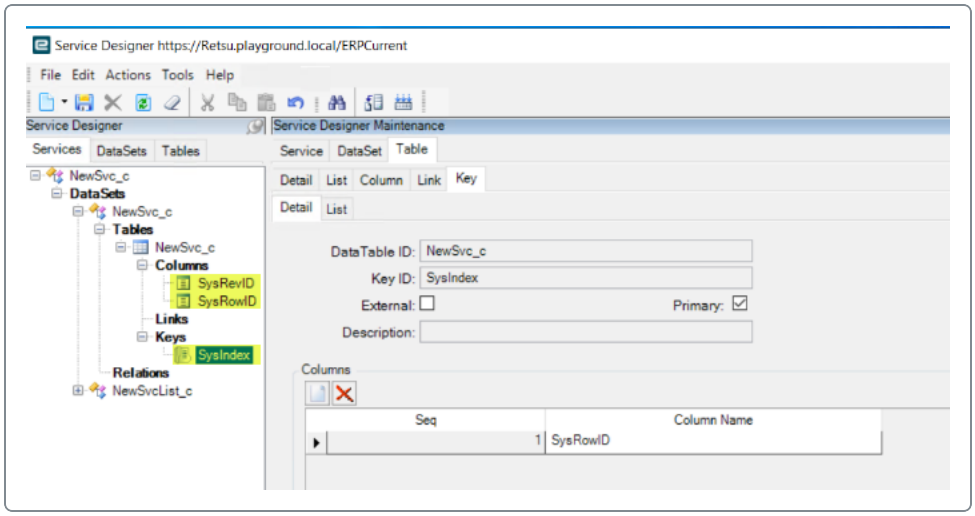
Reviewing the User-Defined Service

When you generate a User-Defined (UD) service, it contains all the standard methods for interacting with the database, such as GetNew, GetById, Update, and so on. It also includes the standard business rules that enforce database integrity.

This help article explores the structure of a UD service. Leverage this information to understand what items the service includes so you can better customize the UI application in Application Studio to handle your business needs.

Table Structure

When you generate a UD service, it contains the basic functionality it needs to run within Kinetic. It has two datasets, one for the main table and then another for the list table. The base service includes the **SysRevID** and the **SysRowID** columns for the main table. It also has the **SysIndex** key field column. This column is the primary key for the service, as by default it uses the SysRowID to identify the key. If you need more key fields, add key field columns to the service before you generate it.

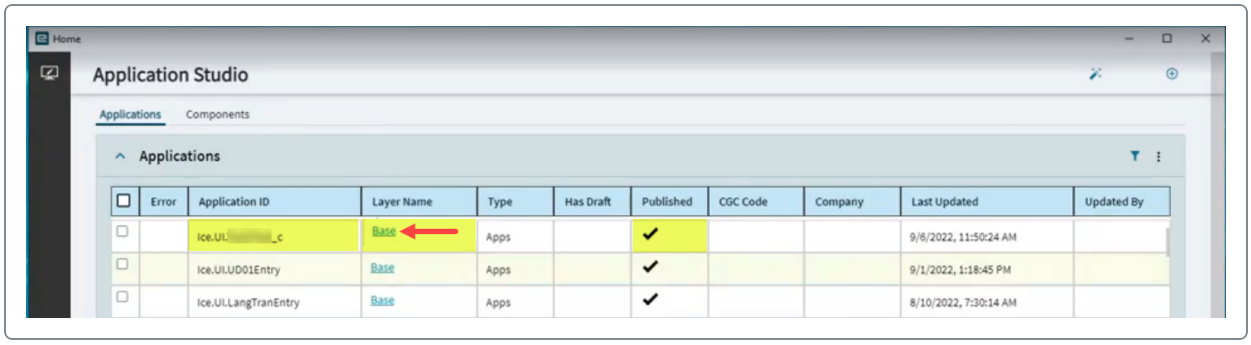


Launching the UI Application

After you generate the UD service and either Epicor Support or you (through the Cloud Management Portal) regenerates the data model and stops/starts the application pool, you can access the UD service within Kinetic. Do this by using Application Studio. The generation process creates a unique UI application for your service that you can customize.

To begin customizing the UD service:

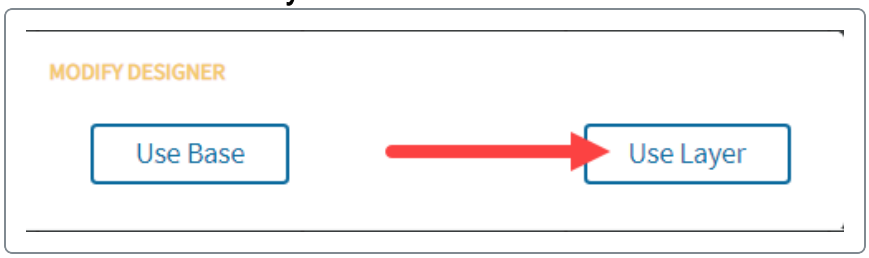
1. Launch **Application Studio Homepage.**



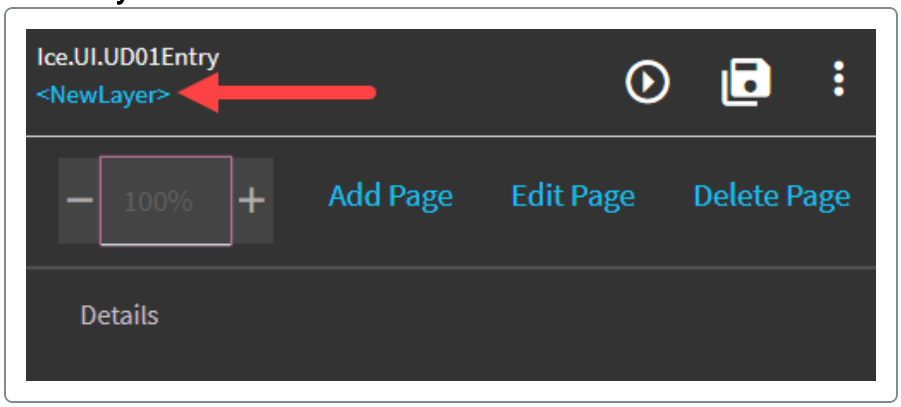
2. Your service displays as a base layer on the Landing Page. It has an **Application ID** and a **Layer Name**. The service also has the **Published** status. The Published status means that you can place this base layer on the menu.

3. Select the **Base** link.

4. The **Modify Designer** dialog displays. As a best practice, create a custom layer instead of modifying the base layer. You can then always return to the base layer if a custom layer has issues. Select **Use Layer**.



5. This launches the user-defined service within **Application Studio**. Select the **<NewLayer>** link.



6. The **Layer Selection** panel slides on. Enter the **Layer Name** for the new custom layer.

7. Now enter a **Description** that briefly describes the purpose of the new layer.

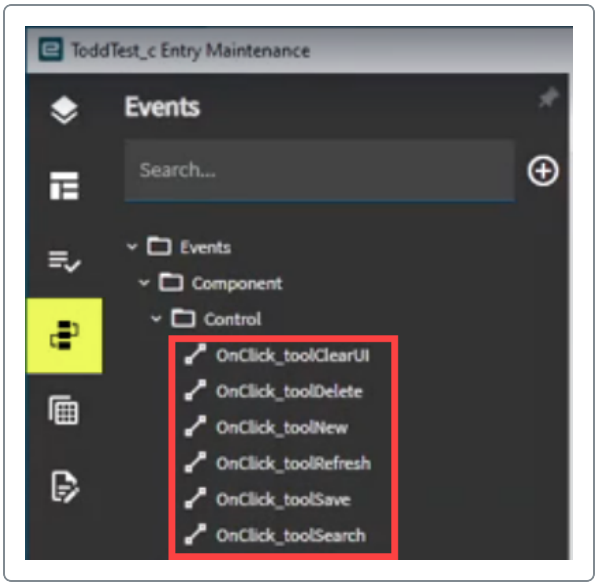
8. Select the **Save Layer** link.

You can now customize the user interface for the UD service. Do this by adding components, such as panel cards, text boxes, combo boxes, and other components to the custom layer. To begin learning how to use Application Studio, review the **Customizing Tools** section later in this help article.

Reviewing Base Events

The UD service generates with the events the system requires to run the UI application.

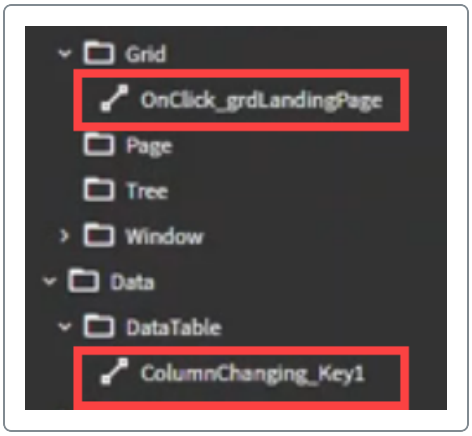
1. Select the **Events** toolbar button to review these base events.



2. The UI application contains the base **Control** click events for adding, updating, and saving records:

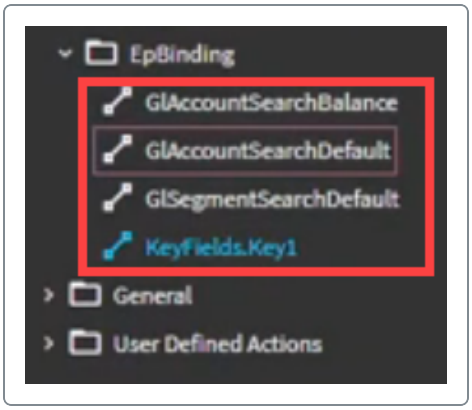
    - **OnClick_toolClearUI**
    - **OnClick_toolDelete**
    - **OnClick_toolNew**
    - **OnClick_toolRefresh**
    - **OnClick_toolSave**
    - **OnClick_toolSearch**

3. The UI application has the **Grid** event, **Onclick_grdLandingPage**, for selecting the Landing Page grid.



4. The UI application also has the **DataTable** event, **ColumnChanging_Key1**, that handles data entry changes within the Key1 field.

5. The default **EpBinding** events links GL accounts, balances, segments, and the Key1 field to the database.



If you need more events, create custom events within Application Studio. For example, the base service only has events that handle the Key1 field. If you need more key fields, create the events that access these key fields.

Customizing Tools

You further customize this service using the following Kinetic tools.

- **Application Studio** - Use this design environment to create applications and processes to fit your business needs. Create a user interface for a UD service and then build events, data rules, dataviews, and other items that interact with it. Review the **Welcome to Application Studio** help article to start learning about this primary development tool. You can also download the Application Studio user guides from EpicWeb ((**https://epicweb.epicor.com/documentation/user-guides**).

- **Automation Studio** - Epicor Automation Studio, powered by Workato®, connects Kinetic with external cloud-based tools that help you automate business workflows across cloud and on-premises applications. Do this by creating recipes that integrate with your database. Start learning how to create recipes by reviewing the **Using Automation Studio** help article.

- **BAQ Designer** - Design business activity queries (BAQs) that pull data from a UD service for display in custom grids, dashboards, and BAQ reports. You create queries in the Business Activity Query (BAQ) Designer. Review the **Working with Business Activity Query (BAQ) Designer** help article for more information.

- **Epicor Functions Maintenance** - Create new methods (functions) for a UD service through this tool. While the UD Service Designer generates the base methods, you may need these additional methods to run a business task. Learn more about Epicor Functions Maintenance by reviewing the **Creating Epicor Functions** help article.

- **Business Process Management (BPM)** - Create method and data directives that cause default values to populate fields, make specific fields mandatory, and so on. To find the methods created for the UD service, search for User-Defined methods. Learn more about creating BPM method and data directives by reading the **Using the BPM Workflow Designer** help article.

- **Extended Properties Maintenance** - Through this application you define additional, or extended properties for selected fields within the UD service tables. Define Like tables, BAQ Zones, and other properties using Extended Properties Maintenance. Read the **Defining Extended Column Properties** help article for more information.

- **Field Security Maintenance** - Use this application to define security privileges for the fields within the UD service table. Limit access to specific fields by restricting and granting access to selected users and security groups. Review the **Maintaining Field Security** help article for details.

- **Menu Maintenance** - After you finish developing the UI application for the UD service in Application Studio, make it available to users in your environment by adding it to the menu through Menu Maintenance. Learn more by reading the **Working with Menu Maintenance** help article.

- **Solution Workbench** - When you are ready to install the UD service in your live environment, package the UD service and its functions, method directives, queries, and other items into a solution file. You create solution files using the Solution Workbench. Learn more about this application by reviewing the **Using Solution Workbench** help article.

Selecting Security Codes for User Defined Applications

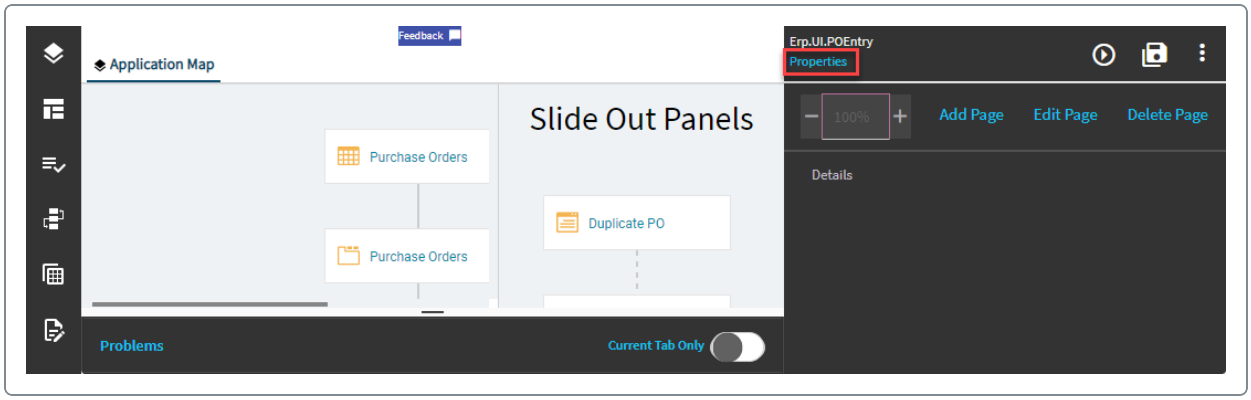
You can define how users access the custom application's base layer by assigning a Security Code. If you are a member of the security group assigned to this application, with your account level setting that allows you edit the base layer, you then get the access to edit the base layer where the security code was added.

If you do not have the required access at your account level setting, you will not be able to edit the base layer even if you are a member of the security group assigned to this application.

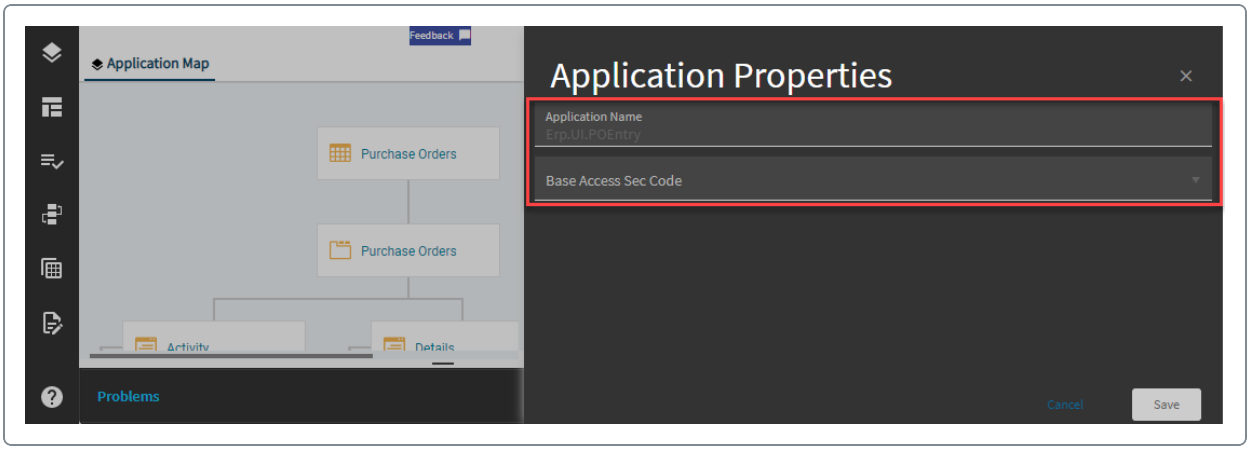
This feature is only available on user defined applications and not available on system applications.

A user with Security Manager access and the creator of the application can now add or edit the security code on the base layer.

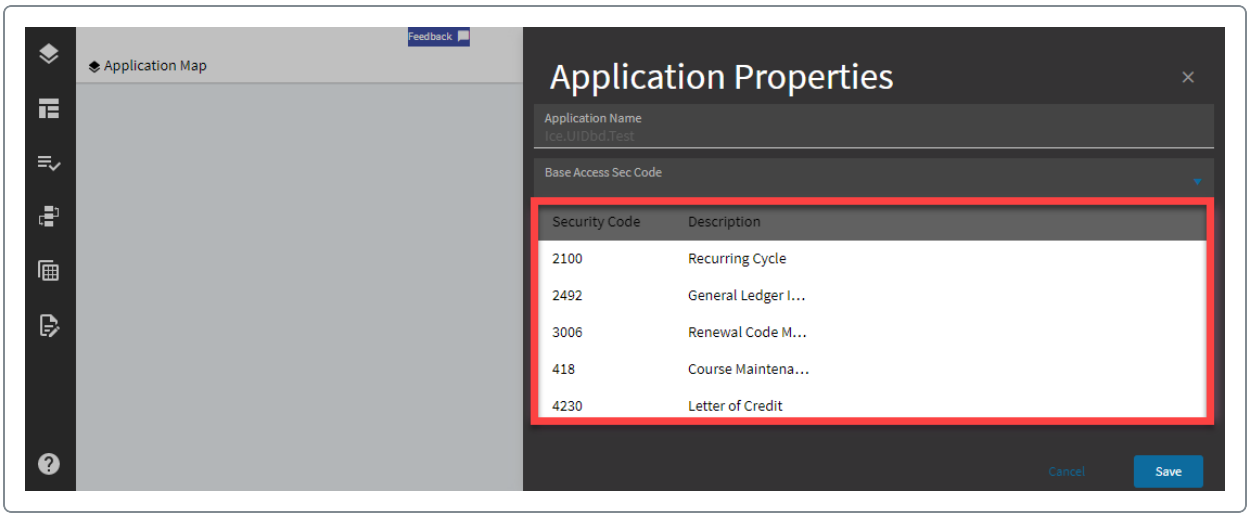
In the custom application's base layer, use the sliding panel on the right to edit the properties. You will now see a hyperlink on **Properties**, below the application name.



When you select **Properties**, the **Application Properties** panel slides in, which displays the **Application Name** and **Base Access Sec Code**.



When you select the arrow on the **Base Access Sec Code**, a combo box with **Security Code** and **Description** displays. You can select only one security code.



Configuring the On-Premise Application Pool

Application Pool service accounts, such as ApplicationPoolIdentity, do not have permissions to generate user defined services. If you will use Cloud SDK in an on-premise environment, you must configure the application pool so it can generate user defined services on the server.

You can configure your AppPool user account with either a **local** or **domain** account. If you will use a domain account, launch the **Computer Management** application on your server to see what domain accounts are available, such as Administrator. If you will use a local account that is stored on your computer, select an account that is for common use. It CANNOT be a service account such as ApplicationPoolIdentity or NetworkService.

1. Launch the **Epicor Administration Console**.
2. Access the application server for your on-premise environment.
3. Update the **AppPool user account** with the local or domain account.
4. **Stop** and then **Start** the application pool.

The account for the application pool runs with enough privileges to generate a user defined service. You can now create SDK projects for this environment.

Advanced Print Routing (APR)

As part of the Kinetic Smart Client Sunset in May 2026, the following products/modules are deprecated:

- Information Worker
- Epicor Handheld
- Classic Configurator
- Inspection Configurator
- EWA (Epicor Work Anywhere)

You are encouraged to transition to cloud-first alternatives, and we recommend reaching out to your customer account manager for more information.

You can find all the information related to the Smart Client Sunset on our EpicWeb page here: Kinetic Smart Client Sunset

Each Kinetic report has one or multiple report styles that define its type (such as SQL Server Reporting Services (SSRS) and Outbound EDI), its report data definition, and the specific companies which can run the report style. If you are developing an SSRS report style, you can also design a routing rule for it. You do this by using the Advanced Print Routing (APR) toolset. This toolset is available in **Kinetic Power Tools**. When you install this smart client application on your local machine, you can access APR by launching the Kinetic Power Tools icon on your desktop.

To use the Breaking and Routing Rules Designer or create a new routing rule, you must acquire the **Advance Printing** license.

A routing rule determines how the report output generates, prints, and distributes. Routing rules help you streamline reporting for specific business needs. They can be simple rules that define an alternate report style users run when they need to print a report using a unique format, or complex rules that divide, or break, the report run into multiple dataset partitions which they can link to separate rendering workflows for generating, printing, previewing, and sending the report output. Even though the report and data generation process only runs once, you can render multiple reports from this data based on audiences, business requirements, and other factors.

For example, you do business with customers in several countries. You can create a routing rule for an AR Invoice report style that prints different invoices formats. When users run this report style, the routing rule generates different invoice documents based on each customer's country. By further leveraging the Bill-To Address on each customer record, you could define more routing on this rule so each AR Invoice report is sent to each customer as an email attachment. To indicate the purpose

of the email, you could use the template to build the name of the report and then publish this file name in the Subject field.

When you need to print reports, you can set up a rule condition to send each rendered report to a specific client or server printer. Through this feature, you can make one version of the report print for a specific department, and then cause the other version to print in a different department. You can then distribute information more efficiently across your organization.

By designing routing rules that address various needs unique to your company and organization, you will generate targeted documents for customers, suppliers, and internal users. Routing rules significantly improve how reports communicate database information.

Components

Routing rules leverage the following programs and report items. You will use these components to set up the routing rule you need for each report style.

- **Report Data Definition Maintenance** -- Use this program to define from which tables the current report pulls its data. You select the tables, define how they are joined, exclude columns, and add calculated fields on each report data definition.
- **SSRS RDL Definition** -- Use the .rdl file to design the report layout. You open this file in an SSRS report designer like Microsoft® SQL Server Report Builder® to create the layout for each custom .rdl file.
- **Report Style Maintenance** --This maintenance program defines the styles available for each report. If the report style uses the SQL Server Reporting type, you can design a routing rule that defines how report output is rendered through this style.
- **Routing Rules** -- Each SSRS report style can have one routing rule. Through the routing rule elements, you can create different data partitions based on a breaking table, conditions, and filters. Each data partition can then be rendered through an alternate report style. For the report output, you can render these reports as an electronic file, send them to a client or server printer, attach the reports to an e-mail, and display them in a print preview window.
- **Report User Interface (UI) Form** -- Each report window has a **Routing** check box. When a selected report style has a routing rule, this Routing check box is active (selected). Users can shut off the routing rule by clearing (de-selecting) this Routing check box.
- **Business Process Management (BPM) Auto Printing** -- You can set up a method directive to automatically run a report based on conditions you define. When the system detects these directive conditions are met, the report activates. The report renders using the output options you set up on the routing rule.

Rule Elements

The installation contains these routing rule elements:

- **Breaks** -- This element determines which table and columns are used to divide, or break, the data output into separate dataset partitions. You can then apply other elements, such as Filters and Conditions, to render these dataset partitions through different report outputs.
- **Filter** -- Use this report action to evaluate break columns based on criteria you define.
- **Alternate Report Style** -- Through this report action, you define the report style used to render the report. This alternate report style is an existing style available for the current SSRS report, and it can be linked to a custom .rdl file.
- **User Action** -- Through this routing action, the report prints or displays based on what output the user selects on the report window.
- **Conditions** -- Through this element, you create conditions that evaluate incoming data partitions. These conditions determine how the rule renders different reports and distributes the report output.
- **Group By** -- Use this report action to organize, or group, related reports based on criteria you define.
- **Generate** -- This routing action generates an electronic report file. This file is then available for display within the System Monitor.
- **Print** -- Use this routing action to send the report to a specific client or server printer. The selected printer then prints a hard copy of the report output.
- **Print Preview** -- Through this routing action, you generate a report preview that displays for the user.
- **Send E-mail** -- Use this routing action to create an e-mail template for the report. The report file is then sent as an attachment with this e-mail.

Execution Flow

When a user submits a report that contains a routing rule, it renders the output through the following process. This report can either be submitted manually by the user or automatically when a user enters data that triggers an Auto Print Business Process Management (BPM) directive.

1. The system uses the report data definition specified on the report style to pull data from the database. This generates the report dataset; this dataset matches the schema in the report data definition.
2. If a Break element exists on the rule, the break columns are evaluated against the report dataset to divide this dataset into separate partitions. A dataset partition is created for each unique combination of the break columns.
3. Each dataset partition is processed independently, and if any sort order is defined on the break, this sort order determines the sequence through which these dataset partitions process.
4. Now filters and conditions are applied against each dataset partition. The filters restrict which dataset partitions are send to the report output, while conditions route the database partitions to different rendering outputs.

5. Optionally an alternate report style can now receive the dataset partition to change the .rdl file used to render the report output. Otherwise the .rdl selected on the report style is used by default. The sort order defined on the .rdl file determines the column sequence through which the data displays on the report.
6. Lastly, a routing action determines the final rendered output for the report. You can create these routing workflows through any design that makes sense, although some elements, such as the Break element, must be placed as the first element after the Start element.
7. The system creates the resulting documents as a generated file, printed document, print preview, or an e-mail attachment.

Using the Breaking and Routing Rule Designer

The Breaking and Routing Rule Designer is the workflow design tool for creating report routing rules. You launch this program from Report Style Maintenance.

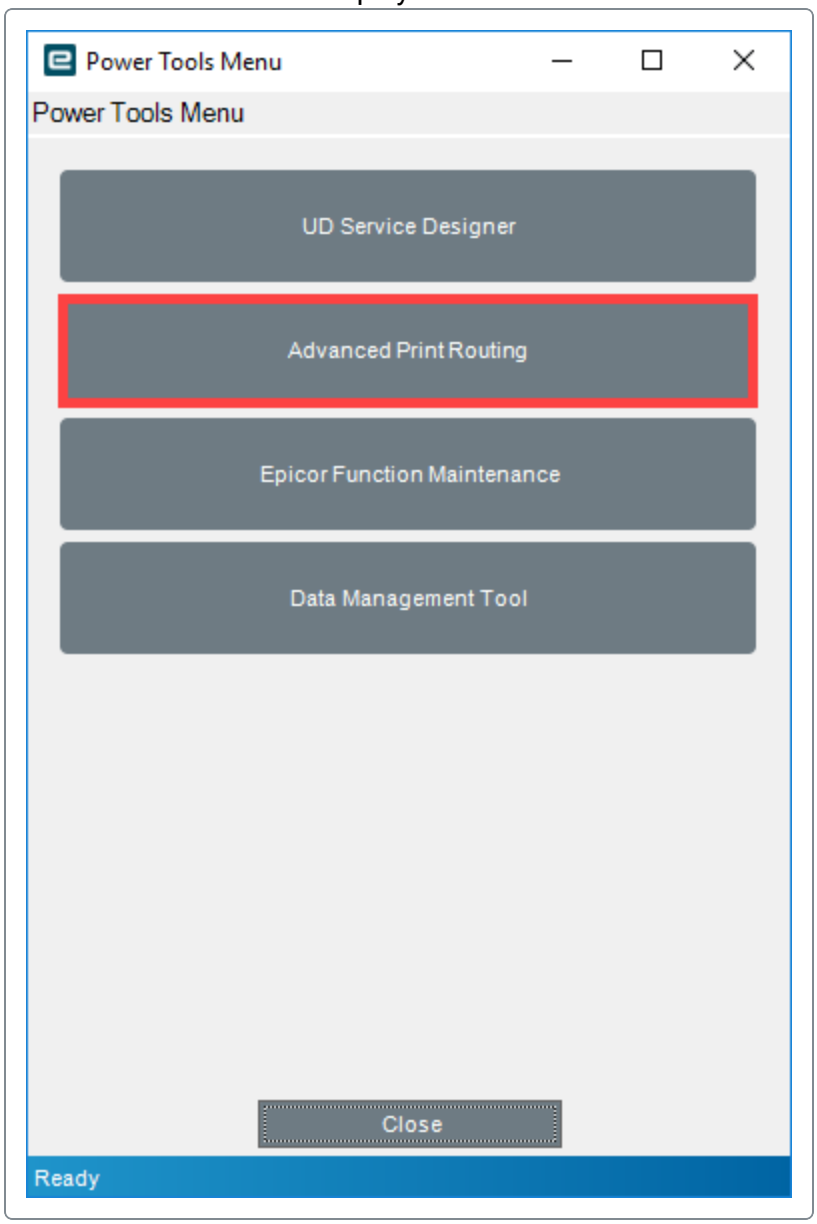
To add a routing rule to a report style, it must be a SQL Server Reporting (SSRS) style that uses the Database Output Location. Each report style can only have one routing rule, but each rule can be set up with a number of routing outputs you define using the Breaking and Routing Rule Designer.

Designing the Routing Rule

This section describes how you create a routing rule within the Breaking and Routing Rule Designer. It is intended as an overview; details about each routing rule element are described later in this area of the application help.

1. Launch the **Kinetic Power Tools** client.
2. Log in with your Kinetic user account.

3. The **Power Tools Menu** displays. Press the **Advanced Print Routing** button.



> If this button IS NOT active, your system administrator needs to activate the **Advance Printing** license for your Kinetic environment. You might also need your user account to have **SSRS Report Designer** rights.

4. **Report Style Maintenance** launches. On the **Detail** sheet, either find and select a report you wish to modify.

5. Now either create or select an SSRS report style.

6. Navigate to the **Styles > Breaking/Routing Rule** sheet. This style must have the following values:
    - **Report Type** - SQL Server Reporting (SSRS)
    - **Output Location** - Database
7. Click **New > Breaking and Routing Rule**.
8. Navigate to the **Breaking/Routing Rule** sheet.
9. Select the **Break Table** you will use for the routing rule. Although you do not need to use this table on the routing rule, you must select a break table before you access the Breaking and Routing Rule Designer. This ensures a break table is selected if you need it.

    If you will use a break table on your routing rule, be sure you select a primary table from the report data definition. You then will generate the report output you need.

10. Click the **Design...** button.
11. The **Breaking and Routing Rule Designer** displays.
12. Notice a number of routing elements display on the left side of this workflow designer. Click and drag an element onto the workflow pane.

    Now click the **Start** element.
13. A series of arrows display around this element. Click and drag a connection line from the Start element to the element you added to the workflow. A line displays, indicating the direction in which the routing moves through the workflow.
14. Select the element you added.
15. A workflow statement displays at the bottom of this window. Depending the selected element, some statements will have links you need to configure (such as the Print and Send E-mail elements), while others do not need additional configuration (such as the Print Preview and Generate elements). If the statement needs configuration, click a link.
16. A selection window displays. Select the value you want on the selection window and then click **OK**.
17. Repeat these steps to design the routing rule that satisfies your business requirement. Eventually your routing rule should end with one or a series of Routing Actions that will cause the report to print to a selected printer, display a print preview, generate an electronic file, or be sent as an e-mail attachment.
18. When you finish, click the **Validate** button to check if the routing rule is correct. When you finish designing the routing rule, click the **Save and Exit** button.
19. You return to Report Style Maintenance.

20. On the Styles > Breaking/Routing Rule sheet, select the **Enabled** check box.
21. Click **Save**.

Now when users launch the report window and select this SSRS style, the **Routing** check box activates on the report window. This indicates the style has a routing rule. When users generate or print this report, it can use this rule to determine how the report generates, prints, and distributes.

However if users just want to preview the report or print it out through other options, they can clear (de-select) the Routing check box. This shuts off the routing rule, and the users can leverage the options on the report to determine the report output.

Adding Breaks to Routing Rules

If you will generate multiple report outputs through your report rule, you must add a Break element to it. This element defines the table from the report data definition used to divide the report dataset into dataset partitions.

The break table you select should be one of the main parent tables used by the report data definition. If you select a secondary child table from the data definition, you may receive unexpected results. In some cases selecting a child table can cause the same form to generate as separate output multiple times. As you plan the routing rule, review the report structure in Report Data Definition Maintenance to decide which table you will select later to break the report dataset.

Before you launch the Breaking and Routing Rule Designer, you select the Break Table within Report Style Maintenance. Then when you add the Break element to your routing rule, you select columns from this break table to determine the unique combinations through which the routing rule generates dataset partitions. The routing rule evaluates the various filters and conditions connected to the Break element to determine the dataset partitions used for the available workflow outputs.

You can also use the Break element to define the sort order, or sequence, through which the dataset partitions process. For example, you need to set up a sort order to define the sequence through which the rendered reports are sent to a printer. Typically you do not need to set up a sort order, as sending e-mail and generating electronic files do not need to process through a sequential sort order. However if you need the routing rule to use a specific sequence while it gathers the dataset partitions, be sure to define the sort order. Through this feature, data you need to evaluate first is pulled before subsequent data is gathered by the routing rule.

Please note that even though you select a Break Table before you can launch the Breaking and Routing Rule Designer, you do not need to add a Break element to every rule. This table is required so that the workflow designer knows which table to use for the Break element. However you only select this element when you need the rule to generate multiple report outputs. When your workflow does not have a Break element, you will set up a routing rule that only builds a single report.

Adding a Break Element

If your routing rule will create multiple outputs, your rule must start with a Break element. You use this element to select the breaking columns that determine the dataset partitions for each rendered report.

Element Features:

- If the routing rule will generate multiple outputs, the Break element is always the first element for the rule.
- It can only be attached to the **Start** element.
- A routing rule can only have one Break element.
- If the rule does not have a Break element, you cannot use the **Filter** and **Group** elements.

To use the Break element:

1. Click and drag the **Break** element to the workflow pane.
2. Select the **Start** element.
3. Click and drag a connection line between the Start element to the Break element.
4. Select the Break element. The following **Actions** statement displays:

   - Break the report using specified columns and sorting

   Click the **specified** link.
5. The **Break and Sort** window displays.
6. On the **Break Columns** sheet, use the **Available Columns** list to select the columns you wish to use for breaking. To help you locate the columns you need:

   a. Click the **Alphabetize** button. The columns then display in alphabetical order.

   b. Enter a starting value in the **Filter** field. Only columns that match this starting value display in the Available Columns list.

   Select the columns. To do this:

   a. Highlight the specific column you want. Click the **Right Arrow** button.

   b. To select all the columns, click the **Double Right Arrow** button.
7. The columns display in the **Selected Break Columns** list.
8. Use the **Up** and **Down** buttons to adjust the order in which the columns display on the list.
9. To remove columns from the **Selected Break Columns** list:

   a. Highlight the specific column you want. Click the **Left Arrow** button.

   b. To remove all the columns, click the **Double Left Arrow** button.
10. You can also use the Break element to indicate the sequence through which you want the rendered reports to process. Click the **Sort Order** tab.
11. The columns you previously selected display on the **Break Columns** list.
12. The same controls display on this sheet. You can add one column or all columns to the **Columns to include in sorting** list by using the **Right Arrow** and **Double Right Arrow** buttons.
13. Use the **Up** and **Down** buttons to adjust the sort order.
14. Likewise you can remove a selected column or all columns from the Columns to include in sorting list by using the **Left Arrow** and Double **Left Arrow** buttons.

15. When you finish selecting the break columns and selecting the sort order, click **OK**.
16. You return to the Breaking and Routing Rule Designer.

You have defined the break columns and the sort order for the routing rule.

Break Columns Fields

This list details the fields that display in the Break Columns section:

- **Alphabetize** - Press this button to sort the Available Columns list in alphabetical order. You can then more easily find the columns on the break table.
- **Available Columns** - Displays all the columns within the selected break table. You select the columns from this list that you want to use for breaking the report output into different dataset partitions, or chunks. You can select specific columns by clicking the **Right Arrow** button or all columns by selecting the **Double Right Arrow** button.
- **Filter** - Enter a starting value in this field to limit the columns that display in the Available Columns list. Only columns that start with the same characters now display on the Available Columns list.
- **Selected Break Columns** - Displays all the columns you have selected to create dataset partitions on the current rule. You can remove selected columns by clicking the **Left Arrow** button; you can remove all the selected columns by clicking the **Double Left Arrow** butto

Sort Order Fields

This list details the fields that display in the Sort Order section:

- **Break Columns** - Displays the columns you selected on the Break Columns sheet. You can use these columns to determine the sort order. You can select specific columns by clicking the **Right Arrow** button or all columns by selecting the **Double Right Arrow** button.
- **Columns to include in sorting** - Displays the columns you have selected to define the sort order on the routing rule. If you need to remove a specific column, click the **Left Arrow** button. To remove all columns, click the **Double Left Arrow** button.

Adding Flowchart Actions to a Routing Rule

Flowchart actions determine how the routing rule evaluates data before it passes along to other elements in the rule. You add the Condition element to your workflow to set up these data evaluations.

Condition Elements

You add one or multiple Condition elements to establish criteria which determines whether a routing workflow will run.

Condition elements create multiple route workflows on the rule. A Condition element evaluates each dataset partition defined in the Break element. You define statements within Condition elements that evaluate data that either comes from the break table or any table within the report data definition related to the break table.

You can set one or multiple conditional statements within each Condition element. To define the logical sequence through which multiple statements evaluate, you use **And/Or** operators and **Prefix/Postfix** parentheses. Through these features, you can create both simple and complex conditions for the routing rule.

When the data matches the statements you defined within the Condition element, it evaluates to **True** and the dataset partition then flows to the True side of the element. Otherwise the dataset partition flows to the **False** side of the Condition element. If any workflows are connected to the True or False sides, the elements within these workflows activate.

Adding a Condition

You add Condition elements to evaluate the data that populates the breaking columns. The Condition element either resolves to True or False, and you then create further routing actions based on either result.

Element Features:

- The Condition element reviews data in columns from the breaking table or a table related to the breaking table.
- You can attach Condition elements to **Break**, **Filter**, and **Group By** elements.
- Each Condition can contain one or multiple condition statements.
- Use **Operators**, **Prefixes**, and **Postfixes** to determine the order through which the condition statements resolve.
- Condition elements either resolve to **True** or **False**.

To use the Condition element:

1. Click and drag a **Condition** element onto the workflow pane.
2. Now connect this element to a preceding element, such as a Break element, in the rule workflow. This indicates data that comes from the preceding element flows into the Condition element.
3. Select the Condition element. The Condition table displays at the bottom of the Breaking and Routing Rule Designer.
4. To add a condition click the **New** button.
5. Click the **Statement** drop-down list to select the statement you wish to use. A condition statement displays within the Condition table. Notice the statement contains a series of links.

   For details on each statement, review the Condition Statements section later in this help article.

6. Click on a link to display a selection window. Use this window to select a value you wish to evaluate through the statement.
7. Continue to click other links to select other values for the condition statement.
8. If you need, add more statements to this Condition element. To do this, click the New button, select the statement, and define the values you need on the statement.
9. Use the **Up** or **Down** buttons to define the sequence in which the statements display.
10. If the current statement and the preceding statement must evaluate to True, click the **Operator** drop-down list and select the **And** option. If either the current statement or the preceding statement can evaluate to True, click the **Operator** drop-down list and select the **Or** option.
11. To indicate two or more statements must evaluate together, combine them by entering parentheses in the **Prefix** and **Postfix** columns.

When you finish adding and defining the statements you need, you can then attach actions to either the True or False links on the Condition element.

Condition Statements

You can add these statements to each Condition element. You can add as many statements as you need to satisfy the requirements needed at this workflow point in your routing rule.

- The **specified** report field on **any row is equal to** the **specified** value

Use this statement to evaluate data from a selected column against a specific value you define.

Link	Selection Interface	Options
specified	Select Table Field	• Select a Table from the report data definition

report		• Select a Field from the table
any row	Drop-Down List	• Any row • All rows • First row • Last row
is equal to	Drop-Down List	• is equal to • is not equal to • is less than • is less than or equal to • is more than • is not more than • begin with • end with • contains • matches
specified value	Specify a Value	Enter a value based on the selected report field type. Depending on the selected field, you will enter dates, numeric values, alphanumeric values, and so on.

- The **specified** report field on **any rowis equal to** the **specified** System Constant

This statement evaluates data from a selected column against a system constant you define.

You can only select one field on each statement.

Link	Selection Interface	Options
specified report	Select Table Field	• Select a Table from the report data definition • Select a Field from the table
any row	Drop-Down List	• Any row • All rows • First row • Last row
is equal to	Drop-Down List	• is equal to • is not equal to • is less than • is not less than

		• is more than • is not more than • begin with • end with • contains • matches
specified system constant	Select a System Constant	The system constants you can select on this window depend on the report field. For example, if this statement is evaluating a date field, you can select system constants like **FirstDayOfWeek**, **Yesterday**, **FirstDayOfNextMonth**, and so on. If you select an alphanumeric field, you select system constants like **CompanyName**, **CountryCode**, and **Product Code**.

The report is **submitted** by the **specified user**

Use this statement to determine whether a specific user did or did not send the report to the server.

Link	Selection Interface	Options
submitted	Drop-Down List	• submitted • not submitted
specified user	User Account Search	Use this selection window to find and select a specific user.

The report is **submitted** by a user who belongs to the **specified** User Group

Through this statement, you determine whether the report was sent to the server by a member of a selected security group.

Link	Selection Interface	Options
submitted	Drop-Down List	• submitted • not submitted
specified User Group	Security GroupSearch	Use this selection window to find and select a specific security group.

Adding Report Actions to a Routing Rule

Report Actions determine the main configurations for the report output. Use these elements to select alternate report styles, filter data from break tables, and group reports together through a unifying value.

Alternate Report Style

Use the Alternate Report Style element to generate or print a report output through a custom .rdl report layout. Use this functionality when you need to render variations on each report based on customer group, country, state, or other requirements.

This flexible element can be attached to several routing rule elements. Any elements which follow this element in the workflow then render the report output using the .rdl file you specify in the Alternate Report Style element. For example, if a Print report action element follows an Alternate Report Style element, the hard copy prints using the format designed the selected .rdl file.

Example:You do business with customers in Poland and the Czech Republic, and these countries require different formats on AR invoices. To address this situation, you would first create an AR invoice style for each country. You would then set up a routing rule that monitors Country values. When you print AR invoices uses this report style, the routing rule automatically prints one report layout for Poland customers and another layout for Czech Republic customers. Customers who are not in either country receive invoices that use the default report layout.

If the Alternate Report Style element is not used during the report run, the output is rendered using the default .rdl file specified on the report style.

You use Alternate Report Style elements to generate or print report output using a different style from the default report style.

Element Features:

- Can be added after **Start**, **Break**, **Group By**, **Condition**, and **Filter** elements.
- Receives the incoming report dataset. If the incoming dataset is a data chunk pulled from a break, the report output displays this dataset partition.
- The report generates or prints using the style selected on the Alternate Report Style element.

To use the Alternate Report Style element:

1. Click and drag an **Alternate Report Style** element to the workflow pane.
2. Connect this element to a preceding element. This preceding element determines the dataset partition that will render through this alternate style.

   Select the Alternate Report Style element.
3. The following **Action** statement displays:

- Generate the report using the **selected** alternate report style

Click the **selected** link.

4. The **Alternate Report Style Selection** window displays.

   Select the check box next to the alternate report style you wish to use.

5. This window displays a list of the SSRS report styles. Any report style that uses the same report data definition displays on this list, including the current report style. This ensures the same report data is used throughout the routing rule. In some workflow situations, you may need to revert back to the original report style after a previous Alternate Report Style element.

   Click **OK**.

6. You return to the **Breaking and Routing Rule Designer**.

The Alternate Report Style element will now generate or print the report using this alternate style.

Filter

The Filter element evaluates whether a specific dataset partition is used in the current workflow. You can then choose which dataset partitions are rendered as report output.

For example, you design a workflow that generates five unique reports. You only need some of these report versions in this part of the workflow, so you add a filter that restricts the rendered output to three dataset partitions -- which will render as three report versions. The other two report versions are not rendered by the routing rule at this point in the workflow.

This element is similar to the Condition element, as you can add multiple statements to the same Filter element. You can also use Operators, Prefixes, and Postfixes to specify the logical sequence the filter statements use to evaluate the incoming report dataset.

Adding a Filter

You use Filter elements to determine the dataset partitions that flow into a subsequent report output.

Element Features:

- The Filter element reviews data from the breaking columns.
- You can only attach Filter elements to **Break** element.
- Each Filter can contain one or multiple filter statements.
- Use **Operators**, **Prefixes**, and **Postfixes** to determine the order through which the filter statements resolve.
- The filtered dataset is passed on to the report output

To use the Filter element:

1. Click and drag a **Filter** element onto the workflow pane.
2. Now connect the **Break** element to the Filter element.

   Tip:You can only have one Break element on each routing rule. However you can have multiple Filter elements connected to the Break element
3. Select the **Filter** icon.

   The following Filter statement displays. Notice the statement contains a series of links:

   - The **specified** breaking column **is equal** to the **specified** value

   For more details on each link, review the **Filter Statements** section below.
4. Click on a link to display a selection window. Use this window to select a value you wish to evaluate through the statement.
5. Continue to click other links to select other values for the filter statement.
6. If you need, add more statements to this Filter element. To do this, click the New button, select the statement, and define the values you need on the statement.
7. Use the **Up** or **Down** buttons to define the sequence in which the statements display.
8. If the current statement and the preceding statement must evaluate to True, click the **Operator** drop-down list and select the **And** option. If either the current statement or the preceding statement can evaluate to True, click the **Operator** drop-down list and select the **Or** option.
9. To indicate two or more statements must evaluate together, combine them by entering parentheses in the **Prefix** and **Postfix** columns.

When you finish adding filter statements, you can then attach report and routing actions to this Filter element.

Filter Statements

You use the links on each filter statement to define the data you wish to filter. You can add as multiple statements to each Filter element.

The **specified** breaking column **is equal to** the **specified** value

Link	Selection Interface	Options
specified breaking column	Select Table Field	Select a Field from the break table. Only fields from the break table can be evaluated on filter statements.
is equal to	Drop-Down List	• is equal to

		• is not equal to • is less than • is not less than • is more than • is not more than • begin with • end with • contains • matches
specified value	Specify a Value	Enter a value based on the selected breaking column. Depending on the selected field, you will enter dates, numeric values, alphanumeric values, and so on.

Group By

Use the Group By element to gather a group of reports together. The routing rule can then perform actions against a related group of reports.

You group the reports using one or multiple breaking columns. For example, use this element when creating a routing rule where multiple sales order acknowledgments must be e-mailed to one customer. By setting up a Group By element that gathers the reports by customer, you can then generate these sales order reports first by customer and then attach the group of reports to the same e-mail message.

Use a Group By element to gather reports through one or multiple breaking columns.

Element Features:

- You attach Group By elements to **Filter** or **Condition** elements.
- You can Group By one or multiple breaking columns.
- The Group By element causes reports to print together or be attached together on a single e-mail.

To use the Group By element:

1. Click and drag a **Group By** element onto the workflow pane.
2. Now connect the **Filter** or **Condition** element to the Group By element.
3. Select the **Group By** element.

   The following **Action** statement displays:

- Group the report data by the **selected** breaking columns

Click the **selected** link.

4. The **Select Table Field(s)** window displays. Notice only columns you selected on the Break element display in this window.
5. Select one or multiple columns through which you will group the generated report.

   Click **OK.**
6. You return to the **Breaking and Routing Rule Designer**.

The Group By element is now defined on your routing rule. You next add other report action elements and routing action elements to complete the workflow. The reports you generate through this workflow are then grouped together using data from the selected breaking columns.

Take ECM Attachment

Use the Take ECM Attachment action to take an ECM attachment from a record in Kinetic and process it along with the report. For example, you can use this action to retrieve the attachment and send it as an email together with the processed report, or you can send it to printing together with report.

This action is only available when the Break node is used on the canvas. You first need to create a Break node and define break column(s), and then you can add the action for each report.

Element Features:

- can be attached to any element as long as the **Break** element is a part of the canvas

1. Click and drag the **Take ECM Attachment** element onto the workflow pane.
2. Select the **Take ECM Attachment** element. The following Actions statement displays:

- For **selected** records take **specified** ECM attachment using **key fields**/**SysRowID column** with **designed** value

1. Click the **selected** link. The **Select records from the report table** window displays. Use this window to define the conditions when you want to take an attachment.
2. From the **For** drop-down list, select the record in the report table that the system should look through when defining whether to take an attachment. The following options are available:

   - each record
   - first record
   - last record
3. From the **In the report table** drop-down list, select the report table in which the system should look through the specified records when defining whether to take an attachment. The report table can be either a break table, or one of its child tables.

4. Click the **New** button to add more query conditions. You can use the following fields to define the conditions:

    - **And/Or** - Defines the criterion string clause in relation with other criterion on this sheet. Use this field when you need to apply a filter based on the values of more than one criterion.
    - **Prefix** - Use this field to insert a left parenthesis to the clause if needed.
    - **Field Name** - Select a field to use as filtering criterion.
    - **Operation** - Define how the selected field will be evaluated against the **Filter Value**.
    - **Filter Value** - Use this drop-down list to define how this criterion filters data. Each option displays with hyperlink text. Click on this link to define the value you wish to use on this criterion.
    - **Postfix** - Use this field to insert a right parenthesis to the clause if needed.
5. You can also use the **Up** and **Down** arrows at the top to change the order of provided query conditions and use the **Delete** button to remove the conditions you do not need.
6. When you finish, click **OK**. You return to the **Breaking and Routing Rule Designer**.
7. Click the **specified** link to define the record from which you want to take the attachment. The **Select Table Attachment** window displays.

    a. From the **Attachment Table** drop-down list, select the table from which the system should retrieve the attachment. This list contains all tables in Kinetic that support attachments.

    b. Use the **Take** drop-down list to specify the attachment to retrieve. The available options include:

        - all attachments
        - the first attachment
        - the last attachment (default option)

    c. Enter the **Document Type ID** or click the button to select the required ID from the results of the **ECM Document Type Search**.

    d. If necessary, define the optional **Attachment Record Properties**.
8. You can also either click the **Insert** button or right-click each of the fields in this section to display a context menu; this menu has the following options:
    - **Insert Fields** - Select this option to display the **Select Table Field(s)** window. Enter a **Name** to identify the field template on the Attachment properties window. Then select the field(s) that will populate the current field with data; fields from the break table and any tables with relationships defined in the report data definition display. When you finish, click **OK**.

> All Insert Field substitutions take the first row from the current report views. The only exception is the table defined **For selected records**. For this table, the current processing row is taken.

- **Insert System Constant** - Select this option to display the **Select a System Constant** window. Then select the system constant that will populate the current field with data. When you finish, click **OK**.
- **Copy Field Template** - After you have created a field template that populates an attachment template field using either a field or a system constant, you can reuse it in another attachment field. Select this option to copy the template to your clipboard.
- **Paste Field Template** - When you copy a field template, this context menu option activates. Select another attachment template field, right-click, and highlight the Past Field Template option. The copied field template now displays in the field.

9. Select either the **key fields** or the **SysRowID column** link option to define from which table row to take the attachment.

    a. If the SysRowID is available for the record, you can use it to retrieve the attachment. After selecting the **SysRowID** option, click the **designed** value link. You can enter a specific value in the **SysRowID** field, or you can click the **Insert** button, or right-click the field to display a context menu.

    b. If the SysRowID is not available, it is possible to use specific Key fields to identify the record from which you want to take an attachment. After selecting the **key fields** option, click the **designed** value link. You can enter a specific value in the **Key** fields that display or you can click the **Insert** button or right-click the fields to display a context menu. When you finish, click **OK**.

10. When you finish defining the statement properties, optionally, select the **Terminate on Error** check box in the end of the statement to stop processing the report in case of error. If this check box is cleared, the error is added to the system monitor log and the action is not executed, but the report is processed anyway.

11. You return to the **Breaking and Routing Rule Designer**. After you specify all conditions and properties used to identify the attachment you want to process with the report, you need to select the action you want to work for the report and attachment. For example, you can include the Send Email routing action to the canvas to add the attachment to an email together with the processed report.

Adding Routing Actions to a Routing Rule

The routing actions determine the final rendered output of a report workflow. You complete each workflow path by placing one or more of these actions at the end of it.

Add ECM Attachment

Use the Add ECM Attachment action to add the processed report as an ECM attachment to a record in Kinetic. You define the conditions when an attachment should be added to a record, specify the table and row where you want to add the attachment, and when the routing rule activates, the workflow connected to the Add Attachment action adds the report as an ECM attachment to the specified record using provided conditions.

This action is only available when the Break node is used on the canvas. You first need to create a Break node and define break column(s), and then you can add the action for each report.

Before you add the ECM Attachment action, consider the following:

- When a branch of an Advanced Print Routing (APR) workflow includes the Add ECM Attachment action (but no Take ECM Attachment actions), the report file is sent to ECM as an unzipped file (example: PDF) unless it is formatted as EMF, which is a report format that is always zipped. Alternatively, if the APR branch also includes one or more Take ECM Attachment actions, the file sent to ECM will be a zip file with both the report and the additional files taken from ECM, unless you select the Allow Multiple Files checkbox on the Add ECM Attachment action. Once selected, the files are sent individually to ECM.
- The server configuration does not allow for generation of multiple ECM attachments via Advanced Print Routing (APR) at the child record level. For example, you have a Purchase Order report with a single PO Header and multiple PO Lines, and you want set up APR to break the report at the PO Header level, setting the routing rule to add attachments for every PO Line - in this case, only a single report file will be attached to the PO Header (instead of multiple attachments; 1 for each PO Line). The server configuration determines that the same report file cannot be attached multiple times.
- If metadata is configured for the table/document type, verify the corresponding columns configured for the metadata are included in the Report Data Definition. If they are not, no metadata will be added; metadata text boxes will be empty in the ECM site. Use the **Data Sources > Report Table > Exclusions** sheet in the **Report Data Definition** program to determine which fields or labels are used on the report definition.

Element Features:

- can be attached to any element as long as the **Break** element is a part of the canvas

Do the following to add an ECM Attachment element:

1. Click and drag the **Add ECM Attachment** element onto the workflow pane.
2. Select the **Add ECM Attachment** element. The following Actions statement displays:
    - For **selected** records add ECM attachment to **specified** table using **key fields/SysRowID column** with **designed** value
3. Click the **selected** link. The **Select records from the report table** window displays. Use this window to define the conditions when you want to take an attachment.
4. From the **For** drop-down list, select the record in the report table that the system should look through when defining whether to add an attachment. The following options are available:
    - each record
    - first record
    - last record
5. From the **In the report table** drop-down list, select the report table in which the system should look through the specified records when defining whether to add an attachment. The report table can be either a break table, or one of its child tables.
6. Click the **New** button to add more query conditions. You can use the following fields to define the conditions:
    - **And/Or** - Defines the criterion string clause in relation with other criterion on this sheet. Use this field when you need to apply a filter based on the values of more than one criterion.
    - **Prefix** - Use this field to insert a left parenthesis to the clause if needed.
    - **Field Name** - Select a field to use as filtering criterion.
    - **Operation** - Define how the selected field will be evaluated against the **Filter Value**.
    - **Filter Value** - Use this drop-down list to define how this criterion filters data. Each option displays with hyperlink text. Click on this link to define the value you wish to use on this criterion.
    - **Postfix** - Use this field to insert a right parenthesis to the clause if needed.
7. You can also use the **Up** and **Down** arrows at the top to change the order of provided query conditions and use the **Delete** button to remove the conditions you do not need.
8. When you finish, click **OK**. You return to the **Breaking and Routing Rule Designer**.
9. Click the **specified** link to define the record to which you want to add the attachment. The **Attachment Properties** window displays.
    a. From the **Attachment Table** drop-down list, select the table to which you want to add the attachment. This list contains all tables in Kinetic that support attachments.

Use the **Attachment template** section fields to define what the attachment should look like. You can enter specific values in these fields:

- **Document Type ID** - Enter the ID into the text box or click the button to select the required ID from the results of the **ECM Document Type Search**.
- **File Name** - Enter the name of the attached file.
- **Title** - Enter a title for the attachment.

b. You can also either click the **Insert** button or right-click each of the fields in this section to display a context menu; this menu has the following options:

- **Insert Fields** - Select this option to display the **Select Table Field(s)** window. Enter a **Name** to identify the field template on the Attachment properties window. Then select the field(s) that will populate the current field with data; fields from the break table and any tables with relationships defined in the report data definition display. When you finish, click **OK**.

> All Insert Field substitutions take the first row from the current report views. The only exception is the table defined **For selected records**. For this table, the current processing row is taken.

- **Insert System Constant** - Select this option to display the **Select a System Constant** window. Then select the system constant that will populate the current field with data. When you finish, click **OK**.
- **Copy Field Template** - After you have created a field template that populates an attachment template field using either a field or a system constant, you can reuse it in another attachment field. Select this option to copy the template to your clipboard.
- **Paste Field Template** - When you copy a field template, this context menu option activates. Select another attachment template field, right-click, and highlight the Past Field Template option. The copied field template now displays in the field.

c. Use the **Allow multiple attachments** check box to specify whether you want to send multiple files to ECM.

> If the APR branch also includes one or more Take ECM Attachment actions, the file sent to ECM will be a zip file with both the report and the additional files taken from ECM, unless you select the Allow Multiple Attachments checkbox. Once selected, the files are sent individually to ECM.

d. Use the **Metadata template** fields to specify the **Comment** and **Status** that will be written in the metadata for attached record. The system adds metadata values to the record in the ECM repository. You can enter a specific value in these fields. You can also either click the **Insert** button or right-click each field to display a context menu.

e. When you finish, click **OK**. You return to the **Breaking and Routing Rule Designer**.

10. Select either the **key fields** or the **SysRowID column** link option to define to which table row to add the attachment.

    a. If the SysRowID is available for the record, you can use it to add the attachment. After selecting the **SysRowID** option, click the **designed** value link. You can enter a specific value in the **SysRowID** field, or you can click the **Insert** button, or right-click the field to display a context menu.

    b. If the SysRowID is not available, it is possible to use specific Key fields to identify the record to which you want to add an attachment. After selecting the **key fields** option, click the **designed** value link. You can enter a specific value in the **Key** fields that display or you can click the **Insert** button or right-click the fields to display a context menu. When you finish, click **OK**.

    c. You return to the **Breaking and Routing Rule Designer**.

11. When you finish defining the statement properties, optionally, select the **Terminate on Error** check box in the end of the statement to stop processing the report in case of error. If this check box is cleared, the error is added to the system monitor log and the action is not executed, but the report is processed anyway.

Generate

Use this action to indicate the report generates as an electronic file.

When the routing rule activates, the workflow connected to the Generate action pulls the selected report dataset. The rule then builds the report using the report layout defined on the .rdl file. Users can then access the electronic file from the **System Monitor**; the report displays on the **Reports** tab. Users can both display this report and save it to a file location they need.

You add a Generate element at the end of a routing workflow.

Element Features:

- Creates electronic file.
- Can be connected to **Start**, **Break**, **Condition**, **Alternate Report Style**, **Filter**, and **Group By** elements.

To use the Generate element:

1. Click and drag a **Generate** element onto your workflow.
2. Connect this element to a preceding element. This preceding element determines which dataset partition renders within the electronic file.
3. Select the Generate icon.

The following Action statement displays:

- Generate the report

4. When you finish defining the statement properties, optionally, select the **Terminate on Error** check box in the end of the statement to stop processing the report in case of error on Print, ECM, E-mail, or Generate actions.

Print

Use this action to send the report output from a route workflow to a client or server printer. You also define the printer settings the report will use.

You first determine whether the report output prints to a client or server printer. You then launch the print settings window for either a server printer or the client printer. Use this window to select the client or server printer. This printer and the settings are saved with the routing workflow; the report prints using the settings you defined on the Print element.

You add a Print element at the end of a routing workflow.

Element Features:

- Sends the report output to a client or network printer.
- Prints a hard copy of the report.
- Can be connected to **Start**, **Break**, **Condition**, **Alternate Report Style**, **Filter**, and **Group By** elements.

To use the Print element:

1. Click and drag a **Print** element onto your workflow.
2. Connect this element to a preceding element. The preceding elements determine what dataset partition is rendered on the report and whether this report is grouped with other reports.
3. Select the Print icon. The following Action statement displays:

   - Print the report using the **Network Printer** with the **specified** printer settings

4. Click the arrow next to the Network Printer link; select either the **Client Printer** or **Network Printer** option.
5. Now click the **specified** link. Either the SSRS Client Printer or SSRS Network Printer settings window displays:

   a. **SSRS Client Printer Settings** -- Select the client printer you will use for this routing action. Define other settings on this window and click **OK**.

b. **SSRS Network Printer Settings** -- Select the network printer you will use for this routing action. Define other settings on this window. Note you cannot access the Fax or Email tabs; you can only send the report to the selected network printer. When you finish defining the settings, click **OK**.

6. When you finish defining the statement properties, optionally, select the **Terminate on Error** check box in the end of the statement to stop processing the report in case of error on Print, ECM, E-mail, or Generate actions.

Print Preview

When you add this action to a routing workflow, the print output displays on the user's screen as a print preview. The user can then decide whether to save the report file, print out the report, or close the preview window.

Element Features:

- Displays the report output in a preview window.
- Can be connected to **Start**, **Break**, **Condition**, **Alternate Report Style**, **Filter**, and **Group By** elements.

To use the Print Preview element:

1. Click and drag a **Print Preview** element onto your workflow.
2. Connect this element to a preceding element. This preceding element determines what dataset partition will render for display on the print preview window.
3. Select the Print Preview icon.

   The following Action statement displays:

   - Print Preview the report

Send E-Mail

When you add this routing action to a workflow, the report output is first generated as an electronic file and then attached to an e-mail. The routing rule then sends this e-mail to the users you defined on the e-mail template.

You need to set up the e-mail template that will be used to send the report(s). When the routing rule activates, it builds components of the e-mail like the To, From, CC, and Body fields using options you defined on the e-mail template. Likewise, you can also determine the file name for the report attachment.

Element Features:

- Generates the report output as one or multiple electronic files.
- Attaches the report file(s) to an e-mail message.
- Can be connected to **Start**, **Break**, **Condition**, **Alternate Report Style**, **Filter**, and **Group By** elements.

To use the Send E-mail element:

1. Click and drag a **Send E-mail** element onto your workflow.
2. Connect this element to a preceding element. The preceding elements determine what dataset partition is rendered as the e-mail attachment and whether this report is grouped with other reports on the same e-mail.
3. Select the Send E-mail icon.

   The following Action statement displays:

   - Send e-mail based on the designed template.

   Click on the **designed** link.
4. The **E-mail template** window displays.
5. Enter the **Name** for the e-mail template. This value helps identify the purpose of the template.
6. Now define how the template will populate the **From**, **To**, **CC**, **BCC**, **Subject**, and **Attachment Name** fields. You can enter a specific value in these fields. You can also either click the **Insert** button or right-click each field to display a context menu; this menu has the following options:

   a. **Insert Fields** -- Select this option to display the **Select Table Field(s)** window. Enter a **Name** to identify the field template on the e-mail template window. Then select the field (s) that will populate the current field with data; fields from the break table and any tables with relationships defined in the report data definition display. When you finish, click **OK**.

      For example, if you are setting up a template for the To, CC or BCC field, you most likely will select a field that contains e-mail addresses.

      For the **To**, **CC**, or **BCC** fields note that when this field comes from the break table, only one record will populate this field per report partition. For a child table of the break table, select **All** from the **Records Select** drop-down list in the E-mail Template dialog to construct a multi e-mail address based on all rows from the chosen table.

      Example:You can add the Customer Contact table to the Report Data Definition and set up APR action to email invoices to all customer contacts on the selected table.

      To further filter the records, for example, customer contacts that qualify to receive an e-mail for a specific report, use the filtering option available in the E-mail Template dialog. To filter contacts by a specific field, select your desired field from the **Filter** drop-down list. You'll then have the option to choose the **Value** for that specific field.

b. **Insert System Constant** -- Select this option to display the **Select a System Constant** window. Then select the system constant that will populate the current field with data; these options include Today, First Day of the Week, Country Code, Current Time, and so on. When you finish, click **OK**.

c. **Copy Field Template** -- After you have created a field template that populates an e-mail field using either a field or a system constant, you can reuse it in another e-mail field. Select this option to copy the template to your clipboard.

d. **Paste Field Template** -- When you copy a field template, this context menu option activates. Select another e-mail template field, right-click, and highlight the Past Field Template option. The copied field template now displays in the field.

Use the **E-Mail Format** drop-down field to select the format of the message - **Plain Text** (default) or **HTML**.

7. This will define the content type of the e-mail generated by Advanced Print Routing.

   In the **Body** field, enter the text of your message.

8. You can also use the insert options by clicking the **Insert** button or by right-clicking the field to display the context menu.

   When you finish, click **OK**.

9. You return to the **Breaking and Routing Rule Designer**.

10. When you finish defining the statement properties, optionally, select the **Terminate on Error** check box in the end of the statement to stop processing the report in case of error on Print, ECM, E-mail, or Generate actions.

User Action

The User Action element indicates the routing rule will run the report using the action the user selected on the report window.

Users can select one of the following actions on each report window:

- Send to Client Printer
- Send to Server Printer
- Display a Print Preview
- Generate the Report (electronic file)

After the routing rule finishes evaluating the report dataset and filtering the results, the report run begins. The routing rule uses the action selected by the user to determine how the report will display.

If the routing rule is active for the style, users cannot send the report as an e-mail attachment or a fax. If users want to send the report electronically, they either need to

disable the routing rule on the report window (clear the Routing check box) or design a routing rule that sends the report as an e-mail attachment.

You add a User Action element at the end of a routing workflow.

Element Features:

- Indicates the report window action selected by the user will run.
- Can be attached to **Start**, **Break**, **Condition**, **Alternate Report Style**, **Filter**, and **Group By** elements.

To add the User Action element:

1. Click and drag a **User Action** element onto your workflow.
2. Connect this element to a preceding element. This preceding element determines what dataset partition is rendered in the user selected report output.
3. Select the User Action icon.

   The following Action statement displays:

   - Execute the Client Print, Server Print, Print Preview or Generate action selected by the user when submitting the report.

Epicor Functions

Use **Epicor Functions Maintenance** to create custom functions that run in Kinetic. You launch this application from the **Kinetic Power Tools** menu.

Kinetic uses services to control data transactions. A service is a type of data Kinetic manages, such as customer, part, or sales order. Each service contains methods that perform a specific task, such as get new, update, and delete.

You can create Epicor Functions that have custom, reusable logic that you apply to services to change how they run. Run this logic both externally and internally. Do this by first creating a library and then creating functions within this library where you define your own methods. Use them to indicate exactly which part of Kinetic you want to interact with such as services, database tables, or other functions.

Functions work because you create a new service that runs using custom methods (functions). Just like Business Process Management (BPM) directives, functions can call into the application server logic or database tables. You can then validate, manipulate, or run actions based on the data that passes through the function using parameters you define. Because they are service side customizations, you can reuse functions across clients, call them from BPM directives, or call them from other Epicor functions.

Because functions exist outside the base services installed with Kinetic, they update more easily with each Kinetic version.

During this help article, we explore:

- Assigning Functions Security Groups
- Launching Epicor Functions

Assigning Functions Security Groups

Kinetic has three pre-defined security groups that control different levels of access to Epicor Functions Maintenance. Assign these security groups to user accounts in **User Account Security Maintenance**.

Functions Administrator

Functions Administrators can maintain existing Epicor Function libraries and edits some library properties. These users CANNOT add new Epicor Function libraries. By default, Security Managers and Global Security Managers are treated as Functions Administrators even if they are not explicitly added to the Functions Administrator security group.

Functions Developer

Functions Developers can create new libraries and edit the existing ones. They can create Widget Functions and design their workflow with the help of the Function Designer.

If a Functions Developer becomes an owner of a library that was previously created by a Functions Power Developer and that has advanced options (like allowed Custom Code Widgets and DB Access from Code) selected, such Functions Developer will not be able to add new or edit existing Widget Functions with Code (if any), but will be able to delete any functions from the library including Widget Functions with Code.

Functions Power Developer

In addition to the rights of Functions Developers, Functions Power Developers also have rights to enable Custom Code Widgets in the Function Designer and Custom Code Functions in a library, as well as allow access to database tables from the custom code added within the Execute Custom Code actions and expressions.

Use the table below to see what operations can be performed by users from the above listed Functions Security Groups depending on the ownership of the library and the security group it is shared with.

A library displays in Read-Only (RO) mode if:

- the current user is not the owner of the library, or is not a member of the Security Group to which the library is shared
- the library WAS NOT created in the current Company
- the library is published

Table 1: Library

Action	Administrator	Developer	Power Developer
Add	No	Yes	Yes
Import	Yes	Yes for non-RO, libraries with Widget Functions only	Yes for non-RO
Export	Yes	Yes for non-RO	Yes for non-RO
Delete	Yes, incl. RO	Yes for non-RO	Yes for non-RO
Map to a Company	Yes, incl. RO	No	No

Action	Administrator	Developer	Power Developer
Enable/Disable	Yes, incl. RO	Yes for non-RO	Yes for non-RO
Change owner or shared with groups	Yes for unpublished libraries in any Company, incl. RO	Yes for non-RO in the Owning Company	Yes for non-RO in the Owning Company
Enable Custom Code Widgets	No	No	Yes for non-RO
Enable Custom Code Functions	No	No	Yes for non-RO
Allow access to database from code	No	No	Yes for non-RO

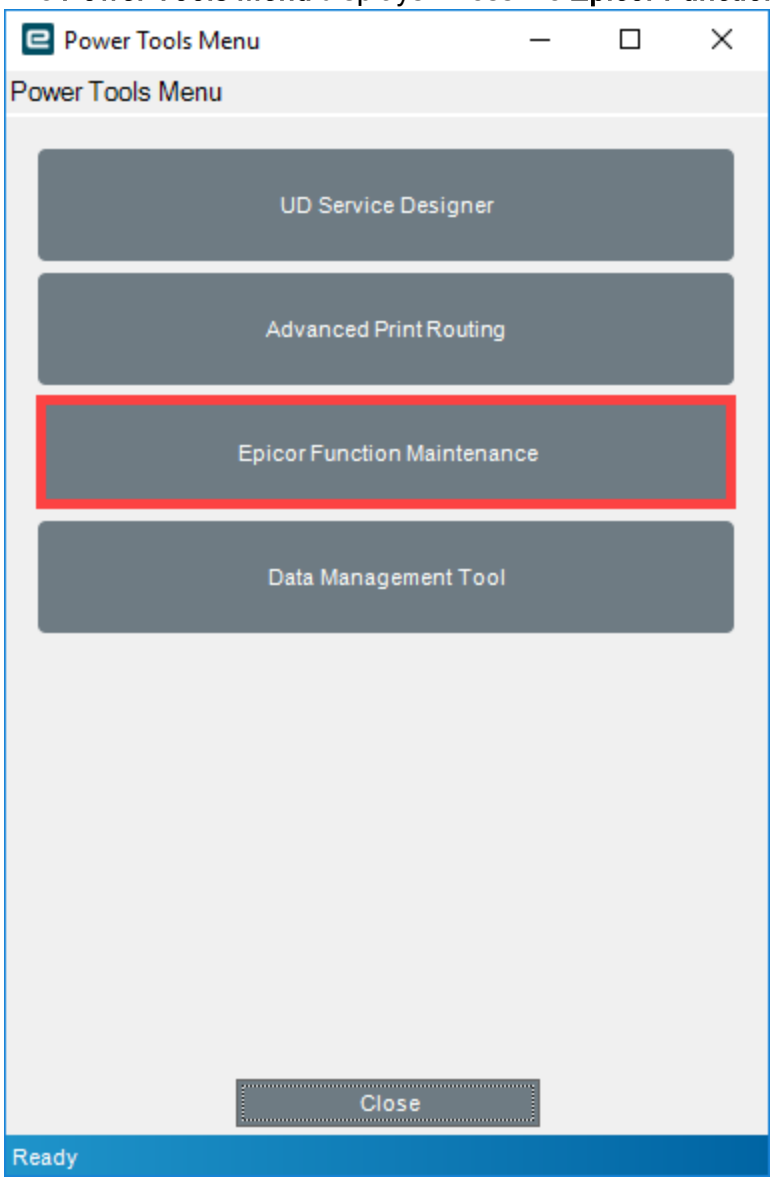
Table 2: Function

Action	Administrator	Developer	Power Developer
Edit function workflow/Run Designer	No	Yes for non-RO Functions that contain custom code widgets or have access to database display in a Read-Only mode.	Yes for non-RO
Add	No	Yes, but Widget Functions only	Yes for non-RO
Copy	No	Yes for non-RO, Widget Functions only	Yes for non-RO
Delete	No	Yes for non-RO	Yes for non-RO
Enable/Disable	Yes, incl. RO	Yes for non-RO	Yes for non-RO
Edit functions' options (Notes, Signature, etc.)	No	Yes for non-RO	Yes for non-RO

Launching Epicor Functions

Epicor Functions is included as part of Kinetic Power Tools.

1. Launch the **Kinetic Power Tools** client.
2. Log in with your Kinetic user account.
3. The **Power Tools Menu** displays. Press the **Epicor Function Maintenance** button.



If this button IS NOT active, your system administrator needs to add one of the Functions security groups to your user account. These security groups are documented in the previous section.

Setting Up Functions in Epicor Functions Maintenance

Before you create a function, you must enter some information that defines the purpose for it. This help article details what values you can enter for the function.

During this help article, we explore:

- Entering Functions
- Adding Signatures
- Entering Notes
- Reviewing the Function Flow

Entering Functions

When you create a function, you enter the following values for it.

Description

This field contains a short general description of the Function.

Design/Edit

Depending on the Function type, this button has different labels and launches different applications:

Button	Function Type	Application
Design	Widget Function or Widget Function with Code	Function Designer
Edit	Custom Code Function	Function Editor

Disabled

Use this button to disable the function. If disabled, the Function will be unavailable to internal and external calls. However you can save a disabled Function even if it is still under development and contains compilation errors that you plan to fix later.

When this option is selected, the Inactive shape displays in the top right corner of the form.

For Internal Use Only

Select this check box to make a specific Function within the library unavailable for external calls via REST.

If a library is marked For Internal Use Only, all its Functions also become internal and can only be called from a BPM directive.

Function ID

This field displays the ID of the Function. Type in the ID of the existing Function to bring up its details or the new ID if you are creating a new one.

You cannot change the ID once the function is saved.

A function ID must be at least one character long while its maximum allowed length is limited by 30 characters. It can contain only letter characters, digits, or dashes. It must begin with a letter and cannot end with a dash. It must not match, but may contain keywords used in C# programming - for example, **new, class, private, public**, etc.).

Example: Private is not allowed, while SuperPrivate can be a Function ID.

Requires Transaction

Select this option if you want your Function to create a transaction. You can use this option in Functions that involve updating the database to ensure the whole workflow rolls back if an error occurs. This will prevent data corruption. You can also use this option when your Function orchestrates several service calls where one or more of these calls depend on the successful completion or failure of the other(s).

Adding Signatures

Use the Signature sheet to manage function parameters.

Function parameters are grouped in two sections:

- Request Parameters
- Response Parameters

While specifying function parameters is optional, most functions typically require them.

Signature fields:

Fields

Fields for the current card are listed in this topic.

Name

Names of request and response parameters must be unique within the group and must be compliant with C# programming syntax. They can contains letters and digits and must start with a letter. The Function parameter name is not valid warning displays if the name fails validation.

Type

This field displays a drop-down list of predefined data types. Users can either select a data type from the list or a TableSet data type from an assembly. Assemblies available for the library are specified on the **References > Assemblies** card of the Epicor Functions Maintenance application.

Complete

When you finish editing function parameters, click this button to switch parameter data to read-only mode.

Edit

When you open an existing function or save a new one, the parameters details display in read-only mode. Use this button to activate parameter grids for editing.

Entering Notes

Use this sheet to add notes to your function.

1. By default, existing notes display in read-only mode. Click **Edit** to edit an existing note or add a new one.

   The Notes editor adds a new line with the current user ID and date in the beginning of your note. The cursor is set in the second line.

2. Enter the text of your note.

3. Click **Commit** to switch the notes back to the read-only mode.

Reviewing the Function Flow

For Widget Functions, the **Overview** sheet displays the preview image of the function workflow.

For Custom Code Functions, this card displays the C# code of the Function in read-only mode. You can use the **Edit** option from the context menu to conduct a quick search, code review, or copy selected fragments.

Defining Functions in Epicor Function Maintenance

Use the **Summary** card to create new or manage existing libraries in Epicor Function Maintenance. Functions are grouped by library. The properties you set up on this card for a particular library will apply to all the functions contained in this library.

Custom Code Functions

This option is available only to users assigned to the **Functions Power Developer** security group. Refer to the **Functions Security Groups** topic for details on security groups related to Epicor Functions.

Select this option to allow adding Custom Code Functions to the library.

Custom Code Widgets

This option is available only to users assigned to the **Functions Power Developer** security group. Refer to the **Functions Security Groups** topic for details on security groups related to Epicor Functions.

Select this option to allow widgets for adding custom code in the **Function Designer - Execute Custom Code** and **Custom Code Condition**.

DB Access from Code

Only members of the Functions Power Developer security group can edit this field. This setting is activated when the Custom Code Widgets or Custom Code Functions option is selected.

It provides three options:

- **None** - Select this option if you do not wish that the expressions and custom code actions from the function workflow had access to the Epicor database.
- **Read Only** - Select this option if you want to enable a read-only access from expressions and custom code workflow actions of your Function to the database tables specified in the library Table References.
- **Read Write** - Select this option if you want to be able to update the database tables that are specified in the library Table References from expressions and custom code workflow actions of your Function.

On the **References > Tables** card, you must additionally mark each table you wish to be able to update as **Updatable**. Please refer to the **References > Tables** topic for details.

Regardless of the option selected in the DB Access from Code field, any query-based actions and conditions in the Function workflow have access to the database tables specified in the library Table References.

Debug Mode

This option indicates the library compilation mode. By default, Epicor Customization Framework generates optimized library assemblies in Release Mode. Select the Debug Mode option to compile the library in **Debug Mode** for improved debugging experience:

- It activates any conditional debugging code you have in your directive or Function.
- It creates a non-optimized assembly and automatically dumps the source files.
- It automatically loads debugging symbol files (.pdb) for the assembly.

Refer to the **Debugging Directive and Function Server-Side Code** topic for details on debugging options provided by the BPM and Functions Frameworks.

Description

This field displays a short general description of the Library.

Use the Notes card for adding specific and detailed description of the Library.

Description (Grid)

This Functions grid column displays the description of the functions that belong to the library.

Disabled

Select this option to disable the library. When a library is disabled, all of its functions are also treated as disabled. Disabled libraries can be browsed by external callers, but their functions cannot be called.

When this option is selected, the Inactive shape displays in the top right corner of the form.

Dump Sources

By default, Functions Framework does not save any source code for compiled libraries. Select this option to dump the source code of the library upon its compilation. The default destination for the Function source code files is <...>\Sources\#Efx\efx.[LibraryID]\[Revision]\.

This option is selected automatically when you select the Debug Mode option.

For Internal Use Only

Select this option to make the library unavailable to external calls via REST.

Internal libraries can only be called from a BPM directive.

For Internal Use Only (Grid)

A selected check box displayed in this **Functions** grid column marks functions that are set as **For Internal Use Only**.

Function ID (Grid)

This **Functions** grid column displays the IDs of the functions belonging to the library.

Use this field to type in the library ID to create a new library or to display the details of an existing one.

A library ID must be at least one character long while its maximum allowed length is limited by 30 characters. It can contain only letter characters, digits, or dashes. It must begin with a letter and cannot end with a dash. It must not match, but may contain keywords used in C# programming - for example, **new, class, private, public,** etc.

Example Class is not allowed, while MyClass can be a library ID.

You cannot change the ID once the library is saved.

Library...

Use this button to search for an existing library via the Library Search form.

Owner (Company)

This second read-only field under the Owner label displays the ID of the Company the library was created in.

Owner (User Name)

This read-only field displays the name of the user who currently owns the library.

Adding Libraries to Epicor Functions

Use the steps in this topic to add a new library.

Only members of the **Functions Developer** or **Functions Power Developer** security groups can add new libraries. Make sure that your user account is assigned to one of these groups.

1. Open the **Epicor Functions Maintenance** application. By default, the **Summary** card displays.
2. In the **Library** field, type in the ID of your new library and press **Tab**. To the message offering to add a new record, click **Yes**.

   You can also add a new library entry form by clicking:

   - **File > New > Add Library** from the Main Menu, or
   - **New > Add Library** from the Main Toolbar.

   A library ID must be at least one character long while its maximum allowed length is limited by 30 characters. It can contain only letter characters, digits, or dashes. It must begin with a letter and cannot end with a dash. It must not match, but may contain keywords used in C# programming - for example, **new**, **class**, **private**, **public**, etc.

   If you enter any restricted character, an error message displays. You cannot change the ID once the library is saved.

3. Enter a short **Description** of your new library.
4. Select the options you need.

   Note: Please remember that these settings will apply to all functions in this library. Library references will be available for use in any Function within this library.

   a. Select the **Custom Code Widgets** option if your library will contain Functions with custom code.

   b. Select the **Custom Code Functions** option if your library will contain pure custom code Function.

   c. In the **DB Access from Code** field, keep or change the default option. Two options are available:

Important: Only members of the **Functions Power Developer** security group can edit this field. This setting is activated when the **Custom Code Widget** option is selected.

Two options are available:

- **None** - Select this option if you do not wish that the expressions and custom code actions from the function workflow had access to the Epicor database.
- **Read Only** - Select this option if you want to enable a read-only access from expressions and custom code workflow actions of your Function to the database tables specified in the library Table References.
- **Read Write** - Select this option if you want to be able to update the database tables that are specified in the library Table References from expressions and custom code workflow actions of your Function.

On the **References > Tables** card, you must additionally mark each table you wish to be able to update as **Updatable**. Please refer to the **References > Tables** topic for details.

Regardless of the option selected in the DB Access from Code field, any query-based actions and conditions in the Function workflow have access to the database tables specified in the library Table References.

d. Select the **For Internal Use Only** option to make the library unavailable to external calls via REST. If you select this option for a library, all of its Functions become internal too. Internal libraries will appear in search results, but their Functions cannot be called via REST.

e. Select the **Disabled** option to disable the library. When a library is disabled, all of its functions are also disabled. Disabled libraries can be viewed by external callers, but their functions cannot be called.

   Note: When this option is selected, the **Inactive** shape displays in the top right corner of the form.

5. Navigate to the **References** card. Click the **Add** button to add necessary assemblies (for example, **Erp.Contracts.BO.ABCCode.dll**), database tables (for example, **ERP.ABCCode**), and ERP services (for example, **ERP:BO:ABCCode**) to your library.

6. Click **Save**.

Entering Library References for Epicor Functions

Use this field to specify library references.

The references to assemblies, database tables and services specified on this card will be available for use within any function in this library.

Only the database tables added to the library as references can be used in:

- Query-based actions - for example, Fill Table by Query or Set by Query.
- Query-based conditions - for example, number of rows in the designed query is more than or equal to 1
- Custom code and expressions if the library's DB Access from Code setting is enabled.

Fields

Add

Use this button to invoke the search form and add items to the list of assemblies, tables, or services referenced from the library.

Clear

Use this button to clear the list of items on the current card (**Assemblies, Tables, or Services**).

Filter

Use this text box to enter criteria for filtering the list of displayed items. The list will display assembly, table or service items that contain the text from this field.

The filtering is used for quick searches only and does not affect availability of library references.

Reference ID

This grid column displays IDs of the references added to the library.

Remove

Use this button to remove selected items from the list of assemblies, tables, or services referenced from the library.

Assemblies

This card contains the list of server assemblies that can be used by library Functions.

Tables

This card contains the list of Epicor database tables that can be used in Functions.

By default, any table you add to your library as a reference has read-only access. Enabling database table updates in a library that allows **Read Write** database access from expressions and custom code widgets is done on a table-by-table basis. Select the **Updatable** option for each table you wish to be able to update from your Functions.

Make sure that all referenced database tables that can be updated from your custom code or expressions are marked as Updatable. All custom update logic for read-only tables will be ignored by the system, and the database will not get updated.

Services

This card contains the list of Epicor services that can be used by library Functions.

You can invoke four types of services from Epicor Functions:

- **Business Object** - Business Objects are services that belong to the **BO** namespace.
- **Simple Service** - This type includes the services that belong to the **Lib** namespace.
- **Report** - The services of this type belong to the Rpt namespace.
- **Process** - Process Services belong to the Proc namespace.

Libraries

This card contains the list of libraries that can be used in this library's Functions.

You add library references if you need to use their Functions in the current library's Functions. For example, you may have a utility Function that performs a simple operation (a calculation or database update). You can reuse its logic across a number of other Functions in the system.

The current library cannot be selected as a library reference. You cannot call a Function from another Function within the same library.

The system also does not allow adding libraries that can contain a direct or nested circular reference to the current library. For example, **Lib 2** references **Lib 1**, and **Lib 3** references **Lib 2**. Adding **Lib 3** as a reference to **Lib 1 (Lib 1 > Lib 3 > Lib 2 > Lib1)** would cause an error.

Defining Security for Epicor Functions

Use this sheet to define rights for accessing each library in Epicor Function Maintenance. You do this by entering values in the following fields:

Authorized Companies

This box contains the list of companies where your library is available. Use the two lower arrows to remove the selected companies from the Authorized Companies list.

Available Companies

This box contains the list of companies you library can be applied to. Use the two upper arrows to move the selected companies into the **Authorized Companies** list on the right.

Change Owner...

This action is available only if the library is not published. Functions Administrators can change ownership of any library they have access to. Functions Developers and Functions Power Developers can change ownership in the library they own or are shared with and in the library's owning company only. Please refer to the **Function Security Groups** topic for more information on user security rights related to Epicor Functions.

Use this button to launch the **User Account Search** form and select the user account that will be the new library owner.

Clear

Use this button to clear the **Shared With Group** field.

Library Owner

This read-only field displays the current owner of the library.

Share With Group

This field displays the ID of the security group the library is shared with. The members of this security group will have rights to edit the library settings.

Use the **Share With Group** button to launch the **Security Group Search** form and select the group of users who will be able to edit the library's properties.

This action is available only if the library is not published. Functions Administrators can edit this field in any library and company they have access to. Functions Developers and Functions Power Developers can edit the **Share With Group** field in a library they own or are shared with in the library's owning company only. Please refer to the

Function Security Groups topic for more information on user security rights related to Epicor Functions.

Entering Notes for Epicor Functions

Use this sheet to add or update notes to the library. You enter the following values:

Commit

Click this button to switch the notes back to the read-only mode.

Edit

By default, existing notes display in read-only mode. Click **Edit** to edit an existing note or add a new one. The Notes editor adds a new line the current user ID and date in the beginning of your note. Enter your note below.

Adding Widget Functions

Follow the steps in this topic to add a new Widget Function to an Epicor Functions library.

1. From the Main Menu of the **Epicor Function Maintenance** application, navigate to **File > New > Add Widget Function** or click **New > Add Widget Function** from the Main Toolbar. The **Function** card displays with an active entry form.
2. Enter the **Function ID**.

   > Please have in mind the function naming requirements described in the **Summary > Fields** topic.

3. Enter a short **Description** of your function.
4. If necessary, select options for your function.
5. Navigate to the **Function > Signature** card if you need to add request and response parameters.
6. On the **Function > Signature** card, click **Edit** to add new parameters. Select a line in the **Request Parameters** or **Response Parameters** section.
7. When you finish the workflow design, save it and exit the Function Designer. The **Function > Overview** card displays a preview of the workflow.
8. Type in the parameter's **Name**. Please make sure the name meets the parameter naming requirements described in the **Function > Signature > Fields** topic.
9. Select the parameter's **Data Type**. You can select one of the predefined types or select a tableset from the available assemblies. To select a tableset:
   a. Click **Select Type** at the end of the **Data Type** drop-down list. The **Select variable type** dialog window displays.
   b. In the **Available assemblies** list, expand the node of the assembly that contains the tableset.
   c. Select the tableset you need and click **OK**.
10. When you are done with adding Function parameters, click **Complete**. Parameter details display in read-only mode.
11. Click **Design** to launch the **Function Designer** application. Use the available widgets to construct the workflow of your function.

    Note: Please refer to the **Function > Function Designer** topics for details on available widgets and a workflow example.

12. Optionally, add **Notes** to your function. Click **Edit** to activate the **Notes** editor. At any point, you can add as many notes as you need.

13. From the Main Menu or from the Main Toolbar, click **Save**.

Adding Widget Functions with Code

Follow these steps to add a new Widget Function with custom code to an Epicor Functions library.

This action is available to members of the **Functions Power Developer** security group in libraries with enabled **Custom Code Widgets**.

Adding a Widget Function with Code is similar to adding a standard Widget Function. The difference is in additional custom code widgets that become available in the **Function Designer**: **Execute Custom Code** as well as **Custom Code Condition**: The custom code... condition is valid. Besides that the **DB Access from Code** option is available only in libraries that can contain Widget Functions With Code.

Please refer to the **Function > Function Designer** topics for details on available widgets and a workflow example.

1. From the Main Menu of the **Epicor Function Maintenance** application, navigate to **File > New > Add Widget Function with Code** or click **New > Add Widget Function ith Code** from the Main Toolbar. The **Function** card displays with an active entry form.
2. Repeat **Steps 2-11** from the **Add Widget Function** topic.
3. In the **Function Designer**, use the widgets that add custom code to your function.
4. Repeat Steps **12-13** from the **Add Widget Function** topic.

Adding a Custom Code Function

Follow these steps to add a new Custom Code Function.

This action is available to members of the **Functions Power Developer** security group in libraries with enabled Custom Code Functions.

1. From the Main Menu of the **Epicor Function Maintenance** application, navigate to **File > New > Add Custom Code Function** or click the **Down Arrow** next to the **New** icon in the Main Toolbar and select **Add Custom Code Function**. The **Function** card displays with an active entry form.
2. Enter the **Function ID**.
3. Enter a short **Description** of your function.
4. If necessary, select additional options for your function.
5. If necessary, define Function parameters on the **Signature** card.
6. Click **Edit** to add the code of your Function. The **Function Editor** displays. Use this free-from C# editor to add or edit your custom code. Please refer to the Function Editor topics for more information on its functionality.
7. Optionally, add **Notes** to your function. Click **Edit** to activate the **Notes** editor. At any point, you can add as many notes as you need.
8. From the Main Menu or from the Main Toolbar, click **Save**.

Using the Function Designer

Use the widgets available in the **Function Designer** to build the workflow of your function. You can access the Function Designer from Epicor Functions Maintenance.

To open the designer, navigate to the **Function** card and click **Design**.

Function Designer is normally compatible with .NET Framework 4.8 but we recommend installing the latest version of .NET 6.0 Runtime to assure the application behaves correctly in all cases.

During this help article, we explore:

What is a Function?

Function creates a custom method for orchestrating the work of ERP business object(s). Designing a Function workflow is similar to designing a BPM Directive workflow. Therefore, the Function Designer inherited its general principles from the BPM Workflow Designer that is used for developing Data and Method Directive workflows.

The topics below highlight the workflow design principles specific to the Function Designer.

Construct Function Workflow

This topic discusses general principles of creating a Function workflow by using the workflow elements.

1. The available items display in the left pane of the Designer. Workflow elements are grouped by categories and except the **Flow Chart** category, they all represent **Actions** the workflow will take.

2. For each workflow element representing **Actions**, you can select the **Terminate on Error** option to halt the workflow execution when it encounters an error.

   For the Raise Exception action, it is mandatory to have this option selected. The system does this by default.

3. The **Condition** block found in the **Flow Chart** category represents a set of conditions the Function workflow may evaluate prior to executing the following workflow element.

4. To use a Function element in the workflow, drag it and drop it in the Function Designer surface.

5. Connect the workflow elements in the logical order they should be executed. When you hover your cursor over each element, the small black triangles surrounding the element represent the available connectors. To connect elements, click your mouse, select any of the connector symbols, drag the line and point it to another element's connector.

6. The initial point of the Function workflow is the **Start** element.

7. The following workflow elements may utilize multiple inbound connectors but only one outbound connector. The exception is the **Condition** block that allows usage of two outbound connectors **True** and **False**, and the **Raise Exception** element that allows no outbound connectors.

   To delete a workflow element, select it and press the **Delete** button on your keyboard. When a workflow element is deleted, its links are automatically removed too.

8. You can use editing buttons in the top toolbar for smother and more efficient workflow design process.

   - **Copy/Paste** - use these buttons to copy and paste workflow elements within the currently open directive or to another directive. Note you can perform the same actions using the **Ctrl+C and Ctrl+V** keyboard shortcuts.

     You can paste copied elements within Function Workflow Designer only (in the same or different directive). You will not be able to paste them to any other Epicor ERP/ICE application or external application.

- **Undo/Redo** - use these buttons to revert or restore your previous design actions with workflow elements and connectors.

  The Undo/Redo function stores up to 20 previous user actions.

9. To set up the behavior of a workflow element representing **Action**, click on it and supply required parameters in the pane below the designer surface.

   For the **Condition** block, build criteria using pre-built condition statements. You can enter more than one condition statement within a single block and use the **And/Or** logical **Operators** to define the relationship between the current statement and the preceding statement.

10. At any stage of a workflow preparation, you can use the **Variables** tab to create custom, Function-specific variables and make them available for actions and conditions within the Function. You can then bind these variables to external method parameters, reference them across available conditions and actions, pass and receive data from BO call actions or to hold intermediate results during Function execution.

    For more information, read the **Function-Level Variables** topic.

11. The **Method Parameters** tab displays the signature for the current Function method. You can review this information at any time during your workflow design process without exiting the Function Designer.

12. The **Use extensions** button turns on/off Extension tables. The **Use extensions** button provides access to Extension tables for all elements in Function Designer except Function query. Extension table exposure in Function query currently is not supported. Extension tables can be used as target tables, as well as expression elements of Function actions and conditions.

    If actions or conditions in the Function are recognized to reference an Extension table (not within expressions), disabling Extension tables exposure is disallowed.

13. Once you finish, click **Save** to save your workflow design. The **Save** button will become inactive and the designer window will remain open.

    If you exit the Designer without saving changes, you will be prompted to save them. In the **Save Confirmation** dialog choose an appropriate action.

14. To return to the Function Summary from which you called the Designer, use the **Close** button in the top right corner of the window.

Applying Execution Rules

This section lists the rules that can be applied to the execution of a Function. Functions can handle one or more table rows in a single call. When multiple rows are sent to the server at the same time, you can specify a Function to take action on the batch of rows in a specific way.

For the **Send E-Mail** and **Set by Query** widgets, you need to specify an execution rule. The available options are:

- **Once for All Passed Rows** - The action will be executed for all passed table rows.
- **Per each Row in Table** - The action will be executed for each row of a specific table selected from the list of tables available for this Function.

Certain Function widgets have default execution rules. The following table lists these Actions and their default execution rules:

Function Widget	Execution Rule
Raise Exception	Once for All Passed Rows
Send Message	Once for All Passed Rows
Set Field	Per each Row in Table
Attach/Remove Data Tag	Once for All Passed Rows
Execute Custom Code	Once for All Passed Rows

Creating Function-Level Variables

You can create custom, Function-specific variables and make them available for actions and conditions within the Function.

You can use these variables to bind them to external method parameters, reference them across available conditions and actions, pass and receive data from BO call actions or to hold intermediate results during Function execution.

Usage of Function-level variables is limited to the Function where they were defined. Intermediate data they hold cannot be passed between multiple Functions.

The basic flow of utilizing Function-level variables within the Function workflow includes the following:

1. Define Function-level variables. You can create simple variables such as a **string** or **decimal**, or complex variables of **TableSet type**.

The list of available TableSets is limited to the Assembly and Service references of the library.

2. Assign data to variables using the available actions or through a custom code.
3. Use the **Invoke BO Method** action and map variables to parameters as needed.
4. If needed, process returned values. Returned values may end up in variables that were mapped to return.

Using Caller Widgets

Use the caller widgets to invoke a Business Object method or perform a custom code action.

Execute Custom Code

Use the Execute Custom Code action to create a custom method of the Function. Leverage this feature to build a custom action that runs when the condition(s) on a Function workflow activate; you create these actions using the C# programming language. This code environment is updated with each .NET runtime version, so you can enter code using the latest C# version.

This action is only available to users belonging to the Functions Power Developer security group.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
No	Yes

Parameters

The parameters of the current workflow item:

Synchronously execute custom code...

Code.....

Click this link to launch the **Enter Custom Code** application.

The following controls are available within this item:

- **Code** - Use this card to enter a code snippet written in C# language you want the Function to execute

- **Usings & References** - This button launches the **Directive Usings and References** dialog. When you design a custom code, you can use this window to specify additional **C# Usings** for the Function. Also, you can use this window to add additional **References** to assemblies from the **Server\Assemblies** or **\Server\Customization\Externals** folders of your Epicor ERP installation.

  This dialog is accessible from either the **Custom Code Editor** and directly from the Function Designer at any stage from a workflow preparation.

  > Following the same security restrictions applied to the Execute Custom Code action, accessing the Directive Usings & References window from the main BPM Designer window is only allowed to users who belong to the Function Power Developer security group.

  Any changes made on the Using & References dialog are automatically applied to the Function.

  > Loading of assemblies from the application server may take a while. You can interrupt and resume the process by using the **Pause** button. To quickly allocate the assembly you need, use the **Assembly name filter** field.

- **Check Syntax** - as you create or complete code expressions, click this button to perform C# code syntax validation. You can perform validations on code expressions at any time during the entry process. The validation process reacts on:

  - Empty code statement
  - Syntax errors

Any error messages are reported in the grid under the Code editor, stamped with the corresponding error type icon. Double-clicking the error brings up the details of the problematic code.

Context Menu

Several context menu options are available to help you design custom code.

To activate the context menu, select the Code... variable in the Execute Custom Code widget statement and right-click anywhere in the code designer area.

The list of options includes:

- **Edit** - This menu allows you to manipulate items within the current code. For example, you can cut, copy, and paste text or undo the latest changes you have made.
- **Parameters** - Use this option to insert Function signature parameters. Function parameters can be either simple parameters such as a string or a decimal, or complex parameters of

tableset type. Complex parameter tables are exposed in Functions in the following format: <functionParameterName>.<tableName> - for example, input.ABCCode. When referencing tables, a list of table columns displays after you type a period (.) after the table name.

Example: input, inputCustomerRow.CustNum

- **Call Context** - Use this option to insert reference to **callContextBpmData** or **callContextClient** tables and fields.

  Type in a dot (.) after the table name and press **Ctrl + Space** to invoke the list of available columns of this table. Select the column you need, it will be added to the table name.

  Example: callContextClient.CurrentCompany

- **Variables** - This menu becomes available if at least one function-level variable has been created for the current Function. You can reference both simple variable types and complex variable types of a tableset type.

  The Variables sub-menu is organized as follows:

  - The first level sub-menu displays all variables you created within the Function; the list of variables is sorted alphabetically. When you click on a variable of simple type, its name is inserted into the code.
  - For a tableset type variable, click on its title to expand additional sub-level. On this list, a name of the complex variable is listed first, followed by the list of tables it contains. When you double-click on a variable name at the top, it again gets inserted into the code. Double-clicking on a table inserts the reference using the **VariableName.TablesetTableName** format.

    Example: CustomerVariable, CustomerContainer.ShipTo

Environment Variables

You can enter code that references both an environment (instance) type and the name of the environment. You can then enter code that only runs when users run the BPM directive in the specific tenant environment. These two variables are included in the **Sandbox.Environment** property:

- **instanceType** - Defines the overall environment instance. Set this variable to Pilot, Third, Live, and so on.
- **nameOfEnvironment** - Defines the unique name for each tenant instance, such as iv185443, plt459313, thrd852094, and so on.

These variables are currently NOT available in SaaS environments such as Government Cloud. The variables in these environments will always default to `null`.

You enter these variables using the following syntax:

```
this.output = $"Name: '{this.Sandbox.Environment.ApplicationEnvironmentName}', Type = '
{this.Sandbox.Environment.ApplicationInstanceType}'";
```

Invoke BO Method

Use this action to call a service method from within a Function.

Function is a part of the customization which runs as a part of the call to the service method initiated by the Epicor Client, another service or an external software. Within the Function, the service method exposes its parameters and if present, also the return value. Method parameters can be either simple parameters such as a string or a decimal, or complex parameters of tableset type. In Functions, complex parameter tables are exposed in the following format: <functionParameterName>.<tableName> - for example, **ds.CustomerDocs**.

To configure this action:

- Specify the service method you want to call.
- Configure parameters exposed by the selected method. You define what data you want to pass to the method as well as what to do with the data the call returns. You can use the following input for the parameters:
    - Function-level variables of the same type as the selected method parameter - simple or TableSet type.
    - Specified expression - use this option compose an C# expression assigned to the selected parameter.
    - [ignore] - The method accepts this option for non-mandatory parameters that have the OUT direction.

You can use this action with the following **Function Types:**

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of the current workflow items are listed in this topic.

Invoke specified BO method with specified parameters

Specified (BO method)

Click this link to launch the **Method Search** window. Use this dialog window to locate a specific method within a service you want to invoke.

The list of services contains only the services that you added to **Library References > Services**.

Select the service you wish to invoke and click **Search**.

From the **Search Results** grid, select the method you want to call.

Specified (parameters)

Click this link to launch the Setup Method Parameters window.

The following information display on this window:

Field	Description
Business Object	Displays the name of the selected service.
Method Name	Displays the name of the selected service method.
Name	Displays the name of the service method parameter (argument).
Type	Displays the .NET classification for each parameter. Method parameters can be either simple parameters such as a string or a decimal, or complex parameters of TableSet type.
Direction	Indicates how the data flows through the argument. • **in** - Indicates the arguments that pass data into the database. • **out** - Indicates the arguments that push the data back to the client for display. The Business Object method does not require any data from the parameters of this direction. You can either assign these method parameters to variables of the same type, or, you can select the [ignore] binding if you do not need to assign any value. • **in-out** - Indicates the arguments that pass data back and forth between the database and the client. • **retval** - Defines the argument that determines the return value.

Field	Description
Binding	Use the drop-down list to provide input for the method parameters. The following options are available: • Function-level variables of the same type - simple or TableSet type. Typically, you assign values to simple types of variables using the Set Argument/Variable action. To prepare row data for the variables of TableSet type, you can use Fill Table By Query and Update Table By Query actions. • Function parameters of the same type - simple or TableSet type • [ignore] - select if do not want to pass any data to the method parameter; this option is only available for non-mandatory parameters that have the OUT direction. • expr: specified expression - use this option to launch the Specify C# expression window to compose an expression assigned to a method argument. This option is only available for input arguments. Use the Check Syntax button to verify the syntax of your expression. Note the validation process reacts to: • Empty Expression Statement • Syntax Error • Security Issue • **create new variable** - use this option to create a new Fucntion-level variable that has the same type as the field from which this option was launched.

Invoke Function

Use this action to call one Function from another.

Use the **Invoke Function** widget to incorporate a Function into a Widget Function workflow.

You can call other Functions from your current library or other libraries. In the second case, prior to configuring this action, make sure the library containing the Function you are going to call is added to the current library as a reference.

To configure this action:

- Specify the Function library and the Function you want to call.
- Map the parameters of the selected Function. You can use the following bindings:
  - **Function parameters or variables of the same type** as the selected Function parameter - Simple or Tableset type.
  - **Specified expression** - Use this option to compose a C# expression and assign it to the selected parameter.
  - **[ignore]** - Use this option if the caller Function does not need to accept the response (**out**) parameter of the invoked Function.

Parameters

Call library / function using specified parameters

Library

Click this link to launch the **Select Library** window. This window displays the list of available libraries that include the current library (**<ThisLib>**) and other libraries that are defined on the **References > Libraries** card of the current library.

Select the library that contains the Function you want to call and click **OK**.

Function

Click this link to launch the **Function Search** window. You can use the **Starting At** field to narrow your search.

By default, the **Search Results** grid contains the list of all Functions in the previously selected library. If you selected the current library, the grid lists all Functions that belong to it except the current Function to prevent recursive logic.

Select the Function you want to call and click **OK**.

Specified

Click this link to launch the **Setup Function Parameters** window that contains the parameters defined in the selected Function.

The **Parameters** grid displays the following information:

Field	Description
Name	Displays the name of a Function parameter.
Type	Displays the .NET data type of each parameter. Function parameters can be either simple parameters such as a string or decimal, or complex parameters of Tableset type.

Field	Description
Direction	This field indicates how the data flows through the parameter. • **in** - Indicates the parameters that pass data into the Function. • **out** - Indicates the parameters that return data to the Function caller, in this case another Function. You can either assign an out Function parameter to an existing or new Function-level variable of the same type or select the [ignore] binding if you do not need the data returned from the Function.
Binding	Indicates how the data flows through the argument. • **in** - Indicates the arguments that pass data into the database. • **out** - Indicates the arguments that push the data back to the client for display. The Business Object method does not require any data from the parameters of this direction. You can either assign these method parameters to variables of the same type, or, you can select the [ignore] binding if you do not need to assign any value. • **in-out** - Indicates the arguments that pass data back and forth between the database and the client. • **retval** - Defines the argument that determines the return value.

Field	Description
Binding	Select the type of binding for a Function parameter from a drop-down list. The following options are available: • **Existing Function parameter or variable of the same type** - Simple or Tableset type. • **Create new variable** - Use this option to create a new Function-level variable that has the same type as the corresponding Function parameter. • **[ignore]** - Select this option if you do not want to consume any data from a Function **out** parameter. This option is only available for parameters that have the out direction. • **Expr: specified expression** - Use this option to launch the **Specify C# expression** window to compose an expression assigned to a Function parameter. This option is only available for input parameters. Use the **Check Syntax** button to verify the syntax of your expression. The validation can fail due to: • An empty expression statement • A syntax error • A security issue

Using Flow Chart Widgets

Use the Condition widget to make sure that your workflow executes appropriate actions only if a certain condition is met.

Condition

The conditions define when the Function actions will execute.

To add a condition, drag the Condition widget to the designer area and click **New** on the **Condition** tab of the widget **Properties**. There are several condition statements available. Please go to the **Flow Chart > Condition > List of Available Conditions** topic for details. Each condition statement contains one or more links. Click each link in the statement to open a application where you can

configure details of the statement. After you configure each link and return to the Conditions properties, the link text is replaced with the values you selected.

If you enter more than one condition statement, you can use a logical **Operator**, such as **And** or **Or**, to define the relationship between the current statement and the preceding statement. You can also group statements by adding parentheses in the **Prefix** and **Postfix** fields. The system evaluates the condition statements in the order they appear and according to how they are grouped. To change the order of the statements, click the Up or Down Arrow buttons in the toolbar. Condition statements that appear higher in the grid supersede those that appear lower.

Within the workflow, the Condition element may have multiple inbound connections and two outbound connections (exit points). The Function condition can be evaluate to either **True** or **False**. This gives you a flexibility in designing the processing flow through directives.

List of Available Conditions

The list of conditions contains the following statements:

- **Number of rows in the designed query is more than or equal to 1**

  Use this condition to create a query and evaluate the number of rows it returns. If this comparison evaluates to **True**, the Function workflow continues the path using the **True** exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
designed	Click this link to launch the Compose Query application similar to Business Activity Query Designer. The query you create evaluates the number of rows returned by the query designed against the comparison value you select later in this statement. In addition to standard tables, the query phrases can run against in-memory tables to analyze values passed in datasets or linq row sets. Note the Function query data access is restricted by the current company the directive runs in.
is more than or equal to	Click this link to toggle between six values: is equal to, is not equal to, is less than, is less than or equal to, is more than or is more than or equal to. Select a comparison option that is used against the number of rows returned by the selected query.
1	Click this link to launch the Specify a Value application. Use this application to enter a numeric value that evaluates against the number of rows returned by the query.

- The specified argument/variable is equal to the specified expression

Use this condition to evaluate a Function method argument.

You can use this condition with the following Function Types:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified (argument)	Click this link to launch the **Select an Argument** application. Use this application to select which parameter you would like to evaluate against the other variables in the condition statement.
is equal to	Click this link to toggle between eight values: is equal to, **is not equal to, is less than, is less than or equal to, is more than, is more than or equal to, begins is equal to with, ends with, contains or matches**. Select a comparison option used to compare the selected field against the value defined in the expression later in this statement.
specified (expression)	Click this link to launch the Specify **C# expression** application. Use this application to compose an expression that evaluates against the comparison. The expression can contain literal values, data from the current transaction, and functions that can perform calculations and data type conversions. Use the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts to: • Empty Expression Statement • Syntax Error • Security Issue

- **The specified argument/variable contains the specific text**

  Use this condition to evaluate if the specific argument includes the specific word expression. If this comparison evaluates to **True**, the Function's actions are run.

  Do not enclose the text in double-quotes, except for the quotes that are a part of the text.

  You can use this condition with the following **Function Types:**

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified (argument)	Click this link to launch the **Select an Argument** application. Use this application to select which parameter you would like to evaluate against the other variables in the condition statement.
contains	Click this link to toggle between six values: **equals to, not equals to, begins with, ends with, contains** or **matches**. Select a comparison option that is used against the argument variable.
specified (expression)	Click this link to launch the **Enter Word** application. Use this application to enter a word expression that evaluates against the argument.

- The specified field has been changed from any to another

  Use this condition to monitor a specific field contained within the current transaction. If the field's value changes to another value that you define, the comparison evaluates to **True** and the Function workflow continues the path using the **True** exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified	Click this link to launch the Select Table Field(s) application. Use this application to specify a field that is part of the Function condition statement. You can choose a field from any table included in the Function Library references (**References > Tables**). The application compares the field value to a value (any to another) you specify.
any	Click this link to launch the **Specify a Value** application. Use this application to enter a value that is evaluated. You can also select the **Null Value** or **Any** value you want to include in the comparison.
another	Click this link to launch the **Specify a Value** application. Use this application to enter a value that is evaluated. You can also select the **Null value** or **Any** value you want to include in the comparison.

- The specified call context field is equal to the specified expression

  Use this condition to monitor the value of a context variable in the current data transaction, such as the **CurrentCompany** or **CurrentUserId**. The application compares this argument

against an expression that you specify. If this comparison evaluates to **True**, the Function workflow continues the path using the **True** exit point.

You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified	Click this link to launch the Select Table Field(s) application. Use this application to specify a field that is part of the Function condition statement. You can choose a field from any table included in the Function Library references (**References > Tables**). The application compares the field value to a value (any to another) you specify.
any	Click this link to launch the **Specify a Value** application. Use this application to enter a value that is evaluated. You can also select the **Null Value** or **Any** value you want to include in the comparison.
another	Click this link to launch the **Specify a Value** application. Use this application to enter a value that is evaluated. You can also select the **Null value** or **Any** value you want to include in the comparison.

- The specified call context field is equal to the specified expression

  Use this condition to monitor the value of a context variable in the current data transaction, such as the CurrentCompany or CurrentUserId. The application compares this argument against an expression that you specify. If this comparison evaluates to True, the Function workflow continues the path using the True exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified call context	Click this link to launch the **Select Table Field(s)** application. Use this application to specify a call context field from one of two available **Call Context** tables. These tables are specified call context available for each business method, and you leverage them for custom data storage on both the client and server. The application compares the field value to a value that you specify.

Variables	Description
is equal to	Click this link to toggle between eight values: **is equal to, is not equal to, is less than, is less than or equal to, is more than,is more than or equal to, begins with, ends is equal to with, contains** or **matches**. Select a comparison option that is used against the call context variable.
specified	Click this link to launch the **Specify C# expression** application. Use this application to compose an expression that evaluates against the comparison. The expression can contain literal values, data from the current transaction, and functions that can perform calculations and data type conversions. Use the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts to: • Empty Expression Statement • Syntax Error • Security Issue

- The specified field of the changed row is equal to the specified expression

  This condition statement monitors a specific field within the data transaction. The Function then compares this field value to the comparison option and a final value that you specify. If this comparison evaluates to **True**, the BPM workflow continues the path using the **True** exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified call context	Click this link to launch the **Select Table** application. Use this application to specify a field that is part of the statement. You can select a field from any table included in the Function Library references (**References > Tables**).
the changed row	Click this link to specify which row set is affected when the rule activates. You can toggle between six values: **the added row, the deleted row**, **the changed row, the unchanged** row, the updated row and **all rows**.
is equal to	Click this link to toggle between eight values: **is equal to, is not equal to, is less than, is less than or equal to, is more than, is more than or equal to, begins with, ends with, contains** or **matches**. Select a comparison option used to compare the selected field against the value defined in the expression later in this statement.

Variables	Description
specified (expression)	Click this link to launch the **Specify C# expression** application. Use this application to compose an expression that evaluates against the comparison. The expression can contain literal values, data from the current transaction, and functions that can perform calculations and data type conversions. Use the Check Syntax button to verify the syntax of your expression. Note the validation process reacts to: • Empty Expression Statement • Syntax Error • Security Issue

- The custom code condition is valid

  Use this condition to evaluate a code snippet written in C# language.

  You can use this condition with the following Function Types:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description

code	Click this link to launch the **Enter Custom Code** editor. The code written in editor should return a boolean value - for example, **True**;
	
	The following controls are available within the Editor:
	
	• **Code** - Use this card to enter a code snippet written in C# language you want the Function to execute.
	
	• **Usings & References** - This button launches the **Directive usings and references** dialog. When you design a custom code, you can use this window to specify additional **C# usings** for the Function. Also, you can use this window to add additional References to assemblies from **Server\Assemblies or \Server\Customization\Externals** folders of your Epicor ERP installation.
	
	Note this dialog is accessible from either the Custom Code Editor and directly from the Function Designer at any stage from a workflow preparation. Any changes made on the Using & References dialog are automatically applied to the Function.
	
	• **Check Syntax** - as you create or complete code expressions, click this button to perform C# code syntax validation. You can perform validations on code expressions at any time during the entry process. The validation process reacts on:
	
	  • Empty code statement
	
	  • Syntax errors
	
	Any error messages are reported in the grid under the Code editor, stamped with the corresponding error type icon. By double-clicking the error, the problematic part of the code is presented to the user.

valid	Click this link to select how the condition evaluates the result of the custom code. If **valid** is specified, the result of the custom code is compared with **True**. When you select **invalid**, the result is compared with **False**.

- The specified expression is valid

  Use this condition to evaluate a logical expression.

  You can use this condition with the following Function Types:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified	Click this link to launch the **Specify C# expression** editor. Use this application to compose a logical expression that evaluates against the comparison. The expression can contain specified literal values, data from the current transaction, or functions that can perform calculations and data type conversions. Use the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts to: • Empty Expression Statement • Syntax Error • Security Issue
valid	Click this link to select how the condition evaluates the result of the logical expression. If **valid** is specified, the result of the expression is compared with **True**. When you select **invalid**, the result is compared with **False**.

- The method is called by specified user

  Use this condition to determine whether a specific user did or did not launch the current transaction. This condition statement can be useful for testing directives so that they are activated only by a specific user account. If this comparison evaluates to **True**, the Function workflow continues the path using the **True** exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
is called	Click this link to toggle between **is called** or **is not called** options. Select the option you need; these options determine whether the user did or did not initiate the data transaction.
specified user	Click this link to launch the **User Account Search** window. All the user records in the current database display. Use this window to select a user that is used for this condition statement.

- The user who called the method belongs to specified group

  Use this condition to determine if the current user is or is not a member of a specific security group. The application compares the groups of the user who initiated the transaction with the

security group you specify in the condition. If this comparison evaluates to **True**, the Function workflow continues the path using the **True** exit point.

You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
belongs	Click this link to switch between **belongs** or **does not belong** options. Select the option you need; these options determine whether the user is or is not a member of the security group you specify later in this condition statement.
specified group	Click this link to launch the **Security Group Search** window. Use this window to specify a security group evaluated against the selected user.

- There is at least one updated row in the specified table

  This condition compares the value of a field included in the row set to a table you specify. If this comparison evaluates to **True**, the Function workflow continues the path using the **True** exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
updated	Click this link to specify which row set is affected when this rule activates. You can select the added row, the deleted row, the updated row or unchanged row.
specified	Click this link to launch the **Select Table** application. Use this application to specify a table that is monitored by this condition statement. Use this application to choose any table linked to the current data transaction.

- This application instance is a Non-Production instance

  Use this condition to specify the status of the application server instance your workflow will apply to.

  The application server status is set up in the Epicor Administration Console (EAC). For more information, please refer to the EAC help documentation.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
Non-Production	Click this link to toggle between two values: **Production** and **Non-Production**. Select the status of the application server instance you need for the workflow.

- Time is in the specified time frame

  Use this condition to check if the directive execution time matches the indicated time frame. If this comparison evaluates to **True**, the Function workflow continues the path using the **True** exit point.

  You can use this condition with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Variables	Description
specified	Click this link to launch the **Specify a Time Frame** application. Use this application to define a time frame that is part of this condition statement.

Using Label Widgets

Use the widgets from the **Labels** group to attach or remove **Data Tags**.

The **Attach Data Tag** and **Remove Data Tag** labels are enabled in the Function Designer only if at least one function-level variable of TableSet type is defined in the Function.

The tags are unstructured text values that provide a way to group otherwise unrelated records so that you or other users can search for them. Epicor Functions can leverage data tags to create custom application functionality.

Example: You can use a condition statement to check for the presence of a data tag on a record. This condition statement could be used as part of a directive that sends you an e-mail when a record you tagged has been modified. Also, actions can be used to add one or more tags to a record being processed or remove tags from a record. Adding a tag to a record means that a previously untagged record would appear in your Data Tag search results after it has been updated.

Attach Data Tag

Use this action to attach a data tag to a record. You can apply data tags to records throughout the application.

A common use of data tags may be to group related records for searches or so that Epicor Functions can run when the records are modified.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of the current workflow items are listed in this topic.

Attach the specified public data tag to the changed row of the specified table

Specified (data tag)

Click this link to launch the **Specify a Value** application. Use this application to enter a free-form text value that matches the data tag you want to apply to the current record. If the data tag has not been previously defined, it will be created.

Public

Click this link to toggle between two values: **public** and **current user**. Select public to attach a public data tag to the current record. Select current user to attach a private data tag for the current user to the current record.

The changed row

Click this link to launch the **Select a Row Set** application. Use this application to specify which row is affected when this rule activates - like the added row, the deleted row, the updated row, and so on.

Specified (table)

Click this link to launch the **Select Table** application. Use this application to specify the table that is changed through this action.

Remove Data Tag

Use this action to remove a data tag from a record. You can apply data tags to records throughout the application.

A common use of data tags may be to group related records for searches or so that Functions can run when the records are modified.

You can use this action with the following Function Types:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of the current workflow items are listed in this topic.

Specified (data tag)

Click this link to launch the **Specify a Value** application. Use this application to enter a free-form text value that matches the data tag you want to remove from the record.

Public

Click this link to toggle between two values: **public** and **current user**. Select public to remove a public data tag from the current record. Select current user to remove a private data tag from the current record for the user that initiated the transaction.

The changes row

Click this link to launch the **Select a Row Set** application. Use this application to specify which row is affected when this rule activates – like the added row, the deleted row, the updated row, and so on.

Specified (table)

Click this link to launch the Select Table application. Use this application to specify the table that is changed through this action.

Using Other Widgets

Use the **Other** group widgets to run the following actions:

- Display an exception message.
- Send an e-mail notification to a user or a group of users.
- Display an informational message.

Log Message

Use this widget to extend the information provided for the Function in the Server Log.

You typically use it in complex workflows for debugging BPM logic in case errors occur. Besides the message type and the custom details, you configure in Design Log Message Template, this action also writes the following information to the log:

- Directive Identifiers
- Company ID

You can use this action with the following **Function Types:**

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of this workflow item are listed on this topic.

Log message based on the designed template.

Designed

Click this link to launch the **Design Log Message Template** application where you can build a message to write to the log when the Function is triggered.

Design Log Message

1. Drag the **Log Message** icon and drop it into the workflow design area.
2. Define inbound and outbound connections of this workflow action as necessary.
3. In the **Properties** pane, navigate to the **Action** tab and in the **Actions** field, select the **designed** link.

   The **Design Log Message Template** displays.
4. By default, the **Name** of the message is **MyMessage**. You can manually enter a different name.
5. Select the **Severity** type for the log message.
6. Enter the body of the message.
7. Construct the body of the message. You can enter plain text and include call context data, simple variables, or values queried from related tables and fields. The following insert options become available when you click **Insert** or right-click in the field and invoke the context menu:

   - **Call Context** - Choose to set up substitution of the value in a selected BPM Call Context field. To use this action, the Function must also include an action that sets the corresponding BPM call context variable. For example you may have an Execute Custom Code action that includes logic that sets a callContextBpmData or callContextClient variable, which you can then select as a Call Context substitution.

- **Field Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of the value from a selected field in the Function table. Selection is limited to one field. By default, the **Name** of the query is **MyFieldQuery**. You can enter a different name for your query.
- **Table Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of values from multiple fields in the Function table. In the message, the default behavior of a table query is to insert a comma-delimited list of the selected table field names followed by a comma-delimited list of the corresponding table values. By default, the **Name** of the query is **MyTableQuery**. You can enter a different name for your query.
- **Scalar Variables** - Choose to set up substitution of values using Function-level variables of simple type. The first five variables you define for the Function are available for selection directly from the context menu. By clicking **More** ..., you are presented with the **Select and Argument/Variable** window that lists all simple variables including their types.

  For more information on how to utilize variables, see the **Function Workflow Designer > General Principles > Function-Level Variables** section in the Application Help.

8. The **Debug Mode Only** check box indicates whether the message should be written to the log only when the customization is deployed in debug mode (**Debug Mode** check box selected on the library properties).

   It allows for quick message-based debugging of the Function code. It also eliminates the need to insert and then remove debugging messages from the Function workflow because they will never display in the Release Mode.

   Tip Refer to the **Debugging Directive and Function Sources** topic under **Business Process Management > Working with Business Process Management** for details on the **Debug** compilation mode for BPM Directives and Functions.

9. Click **OK**.

Raise Exception

Use this action to display an exception message when the execution of the Function stops.

When this action is reached on a Function workflow, execution of this Function is stopped, and execution of the server code which launched the Function is also stopped.

The transactions that were not committed at the time an exception occurred are rolled back only if the **Requires Transaction** option is selected in the Function settings.

Function exception messages are transmitted to alternate clients - Web Services and Service Connect.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of the current workflow item are listed in this topic.

Raise exception based on the designed template

Designed

Click this link to launch the **Design Exception Template** application. Use this application to build an exception message generated when the application executes the directive action. You can select one of the following **Severity** modes when setting up a message: **Information, Warning, Error**, and **Update Conflict**.

Within the message, you can include call context data, simple variables or values queried from related tables and fields. The following options become available for selection when you click **Insert** or right-click in the message to invoke the context menu:

- **Call Context** - Choose to set up substitution of the value in a selected Function Call Context field. To use this action, the Function must also include an action that sets the corresponding Function call context variable. For example, you may have an **Execute Custom Code** action that includes logic that sets a **callContextBpmData** or **callContextClient** variable, which you can then select as a **Call Context** substitution.
- **Field Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of the value from a selected field in the directive table. Selection is limited to one field.
- **Table Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of values from multiple fields in the directive table. In the message, the default behavior of a table query is to insert a comma-delimited list of the selected table field names followed by a comma-delimited list of the corresponding table values.
- **Scalar Variables** - Choose to set up substitution of values using function-level variables of simple type. The first five variables you define for the Function are available for selection directly from the context menu. By clicking **More**..., you are presented with the **Select and Argument/Variable** window that lists all simple variables including their types.

Design Exception Message

1. Click the **Raise Exception** icon and drag it to the workflow design area. Typically, place this action after a condition item to display an exception message when a certain condition is met.

2. Define inbound connections of this workflow action as necessary. Note that a **Raise Exception** widget should be the last in the chain. Any widgets after it will not be executed.

   In case a **Raise Exception** widget has outgoing connections, the workflow validation displays a warning message.

3. Navigate to the **Actions** tab of the **Properties** and click the designed link.

   The **Design Exception Template** displays.

4. By default, the **Name** of the message is **MyError**. You can manually enter a different name for the exception template.

5. Select one of the following **Severity** modes when setting up a message: **Information, Warning, Error** and **Update Conflict**.

6. Enter the text of the message.

   When you compose a message, you can click **Insert** or right-click in the message to invoke the context menu. This way, you can add parameters, simple variables or queried values to the message. The following options are available:

   - **Call Context** - Choose to set up substitution of the value in a selected Function Call Context field. To use this action, the Function must also include an action that sets the corresponding Function call context variable. For example, you may have an **Execute Custom Code** action that includes logic that sets a **callContextBpmData** or **callContextClient** variable, which you can then select as a **Call Context** substitution.

   - **Field Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of the value from a selected field in the directive table. Selection is limited to one field.

   - **Table Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of values from multiple fields in the directive table. In the message, the default behavior of a table query is to insert a comma-delimited list of the selected table field names followed by a comma-delimited list of the corresponding table values.

   - **Scalar Variables** - Choose to set up substitution of values using function-level variables of simple type. The first five variables you define for the Function are available for selection directly from the context menu. By clicking **More**..., you are presented with the **Select and Argument/Variable** window that lists all simple variables including their types.

7. Click **OK**.

Send E-Mail

Use this action to send an e-mail notification when the function executes.

You can select to send an e-mail synchronously or asynchronously. Use the Design E-mail template application to define e-mail parameters such as subject, recipients and the body of an e-mail.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of this workflow item are listed in this topic.

Send e-mail asynchronously based on the designed template with rule

Asynchronously

Click this link to toggle between asynchronous and synchronous execution:

- **Synchronously** - The call is made immediately when the action executes.
- **Asynchronously** - The call is queued and executed by the Function system tasks which are part of the Epicor ICE Task Agent service running behind the scenes. The system task runner performs calls to the server every 20 seconds and processes asynchronous actions in the queue. Users cannot configure this system task.

Designed

Click this link to launch the Design Email Template application to build an e-mail message when the BPM action fires.

With Rule

Click this link to launch the **Action Execution Rule** application. You can use this application to define how the action performs. There are two options:

- **Once for All Passed Rows** - The action will be executed for all passed table rows.
- **Per each Row in Table** - The action will be executed for each row of a specific table selected from the list of tables available for this Function.

Design E-Mail Message

1. Click the **Send E-Mail** icon and drag it to the workflow design area.
2. Define inbound and outbound connections of this workflow action as necessary.

3. Select if you want to trigger this action **synchronously** (when the Function executes) or **asynchronously** using the automated Function system task.
4. Click the **designed** link to launch the **Design E-mail Template** application.
5. By default, the **Name** of the template is **MyEmail**. You can manually enter a different name for the e-mail template.
6. Enter e-mail addresses in the **From, To, CC, BCC** and **Reply to** fields as appropriate. The **From, CC, BCC**, and **Reply to** fields are optional.

   When you leave the **From** field empty, the Function first takes the current user's email address defined in **User Account Security Maintenance**. If the user's email address is not specified, the email specified in **Company Maintenance** is used.

7. Enter a main Subject.
8. Construct the body of the message.

   Within the **From, To, CC, BCC, Reply to, Subject,** and email body, you can include call context data, simple variables or values queried from related tables and fields. The following options become available for selection when you click Insert or right-click in a field and invoke the context menu:

   - **Call Context** - Choose to set up substitution of the value in a selected Function Call Context field. To use this action, the Function must also include an action that sets the corresponding Function call context variable. For example, you may have an **Execute Custom Code** action that includes logic that sets a **callContextBpmData** or **callContextClient** variable, which you can then select as a **Call Context** substitution.
   - **Field Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of the value from a selected field in the directive table. Selection is limited to one field.
   - **Table Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of values from multiple fields in the directive table. In the message, the default behavior of a table query is to insert a comma-delimited list of the selected table field names followed by a comma-delimited list of the corresponding table values.
   - **Scalar Variables** - Choose to set up substitution of values using function-level variables of simple type. The first five variables you define for the Function are available for selection directly from the context menu. By clicking **More**..., you are presented with the **Select and Argument/Variable** window that lists all simple variables including their types.

9. Click **OK**.

Show Message

Use this action to display an informational message that you create.

After users read the message and click **OK**, the Function continues processing. Function informational messages are transmitted to alternate clients - Web Services and Service Connect.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

The parameters of this field are listed in this topic.

Show informational message based on the designed template.

Designed

Click this link to launch the **Design Informational Message Template** application. Use this application to enter a message that contains information you want users to see. Based on the condition(s) you define for the Function, the information message displays. You can select one of the following **Severity** modes when setting up an informational message: Information, Warning, Error, and Update Conflict. You can also select if the message displays as an individual message or as a grid item.

Within the message, you can include call context data, simple variables or values queried from related tables and fields. The following options become available for selection when you click Insert or right-click in the message to invoke the context menu:

- **Call Context** - Choose to set up substitution of the value in a selected BPM Call Context field. To use this action, the Function must also include an action that sets the corresponding BPM call context variable. For example, you may have an Execute Custom Code action that includes logic that sets a callContextBpmData or callContextClient variable, which you can then select as a Call Context substitution.
- **Field Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of the value from a selected field in an in-memory table. Selection is limited to one field.
- **Table Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of values from multiple fields in an in-memory table. In the message, the default behavior of a table query is to insert a comma-delimited list of the selected table field names followed by a comma-delimited list of the corresponding table values.
- **Scalar Variables** - Choose to set up substitution of values using Function-level variables of simple type. The first five variables you define for the Function are available for selection

directly from the context menu. By clicking **More** ..., you are presented with the **Select and Argument/Variable** window that lists all simple variables including their types. For more information on how to utilize variables, see the **Function Workflow Designer > General Principles > Function-Level Variables** section in Classic User Interface Help.

Use the **Debug Mode Only** option to make the message display only if the Function library is in Debug Mode (**Debug Mode** option is enabled in the library properties). This allows for quick message-based debugging of the Function code. It also eliminates the need to insert and then remove debugging messages from the Function workflow because they will never show in the Release Mode.

Design Informational Message

1. Select the **Show Message** icon and drag it to the workflow design area.
2. Define inbound and outbound connections of this workflow action as necessary.
3. Navigate to the **Actions** tab of the **Properties** and click the **designed** link in the action statement. The **Design Informational Message Template** displays.
4. By default, the **Name** of the message is **MyMessage**. You can manually enter a different name.
5. Enter the text of the message.

   When you compose a message, you can click Insert or right-click in the message to invoke the context menu. This way, you can add parameters, simple variables or queried values to the message. The following options are available:

   - **Call Context** - Choose to set up substitution of the value in a selected Function Call Context field. To use this action, the Function must also include an action that sets the corresponding Function call context variable. For example, you may have an **Execute Custom Code** action that includes logic that sets a **callContextBpmData** or **callContextClient** variable, which you can then select as a Call Context substitution.
   - **Field Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of the value from a selected field in the directive table. Selection is limited to one field.
   - **Table Query** - Choose to open the **Select Table Field(s)** dialog box and set up substitution of values from multiple fields in the directive table. In the message, the default behavior of a table query is to insert a comma-delimited list of the selected table field names followed by a comma-delimited list of the corresponding table values.
   - **Scalar Variables** - Choose to set up substitution of values using function-level variables of simple type. The first five variables you define for the Function are available for selection directly from the context menu. By clicking More..., you are presented with the

Select and Argument/Variable window that lists all simple variables including their types.

6. Select the **Debug Mode Only** option if you wish to display the message only when the Function library is compiled and running in the **Debug Mode**.

Please refer to the **Debugging Directive and Function Sources** topic under **Business Process Management > Working with Business Process Management** for details on the **Debug** compilation mode for BPM Directives and Functions.

7. Click **OK**.

Using Setter Widgets

Use the items found within the Setters group to change values of selected fields.

Fill Table by Query

Use this action to add data into a target in-memory table. In-memory table can be a table from a dataset parameter or a Function-level variable of tableset type.

Configure this action through a three-step process:

- First, design a Function query. When you construct the query, you can reference standard database tables such as **ERP.Customer** or Function-level variables of tableset type. To complete the query, you must explicitly define which columns you want to display in the result set. You can then use these columns to fill target table columns with data.
- Select a single in-memory table serving as the target.

  If you need to update more than one table within a tableset, for example, **OrderHed** and **OrderDtl**, you must use two **Fill Table By Query** actions within the workflow.
- Configure in-memory table mappings. You can use the following inputs for table fields:
  - Source data retrieved from the Function Query
  - Custom expressions
  - Function-level variables - These variables can be created anytime within the Function workflow design process on the **Variables** tab or while configuring table column mappings.

The number of records added to the selected table is equal to the number of records returned from the Function Query.

You can use this action with the following Function Types:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of the current widget are listed in this topic.

Use the designed query to insert data into the specified table with specified mapping

Designed

Click this link to launch the Compose Query application. Use this application to build a Function Query used with this action. In addition to standard database tables, the query phrases can be run against dataset tables to return data you need. Note the Function query data access is limited by the tables specified in the Function Library references.

Use the Display Fields tab to define which columns display in the Function Query result set and fill the selected target table.

Specified (table)

Click this link to launch the **Select Table** window. Use this window to specify a table you want to fill with data through this action. Number of rows added to the table is equal to the number of unique rows the Function query returns.

On this dialog, use the **Table** field to select either target dataset tables included in the current Function or Function-level variables of tableset type.

Specified (mapping)

Click this link to launch the **Setup Table Mapping** window to configure field assignments within a target table. The following information displays on this window:

Field	Description
Table Name	Displays the name of the selected in-memory table.
Columns Filter	Use this field to search for a specific column within the selected table.
Name	Displays the list of columns within the selected table.

Field	Description
Type	Displays the .NET classification for each field.
Custom	If selected, this check box indicates the field is a custom, user-defined field serving as an extension of the selected table.
Binding	Use the drop-down list to create fields mappings. The list of options includes the following: • **[ignore]** - default state. Accept this value for all fields where no data assignments should be performed by this action. When the table is filled with data, fields with no mappings are assigned their default values. For example, a field of the string type is assigned an empty string. • **Field: [TableName_ColumnName]** - Displays the list of columns you selected on the **Display Fields** tab within the Function Query. Use this option to fill target table column rows using rows retrieved by Function Query. • **Create new variable** - use this option to create a new Function-level variable that has the same type as the field from which this option was launched. • **Existing Function-level** variables of the same type. Note only simple variable types such as string display on this list. • **Specified expression** - Use this option to launch the **Specify C# expression** window to compose an expression assigned to this table column. Use the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts on: ◦ Empty Expression Statement ◦ Syntax Error ◦ Security Issue

Bind Automatically

By selecting this button, Function Query display fields with the matching name and type are automatically mapped to the corresponding target table columns. This function does not overwrite existing column mappings created manually.

Note the automatic binding uses the first matching column from the query display fields.

The auto-mapping feature uses the information about the table, field name and its type. The following rules are applied when two or more fields having the same name are selected from two or more tables:

- Only the fields from the Action's source query with matching type as target fields are considered as candidates for auto-mapping.
- Only fields with matching name are considered as candidates for auto-mapping. This includes incremental naming such as Company and Company1.
- If the target field is found on the list of source query's display fields, it is taken as the source of mapping and the search stops.
- A display field from the Action's source query having the matching name and type is taken as the auto-mapping source. If there is more than one candidate, the one whose full name (Table_FieldName) comes first in alphabetical order is selected.
- A display field from the Action's source query having the matching name and type, but another instance, is considered as a candidate for auto-mapping.

  In Functions, this would be a dataset table or **variable.table** having the same C# type as the target table. If there is more than one candidate, the one whose full name (**Table_FieldName**) comes first in alphabetical order is selected.
- A display field from the Action's source query having the matching name and type, which belongs to or is derived from the same database table as the target one, is considered as a candidate for auto-mapping.

  In Functions, it would be a field from **{Erp./Ice.}Table** from which the target dataset table is derived.
- If there is more than one candidate, the one whose full name (**Table_FieldName**) comes first in alphabetical order is selected.

Clear Binding

By selecting this option, all existing table column mappings are removed. This includes manual and automatic column mappings created for a table.

Set Argument/Variable

Use this action to set the Function argument/variable value to the expression you define.

Usage of this action requires presence of either:

- Simple parameters within the Function.

- Function variables of simple type such as String, Decimal, Boolean, Long, DateTime and so on.

You can use this action with the following **Function Types:**

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of this workflow widget are listed in this topic.

Set the specified argument/variable to the specified expression

Specified (argument/variable)

Click this link to launch the **Select an Argument** application. Use this application to select a Function argument that passes data into the database. The **Name** column displays the name of the argument. The **Type** field displays the .NET classification for each argument.

Specified (expression)

Click this link to launch the **Specify C# Expression** application. Use this application to compose an expression used as Function argument value. The expression can contain literal values, data from the current transaction, and functions that can perform calculations and data type conversions.

The Function code editor supports intelligent code editing, which includes syntax highlighting and code completion working for C# keywords, parameters, directive variables, member lists, etc. Start typing and then press **Ctrl + Space** to bring up suggested options. For example, to invoke a list of columns for a table, enter the name of the table followed by a dot (.) and press **Ctrl + Space**. Select the column you need, it will be added to the table name - for example, **dsABCCodeRow.Company** where **ds** is the name of a method parameter or variable of tableset type.

Click the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts on:

- Empty Expression Statement
- Syntax Error
- Security Issue

Set BPM Data Field

Use this action to set the value of a field in a BPM data table.

This table is used with the custom data storage functionality available through **CallContext** tables. You can check the field value in a condition of a further Function or pass this value on to the client application. For more information, review the **Call Context** topics within application help.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

The parameters of this workflow widget are listed in this topic

Set the specified field of BPM Data to the specified expression

Specified (field)

Click this link to launch the **Select Table Field(s)** application. Use this application to specify a call context field that is part of the Function action.

Specified (expression)

Click this link to launch the **Specify C# expression** application. Use this application to compose an expression used as the call context variable value. The expression can contain literal values, data from the current transaction, Function Parameters, Function-Level Variables and functions that can perform calculations and data type conversions.

The Function code editor supports intelligent code editing, which includes syntax highlighting and code completion working for C# keywords, parameters, directive variables, member lists, etc. Start typing and then press **Ctrl + Space** to bring up suggested options. For example, to invoke a list of columns for a table, enter the name of the table followed by a dot (.) and press **Ctrl + Space**. Select the column you need, it will be added to the table name: **ttABCCodeRow.Company**.

Click the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts on:

- Empty Expression Statement
- Syntax Error
- Security Issue.

Set by Query

Run this action to use a query to change the value of a selected field.

This action is only available to users belonging to the **Functions Power Developer** security group.

Within the Epicor ERP Cloud, this application or feature may not be available or may operate under certain restrictions.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
No	Yes

Parameters

The parameters of the current Widget are listed in this topic.

Set the specified field of all rows identified by the designed query to the specified expression with rule

Specified (field)

Click this link to launch the **Select Table Field(s)** application. Use this application to specify a standard or user-defined field that is changed through this action.

On this dialog, the **Table** field displays tables used within the query. These refer to subsets of the corresponding data of dataset table or Function-level variable of tableset type that matches conditions and relations defined in the query.

Designed

Click this link to launch the **Compose Query** application. Use this application to build a Function query used with this action. In addition to standard tables, the query phrases can be run against in-memory tables to analyze values passed in datasets or LINQ row sets. Note the Function query data access is limited by the tables specified in the Function Library references.

Specified (expression)

Click this link to launch the **Specify C# expression** application. Use this application to compose an expression that is evaluated against the comparison. The expression can contain literal values, data from the current transaction, method parameters, Function-level variables, functions that can perform calculations and data type conversions.

Function code editor supports intelligent code editing, which includes syntax highlighting and code completion working for C# keywords, parameters, directive variables, member lists, etc. Start typing and then press **Ctrl + Space** to bring up suggested options. For example, to invoke a list of columns for a table, enter the name

of the table followed by a dot (.) and press **Ctrl + Space**. Select the column you need, it will be added to the table name: **dsABCCodeRow.Company** where **ds** is the name of a method parameter or variable of tableset type.

Click the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts on:

- Empty Expression Statement
- Syntax Error
- Security Issue

With rule

Click this link to launch the **Action Execution Rule** application. You can use this application to select how the action performs. The available options are:

- **Once for All Passed Rows** - The action will be executed for all passed table rows.
- **Per each Row in Table** - The action will be executed for each row of a specific table selected from the list of tables available for this Function.

Set Field

Use this action to change a selected field to a different value, using a row set action that you define.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters for this workflow widget are listed in this topic.

Set the specified field of the changed row to the specified expression

Specified (field)

Click this link to launch the **Select Table Field(s)** application. Use this application to specify a standard or user-defined field that is changed through this action. Use this application to select a field from a database table included in the Function Library references or an in-memory dataset table from method parameters of tableset type.

The changed row

Click this link to launch the **Select a Row Set** application. Use this application to specify which row set is affected when this rule activates – for example, the added row, the deleted row, the updated row, and so on.

Specified (expression)

Click this link to launch the **Specify C# expression** application. Use this application to compose an expression that is evaluated against the comparison. The expression can contain literal values, data from the current transaction, method parameters, directive level variables, functions that can perform calculations and data type conversions.

Function code editor supports intelligent code editing, which includes syntax highlighting and code completion working for C# keywords, parameters, directive variables, member lists, etc. Start typing and then press **Ctrl + Space** to bring up suggested options. For example, to invoke a list of columns for a table, enter the name of the table followed by a dot (.) and press **Ctrl + Space**. Select the column you need, it will be added to the table name: **dsABCCodeRow.Company**

Click the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts on:

- Empty Expression Statement
- Syntax Error
- Security Issue

Update Table by Query

Use this action to update data within an in-memory table. In-memory table can be a dataset table or a Function-level variable of tableset type.

Configure this action through the following process:

- First, design a Function query. When you construct the query, you can reference both in-memory tables and standard database tables. To complete the query, you must explicitly set which columns you want to display in the result set. You can then use the query rows to update the information within a target table.
- Specify which row set within an in-memory table is affected, when this action activates. You can select added rows, deleted rows, updated rows, added and updated rows, changed rows, unchanged rows and all rows.
- Select a single in-memory table serving as the target for the data update.
- Specify the relation between the target table and the Function Query. This relation defines how rows returned by the query are associated with existing table rows.

- Specify how data is assigned to the target table columns. When you create column mappings, the following options are available for use:
    - By default, in the **Binding** column, all fields are set to **[ignore]**. This state indicates the original state of a fields is preserved and no data updates are performed.
    - Update a target table column rows using column rows retrieved by a Function Query. Note only query columns selected for display and those with a matching data type as a target column display on the list.
    - Create custom expressions.
    - Use existing Function-level variables of the matching type.
    - Create new Function-level variables. These variables can be created anytime within the Function workflow design process on the **Variables** tab or while configuring table column mappings.

When this action executes, the data is first retrieved from a Function Query. This data is then matched with selected set of rows from a target in-memory table, for example, with added rows. This matching is based on rows returned by the query and the relation specified between the query and the target table defined on the **Setup Table Mapping** window.

If the Function Query returns more than one row matching a target table row, only the first row found in the query is used to perform data assignments. The remaining query rows are ignored. The other way around, if for a target table row, no matching rows are returned by the query, the row is excluded from update.

You can use this action with the following **Function Types**:

Widget Function	Widget Function with Code
Yes	Yes

Parameters

Parameters of this workflow widget are listed in this topic.

Use the designed query to update all rows of specified table with specified mapping

Designed

Click this link to launch the **Compose Query** application. Use this application to build a Function Query used with this action. In addition to standard database tables, the query phrases can be run against dataset tables to return data you need. Note the Function query data access is limited by the tables and service tablesets specified in the Function Library references.

Use the **Display Fields** tab to define which columns display in the Function Query result set. You can then use the data retrieved from the query to update the rows within the target in-memory table.

All rows

Select the state of in-memory table record rows you want to update through this action. The state of each record row is defined in the **RowMod** column. Select one of the following row sets:

- all rows - default state
- added rows
- deleted rows
- updated rows
- added and updated rows
- changed rows
- unchanged rows

Specified (table)

Click this link to launch the **Select Table** window. Use this window to specify a single in-memory table you want to update using this action.

On this dialog, use the **Table** field to select either target dataset tables included in the current Function or Function-level variables of tableset type.

Specified (mapping)

Click this link to launch the **Setup Table Mapping** window to configure field assignments within a target table.

The following information displays on this window:

Field	Description
Table Name	Displays the name of the selected in-memory table.
Columns Filter	Use this field to search for a specific column within the selected table.
Name	Displays the list of columns within the selected table.
Type	Displays the .NET classification for each field.

Field	Description
Custom	If selected, this check box indicates the field is a custom, user-defined field serving as an extension of the selected table.

Binding	Use the drop-down list to create fields mappings. The list of options includes the following:
	• **[ignore]** - default state. Accept this value for all fields where no data assignments should be performed by this action.
	When the table is filled with data, fields with no mappings are assigned their default values. For example, a field of the string type is assigned an empty string.
	• **Field: [TableName_ColumnName]** - Displays the list of columns you selected on the **Display Fields** tab within the Function Query. Use this option to fill target table column rows using rows retrieved by Function Query.
	• **Create new variable** - use this option to create a new Function-level variable that has the same type as the field from which this option was launched.
	• **Existing Function-level** variables of the same type. Note only simple variable types such as string display on this list.
	• **Specified expression** - Use this option to launch the **Specify C# expression** window to compose an expression assigned to this table column. Use the **Check Syntax** button to verify the syntax of your expression. Note the validation process reacts on:
	  • Empty Expression Statement
	  • Syntax Error
	  • Security Issue

Entering Code with the Function Editor

Use the Functions Editor to add and edit the C# code of your Function.

Function Editor is normally compatible with .NET Framework 4.8 but we recommend installing the latest version of .NET 6.0 Runtime to assure the application behaves correctly in all cases.

Only **Functions Power Developers** can edit Custom Code Functions.

Review the topics of this section to learn more about the components of the Function Editor.

Here are the three elements that are available regardless of the current layout combination:

Check Syntax

Use this button to validate your code. If the code compiles with errors, their description will be displayed on the **Error List** card below the Editor panel.

Cancel

Click this button to close the Function Editor window. If you have unsaved changes, you will be offered to save them prior to closing the application. Select **Yes** to save or **No** to ignore unsaved changes and exit the application.

OK

Click this button to save your code.

> Prior to saving the changes, the system runs code check to ensure its validity. In case compilation errors are detected, a warning message displays. At this stage, you may temporarily ignore the errors and save the code as is.

Using the Code Sheet

Use this sheet to enter the C# code of your Function.

Editor

Enter your code in this field.

The Function Editor supports intelligent code editing, which includes syntax highlighting and code completion working for C# keywords, parameters, directive variables, member lists, etc.

Start typing and then press **Ctrl + Space** to bring up suggested options. For example, to invoke a list of columns for a table, enter the name of the table followed by a dot (**.**) and press **Ctrl + Space**. Select the column you need, it will be added to the table name - for example, **ABCCodeRow.Company**.

Context Menu

- **Edit** - This menu allows you to manipulate items within the current code. For example, you can cut, copy, and paste text or undo the latest changes you have made.
- **Parameters** - Use this option to select Function parameters and insert them into your code. Parameters are added in the following format: **this.[ParamName]**, where **[ParamName]** is the name of the parameter - for example, `this.input1`.

  Tables from the tablesets of complex parameters are exposed on the **Current Scope** panel as child nodes of these parameters - for example, **Tip**, **Customer**, etc. Double-click on a parameter table in the tree to bring it into your code. You can also type a period (.) after

parameter name, press **Ctrl + Space**, and select a table from the list of options. Tables of complex parameters of Tableset type are added to the code in the following format: this.[ParamName].[Table] - for example, `this.input2.Customer`, where **Customer** is the table from the parameter's tableset.

Use an indexer to address a row in a parameter's table. For example, `this.input2.Customer[0].CustNum` reads the value of the **CustNum** field from a row at position 0.

- **Call Context** - Use this option to insert reference to **callContextBpmData** or **callContextClient** tables and fields.

  Type in a dot (.) after the table name and press **Ctrl + Space** to invoke the list of available columns of this table. Select the column you need, it will be added to the table name.

Example: callContextClient.CurrentCompany

Coding Examples

This topic contains some basic code snippets that explain how you can include Epicor objects into your code.

Function-to-Function Calls

Use the following format to call a Function from the same (current) library: **this.ThisLib.[FunctionID] ([FunctionParameter1], [FunctionParameter2])**. For example, enter this code to invoke **MyFunc2** belonging to your current library:

```
this.ThisLib.MyFunc2(1,2)
```

Use the following format to call a Function from another library: **this.EfxLib.[LibraryID].[FunctionID] ([FunctionParameter1], [FunctionParameter2]).** For example, a call to the **function-1** of the **NewLib** library can coded like this:

```
this.EfxLib.NewLib.function_1(1,2);
```

Use an underscore instead of a dash in a Function ID when you call the Function from your C# code.

Handling Function Output

Single output can be assigned to a Function variable as shown in the below example:

```
var result = this.EfxLib.NewLib.function_1();
```

If the called Function returns several response parameters, you need to map each response parameter - for example:

```
var result =this.EfxLib.Lib_1.Func_1();
this.myTsVar = result.r1;
this.myStrVar = result.r2;
this.myBoolVar = result.r3;
```

where **r1**, **r2**, and **r3** are names of response parameters.

Reviewing Using Statements

Use this sheet to review standard and add custom using statements.

Standard Usings

This panel contains a read-only list of using statements that the Functions framework, by default, adds to the generated custom code. This list depends on the type of application environments and is either ICE or ERP specific.

Usings

This panel contains custom using statements. You can add new and edit existing statements here.

Reviewing a Function's Current Scope

This sheet displays the summary of a Function's scope.

Request/Response

These two nodes list the Function's request and response parameters defined on the **Signature** card.

These nodes do not display in the tree if no input or output parameters are defined in the Function.

Tablesets of complex parameters are displayed as child nodes of these parameters.

BPM Context

This node contains the default call context options - **callContextBpmData** and **callContextClient**. Expand these nodes to view available columns.

Advanced

This node contains the following child nodes:

- **Session** - lists session properties.
- **BAQ Constants** - lists the available system BAQ Constants.

  Please refer to the **System Management > Business Activity Query > Query Builder > Display Fields > Column Select > Calculated Field > BAQ Constants List** topic in the Application help.
- **DB Context** - lists database tables from the library's **Table References** if database access from custom code is allowed (**DB Access from Code** option is not set to **None**).

Make sure that all referenced database tables that can be updated from your custom code or expressions are marked as **Updatable**. All custom update logic for read-only tables will be ignored by the system, and the database will not get updated.

Functions

The tree on this card contains two child nodes - Functions and Operators. These are standard functions and operators that are also available in the BPM Expression Editor.

Functions

This topic contains the list of functions available in the Function Editor.

Math

Mathematical functions execute mathematical operations usually based on input values that are provided as arguments, and return a numeric value as the result of the operation.

Function	Editor View	Description
(x % y)	(%)	(top expression % bottom expression) Returns the remainder of the top numeric expression divided by the bottom numeric expression.
Abs(x)	Math.Abs()	abs(expression) Returns the absolute value of a numeric expression.
Log(x)	Math.Log()	Log(expression) Calculates the natural (e) logarithm of a decimal expression.
Log10(x)	Math.Log10()	Log10(expression) Calculates the base-10 logarithm of a decimal expression.

Function	Editor View	Description
Negate (-x)	decimal.Negate()	negate function Reverses the sign of a decimal expression.
Pow (x,y)	Math.Pow(,)	Pow(expression, exponent) Returns the floating-point expression raised to the power of a floating-point exponent.
Round(x, y)	Math.Round(,)	round(expression , precision) Rounds a decimal expression to a specified number of places after the decimal point.
Sqrt(x)	Math.Sqrt()	sqrt(expression) Returns the square root value of a floating-point expression.
Truncate (x, y)	Math.Truncate()	round (expression, precision, 1) Truncates a decimal expression to a specified number of decimal places, returning a decimal value.

String

String functions manipulate a string or query information about a string.

Function	Editor View	Description
Format (x,y,...)	string.Format("{0}",)	string.Format(format, object1, ...) Converts a value of any data type into a character value using format string.
Replace (x,y,z)	BpmFunc.Replace(, ,)	Replace(expression ,search ,replace) Performs a search and replace function on a string expression.
ToLower(x)	BpmFunc.ToLower()	ToLower(expression) Converts any uppercase characters in a string expression to lowercase characters.
ToUpper(x)	BpmFunc.ToUpper()	ToUpper(expression) Converts any lowercase characters in a string expression to uppercase characters.

Function	Editor View	Description
x.Length	.Length	expression.Length Returns the number of characters in an expression.
x.Substring (y,z)	.Substring(,)	expression.Substring(position, length) Extracts a portion from a string expression.
x.ToString()	.ToString()	expression.ToString() Converts a value of any data type into a string value.
x.Trim()	.Trim()	expression.Trim() Removes leading and trailing white spaces from a string expression.
x.TrimEnd()	.TrimEnd()	expression.TrimEnd() Removes trailing white space from a string expression.
x.TrimStart()	.TrimStart()	expression.TrimStart() Removes leading white spaces from a string expression.

Date

The following is the list of date functions available for use within an expression.

Function	Editor View	Description
AddInterval (x,y)	BpmFunc.AddInterval(,)	AddInterval(expression, timespan) Adds the specified TimeSpan to the specified DateTime expression and returns the result as a new DateTime.
AddInterval (x,y,z)	BpmFunc.AddInterval(, ,)	AddInterval(expression, interval-amount, interval-unit) Adds a date interval to, or subtracts a date interval from specified expression. Expression should be of date type. Interval-unit is a character constant with one of the following values: IntervalUnit.Years, IntervalUnit.Months, IntervalUnit.Weeks, IntervalUnit.Days, IntervalUnit.Hours, IntervalUnit.Minutes, IntervalUnit.Seconds.

Function	Editor View	Description
Date(yyyy, mm, dd)	BpmFunc.Date(, ,)	Date(year, month, day) Initializes a DateTime value with a set of year, month, and day.
Day(x)	BpmFunc.Day()	Day(expression) Evaluates a date expression and returns a day of the month as an integer value from 1 to 31.
Month(x)	BpmFunc.Month()	Month(expression) Evaluates a date expression and returns a month integer value from 1 to 12.
Now	BpmFunc.Now()	Now Returns the current company date and time.
Today	BpmFunc.Today()	Today Returns the current company date.
WeekDay(x)	BpmFunc.WeekDay()	WeekDay(expression) Evaluates a date expression and returns the day of the week as an integer value from 1 (Sunday) to 7 (Saturday) for that date.
Year(x)	BpmFunc.Year()	Year(expression) Evaluates a date expression and returns the year value of that date in the Gregorian calendar, including the century, as an integer value.

Conversion

Conversion functions transform an expression of one data type to another.

Function	Editor View	Description
Convert.ToBoolean (x)	Convert.ToBoolean()	Convert.ToBoolean(expression) Converts any data type into the boolean data type.
Convert.ToDecimal (x)	Convert.ToDecimal()	Convert.ToDecimal(expression) Converts an expression of any data type to a decimal value.

Function	Editor View	Description
Convert.ToInt32(x)	Convert.ToInt32()	Convert.ToInt32(expression) Converts an expression of any data type to a 32-bit integer value, rounding that value if necessary.
Date(yyyy,mm,dd)	BpmFunc.Date(, ,)	Date(year, month, day) Initializes a DateTime value with a set of year, month, and day.
DateTime(x)	new DateTime()	DateTime(ticks) Initializes a DateTime value with an integer number of ticks.
DateTime.Parse("x")	DateTime.Parse("")	DateTime.Parse(expression) Converts a string expression into a datetime value.
Format(x,y, ...)	string.Format("{0}",)	string.Format(format, object1, ...) Converts a value of any data type into a string value using format string.
x.ToString(date format)	.ToString("MM-dd-yyyy")	expression.ToString(format) Converts a date value into a string value using a date format string.
x.ToString(decimal format)	.ToString("N")	expression.ToString(format) Converts a decimal value into a string value using a decimal format string.
x.ToString(integer format)	.ToString("G")	expression.ToString(format) Converts an integer value into a string value using an integer format string.
x.ToString(time format)	.ToString("HH-mm-ss")	expression.ToString(format) Converts a date value into a string value using a time format string.
x.ToString()	.ToString()	expression.ToString() Converts a value of any data type into a string value.

Operators

This topic contains the list of operators available in the Function Editor.

Operators are mathematical expressions either used with a single function or used to process two or more functions.

Arithmetic

The following is the list of available arithmetic operators; a mathematical functions that takes two operands and performs a calculation on them.

Operator	Editor View	Description
Add (x + y)	(+)	+ Addition operator Adds two numeric expressions.
Divide (x / y)	(/)	/ Division operator. Divides one numeric expression by another numeric expression.
Modulo (x % y)	(%)	% Modulo operator. Determines the remainder after division.
Multiply (x * y)	(*)	* Multiplication operator Multiplies two numeric expressions.
Substract (x - y)	(-)	- Substraction operator Subtracts one numeric expression from another numeric expression.

Boolean

The following is the list of available Boolean (Logical) operators. Boolean operators evaluate the values and return true or false as output.

Operator	Editor View	Description
And (x && y)	&&	&& And operator Returns a true value if each logical expression is true.
Equal (x == y)	(==)	== Equal operator. Returns a true value if two expressions are equal.
Not (!x)	!()	! Not operator. Returns true if an expression is false and false if an expression is true.

Operator	Editor View	Description
Or (x II y)	||	|| Or operator. Returns a true value if either of two logical expressions is true.

Comparison

The following is the list of available comparison operators used to compare two operands. These operators return true or false based on comparison.

Operator	Editor View	Description
Equal (x == y)	==	== Equal operator. Returns a true value if two expressions are equal.
Greater or Equal (x >= y)	>=	>= Greater or Equal operator. Returns a true value if the first of two expressions is greater than or equal to the second expression.
Greater Than (x > y)	>	> Greater Than operator. Returns a true value if the first of two expressions is greater than the second expression.
Less or Equal (x <= y)	<=	<= Less or Equal operator. Returns a true value if the first of two expressions is less than or equal to the second.
Less Than (x < y)	<	< Less Than operator. Returns a true value if the first of two expressions is less than the second.
Not Equal (x != y)	!=	!= Not Equal operator. Compares two expressions and returns a true value if they are not equal.

Condition

This section holds the conditional (ternary) operator.

Operator	Editor View	Description
x ? y : z	() ? :	If-Then-Else condition. Evaluates and returns one of two expressions, depending on the value of a specified condition.

Other

This section holds other operators you can use to test or compare string expressions.

Operator	Editor View	Description
Begins(x,y)	BpmFunc.Begins(,)	Begins with ... condition. Tests a string expression to see if that expression begins with a second string expression.
Ends(x,y)	BpmFunc.Ends(,)	Ends with ... condition. Tests a string expression to see if that expression ends with a second string expression.
Matches(x,y)	BpmFunc.Matches(,)	Matches(expression, pattern) Compares a string expression to a pattern and evaluates to a true value if the expression satisfies the pattern criteria.
x.Compare(y)	.Compare()	left.Compare(right) Compares two case-insensitive string expressions; the result is less than 0 if left string is less than right string, 0 if the strings are equal, and greater than 0 if right string is greater than left string.
x.Contains(y)	.Contains()	Contains condition. Tests a string expression to see if that expression contains a second string expression.
x.StringEqual(y)	.StringEqual()	left.StringEqual(right) Compares two string expressions and returns true if strings are equal and false otherwise.

Libraries

This card displays the list of library references of the current library and the current library itself.

By default, the **Libraries** tree view contains the **<ThisLib>** node referring to the current library. If the library has references to other libraries, these library references also appear in the tree view. You can expand each library node to view its Functions (if any). You can then perform the following operations:

View Function Details

Select a Function in the tree view to display its signature and short description on the Online Help Panel.

Add a Function to the Code

Double-click on a Function or use drag-and-drop to bring it into your code. Depending on what library the Function belongs to, it is added in the following format:

- Current library Function - **this.ThisLib.[FunctionID]();** For example, **this.ThisLib.MyFunction ();**
- Function from another library - **this.EfxLib.[LibraryID].[FunctionID]();** For example, **this.EfxLib.MyLibrary.MyFunction();**

You can also double-click on a library ID in the tree view or drag and drop it into the Code Editor area. This action pastes the library name into your code in the following format: **this.EfxLib. [LibraryID]** - for example, **this.EfxLib.MyLibrary**, or **this.ThisLib** if you add the current library. As the Function Code Editor supports code completion, you can type in a dot (.) and press **Ctrl + Space** to invoke the list of available entities. From this list, you can manually select a Function that belongs to this library.

If the library or Function ID you are adding to your code contains a dash, this dash is automatically replaced with an underscore to comply with C# syntax rules - for example, function-1 from the my-lib library is added as this.EfxLib.my_lib.function_1();

Online Help

This panel displays description for a selected function or operator from the Functions card.

Select any function or operator on the **Functions** card to display its description on the **Online Help** card below the Editor panel.

This panel also displays the signature and description of the Functions from the libraries added to the current one as references.

Reviewing the Error List

This sheet displays the code and message of error(s) detected during validation.

Code validation reacts to:

- Absence of code in the editor
- Incorrect syntax in code and using statements
- Security issues.

Click **Check Syntax** or **OK** (Save) to trigger code validation. In case compilation errors are detected during saving, an error warning message displays. At this stage, you may temporarily ignore the errors and save the code as is.

If you need to save an unfinished Function that may fail to compile, mark it as **Disabled**. Disabled Functions are not compiled by the system and can be successfully saved for later editing.

Accessing Directories and Files Using the Sandbox Property

You can access system directories and files through both Business Process Management (BPM) directives and Epicor Functions. When you use the Custom Code widget for a directive or function workflow, enter code that uses the **Sandbox** property. This Sandbox property exposes the **ISandbox** interface, and you use this interface to access directories and files.

The ISandbox interface prevents both users from accessing sensitive areas and malicious code from harming your system.

During this help article, we explore:

- Using the FilePath Class
- Accessing Files
- Accessing Directories
- Reviewing Example Code

Using the FilePath Class

You specify the path to a file or directory by using the **FilePath** class. It limits your code so that users can only use the appropriate files that they have rights to access. When you create a FilePath object, you first specify the **ServerFolder** that your code will access. Use these ServerFolders to read and write to directories and files. Two server folders are available:

- **UserData**
- **CompanyData**

You CANNOT specify a path outside of these ServerFolders, so you cannot enter root paths or relative paths.

Typically you enter a FilePath object using this syntax:

```
var filePath = new FilePath(ServerFolder.CompanyData, "TestFile.txt");
```

The FilePath class has both the **Folder** and **Subpath** properties. You CANNOT change the Folder property, but you can enter the Subpath properties you need.

The following examples show ways you can combine a Subpath with additional FilePath data:

```
filePath.Combine(@"Subpath\File.txt");
filePath.Combine(@"Subpath\Subpath2\File.txt");
filePath.Combine("Subpath", "Subpath2", "File.txt"); // Recommended syntax
filePath.Subpath = @"Subpath\Subpath2\File.txt";
```

You can also create new FilePath objects from existing FilePath objects:

```
var filePath2 = new FilePath(filePath1, @"Subpath2\File.txt");
```

When the path has multiple parts that point to directories and files, you MUST specify each part in the path. Do not use a single string. Depending on your operating system, the path separator can be different. By specifying each part, you can then define a path without specifying the path separator.

Accessing Files

The Sandbox.IO.file property gives you access to the .NET **System.IO.File** class. You pass in a FilePath INSTEAD of a string path. You can call most file methods; only the less common FILE methods are not available. For example:

```
var filePath = new FilePath(ServerFolder.CompanyData, "TestFile.txt");
this.Sandbox.IO.File.AppendAllText(filePath, $"Append this text!");
```

You can learn more about how each file method works by reviewing the .NET documentation:

https://learn.microsoft.com/en-us/dotnet/api/system.io.file

Accessing Directories

The Sandbox.IO.Directory provides most of the .NET **System.IO.Directory** class methods. Just like when you access files, you pass in a FilePath INSTEAD of a string path. You can call most directory methods; only the less common directory methods are not available. For example:

```
var filePath = new FilePath(ServerFolder.CompanyData, "Subdirectory");
this.Sandbox.IO.Directory.Create(filePath);
```

You can learn more about how each directory method works by reviewing the .NET documentation:

https://learn.microsoft.com/en-us/dotnet/api/system.io.directory

Reviewing Example Code

This section shows you some examples that use the Sandbox property. These examples all use the following FilePath example:

```
var filePath = new FilePath(ServerFolder.UserData, "Test.txt");
```

Creating Directory Example

```
this.Sandbox.IO.Directory.CreateDirectory(filePath);
```

Deleting Directory Example

```
this.Sandbox.IO.Directory.Delete(filePath);
this.Sandbox.IO.Directory.Delete(filePath, recursive: true);
```

Checking for Existing Directory Example

```
this.output = $"Exists '{filePath}' = {this.Sandbox.IO.Directory.Exists(filePath)}.";
```

Getting Directory Example

```
this.output = $"Directories in '{filePath}': {string.Join(", ",
this.Sandbox.IO.Directory.GetDirectories(filePath).Select(path => path.ToString()))}.";
this.output = $"Directories in '{filePath}': {string.Join(", ",
this.Sandbox.IO.Directory.GetDirectories(filePath, "*abc*").Select(path => path.ToString
()))}.";
this.output = $"Directories in '{filePath}': {string.Join(", ",
this.Sandbox.IO.Directory.GetDirectories(filePath, "*abc*",
System.IO.SearchOption.AllDirectories).Select(path => path.ToString()))}.";
```

Getting Files Example

```
this.output = $"Files in '{filePath}': {string.Join(", ", this.Sandbox.IO.Directory.GetFiles
(filePath).Select(path => path.ToString()))}.";
this.output = $"Files in '{filePath}': {string.Join(", ", this.Sandbox.IO.Directory.GetFiles
(filePath, "New*").Select(path => path.ToString()))}.";
this.output = $"Files in '{filePath}': {string.Join(", ", this.Sandbox.IO.Directory.GetFiles
(filePath, "New*", System.IO.SearchOption.AllDirectories).Select(path => path.ToString
```

```
()))}.";
```

Appending Lines and Text Example

If the file you need DOES NOT exist, the following methods create the file. If you use the **encoding** parameter, a single file SHOULD NOT have text that uses two different encoding values.

You can see the encoding type using either **Notepad++** or **VS Code**.

```
this.Sandbox.IO.File.AppendAllLines(filePath, new[] { "Line1", "Line2" });
this.Sandbox.IO.File.AppendAllLines(filePath, new[] { "Line1", "Line2" },
System.Text.Encoding.Unicode);
this.Sandbox.IO.File.AppendAllText(filePath, "Some text!");
this.Sandbox.IO.File.AppendAllText(filePath, "Some text!", System.Text.Encoding.Unicode);
```

Copying Files Example

```
this.Sandbox.IO.File.Copy(filePath, filePath2);
this.Sandbox.IO.File.Copy(filePath, filePath2, overwrite: true);
```

Creating Files Example

```
using (var fileStream = this.Sandbox.IO.File.Create(filePath))
{
  fileStream.Write(new byte[] { (byte)'A', (byte)'B', (byte)'C' }, 0, 3);
}
```

Creating Text Example

```
using (var stream = this.Sandbox.IO.File.CreateText(filePath))
{
  stream.WriteLine("DEF");
}
```

Deleting Files Example

```
this.Sandbox.IO.File.Delete(filePath);
```

Checking for Existing Files Example

```
this.output = $"File '{filePath}' exists = {this.Sandbox.IO.File.Exists(filePath)}.";
```

Moving Files Example

```
this.Sandbox.IO.File.Move(filePath, filePath2);
this.Sandbox.IO.File.Move(filePath, filePath2, overwrite: true);
```

Opening Files Example

This complex method can open, create, and truncate a file. It can also lock a file, preventing users from reading or writing to it.

This method only reads bytes. If you are reading text, consider another method. You can read text using a **StreamReader** method; you can write text using a **StreamWriter** method.

```
using (var fileStream = this.Sandbox.IO.File.Open(filePath, System.IO.FileMode.Open,
System.IO.FileAccess.Read,
System.IO.FileShare.Read))
{
  var bytes = new byte[3];
  fileStream.Read(bytes, 0, 3);
  this.output = $"Read first 3 bytes of '{filePath}': {string.Join(", ", bytes.Select(b =>
b.ToString()))}.";
}
```

Open File for Reading Example

```
using (var fileStream = this.Sandbox.IO.File.OpenRead(filePath))
{
  var bytes = new byte[4];
  fileStream.Read(bytes, 0, 4);
  this.output = $"Read first 4 bytes of '{filePath}': {string.Join(", ", bytes.Select(b =>
b.ToString()))}.";
}
```

Open Text for Reading Example

```
using (var streamReader = this.Sandbox.IO.File.OpenText(filePath))
{
```

```
  var line = streamReader.ReadLine();
  this.output = $"Read first line of '{filePath}': {line}.";
}
```

Open Write Example

```
using (var fileStream = this.Sandbox.IO.File.OpenWrite(filePath))
{
  var bytes = new byte[] { (byte)'M', (byte)'N', (byte)'O' };
  fileStream.Write(bytes, 0, 3);
}
```

Read All Bytes Example

```
var bytes = this.Sandbox.IO.File.ReadAllBytes(filePath);
this.output = $"Read all bytes of '{filePath}': {string.Join(", ", bytes.Select(b =>
b.ToString()))}.";
```

Read All Text Example

```
var text = this.Sandbox.IO.File.ReadAllText(filePath, System.Text.Encoding.UTF8);
this.output = $"Read all bytes of '{filePath}': {text}.";
```

Reading All Lines Example

```
var lines = this.Sandbox.IO.File.ReadAllLines(filePath, System.Text.Encoding.UTF8);
this.output = $"Read all lines of '{filePath}': {string.Join("\r\n", lines)}.";
```

Write All Bytes Example

```
this.Sandbox.IO.File.WriteAllBytes(filePath, new[] { (byte)'a', (byte)'b', (byte)'c' });
```

Write All Lines Example

```
this.Sandbox.IO.File.WriteAllLines(filePath, new[] { "Line1", "Line2" });
this.Sandbox.IO.File.WriteAllLines(filePath, new[] { "Line1", "Line2" },
```

```
System.Text.Encoding.Unicode);
this.Sandbox.IO.File.WriteAllLines(filePath, (IEnumerable<string>)(new[] { "Line1", "Line2"
}));
this.Sandbox.IO.File.WriteAllLines(filePath, (IEnumerable<string>)(new[] { "Line1", "Line2"
}), System.Text.Encoding.Unicode);
```

Write All Text Example

```
this.Sandbox.IO.File.WriteAllText(filePath, "Testing");
this.Sandbox.IO.File.WriteAllText(filePath, "Testing", System.Text.Encoding.Unicode);
```

Using the Actions Menu to Manage Functions

You manage functions by selecting options on the Actions menu. This help article details the options you can select from this menu.

During this help article, we explore:

- Promoting Libraries to Production
- Demoting Libraries from Production
- Exporting Libraries
- Importing Libraries
- Copying Functions

Promoting Libraries to Production

Use this action to publish Function libraries.

> This action is available to users that belong to the **Functions Administrator** security group only. It is not available to Functions Developers.

1. Open the library you wish to publish.
2. From the **Actions** menu, select **Promote Library to Production**. Click **Yes** to the warning message. The library and all its Functions have been moved to the public space.

   > A **Published** shape appears in the top right corner of the form when the library is published.

When you promote a library to the production mode, most of its settings will become read-only even for its owner because no changes to library Functions are allowed while the library is available for public use. However, Functions Administrators can disable/enable a published library, add/remove Company mapping, and demote it from the production mode for editing.

> To be able to call a published library via REST or from a BPM directive, you must explicitly map it to a Company on the **Security** card of the Library Maintenance. Calling an unpublished library from the current Company does not require mapping this library to a Company. However, the mapping is required if the unpublished library is called from a different Company. Please refer to the **Epicor Functions** topic of the REST Services v.2 Guide in the Application Help for details on Epicor Functions availability via REST.

Demoting Libraries from Production

Use this action to remove a library from production mode. When you demote a library from production, it becomes available for editing.

1. Open the library you wish to demote.
2. From the **Actions** menu, select **Demote Library from Production**.
3. Click **Yes** to the warning message. The library and all its Functions have been removed from the public space.

Use this mode when you develop a new or edit/update existing library to make sure it is not called by users during this time. For development purposes, unpublished libraries are still available via REST by using a special 'staging' URL of the Library/Function. Please refer to the **Epicor Functions** topic of the REST Services v.2 Guide in the Application Help for details on Epicor Functions availability via REST.

Exporting Libraries

Use this action to export a library to a file.

Functions Administrators can export any libraries (published or unpublished) that they can access, while **Functions Developers** and **Power Functions Developers** can export only unpublished libraries that they own or that are shared with them. Unlike Power Functions Developers who can export all types of Functions, basic Functions Developers can export Widget Functions only.

Follow the steps below to export a library:

1. Open the library you wish to export.
2. From the **Actions** menu, select **Export Library**.

   The **Export Library** window displays.
3. Specify the path for the export file.
4. Specify file name.

   By default, the file name matches the system name of the exported library. You can specify a different name for the file.
5. In the **Save as type** field, select the format of the export file. Two formats are available:

   - **Binary format (.efxb)** - A binary JavaScript Object Notation with a Functions-specific header

- **JSON format (.efxj)** - A plaint text JSON

6. Click **Save**.

You can now import this file into a different environment.

Importing Libraries

Use this action to import function libraries from other Epicor ERP instances.

Functions Administrators can import any libraries regardless of their published status. If a published library is imported by a Functions Administrator, the library keeps its published status in the system. However, when imported by Functions Developers and Functions Power Developers a published library becomes unpublished and has to be manually promoted to production via Epicor Functions Maintenance. Please refer to the **Actions > Promote Library for Production** topic for details. **Functions Developers** and **Functions Power Developers** can import only libraries that they own or that are shared with them.

Cloud-Specific Information: In Multi-Tenant environments, only Global Security Managers can import libraries with allowed code widgets.

During import, the owning company of a library can change:

- if a library is imported by a developer and its original Owning Company is different from the current one, the Current Company becomes the owning company of this library;
- If a library is imported by a Functions Administrator and its original Owning Company does not exist or is not available to the user, the Current Company becomes the owning company of this library.

Follow these steps to import a library:

1. From the **Actions** menu, select **Import Library**.

   The **Import Epicor Functions Library** window displays.

2. In the **File** field, specify the path to the function library file. You can either enter the path manually or use the **Browse** button to select file location.

   When you click **Browse**, the **Import Library** window displays. The **File name** drop-down field contains the list of files of the format selected in the **Files of type** field (**Binary format** by default). Change the format to JSON to view the available files of this type. Once you locate the file, click **Open**.

3. Select the **Overwrite mode**. There are three options available:

- **None** - Select this option to prevent importing a library with the same **Library ID** as the ID of some existing library.
- **Upgrade** - Select this option if you would like to upgrade an existing library under an existing ID. If there is a library with a **Library ID** that matches the imported library ID, the system compares their **OriginalID** and **GlobalID**. If these IDs match and the version of the imported library is later than the version of the existing one, the system overwrites the existing library.

  If a **New Library ID** is not specified during import, the imported library's ID is taken from the **LibraryID** property specified in the library file.

  OriginalID is the ID that a developer assigns to a library in the **Library ID** field during its creation. If you change Library ID in Epicor Functions Maintenance, OriginalID will not change. **GlobalID** is a GUID assigned to a library by the system. Both of these IDs are not visible in the user interface.

  You may have several versions of a library under different IDs in the system.

- **Force** - Select this option to import a library ignoring version and Library ID validation. The system will overwrite the library with matching **Library ID** even it has different **OriginalID** and **GlobalID** or is an earlier version of an existing library.

4. If you want to import the library into your system under a different ID, specify a **New Library ID**.
5. Click **Import**.

   If an error occurs during import, the **Import process messages** panel displays the detailed information on the error.

6. Click **Close**.

Copying Functions

Use this action to create a copy of a Function within a library.

When you copy a Function, all of its settings (ID, Description, Disabled/Enabled status, etc.) and properties (workflow design, request and response parameters, and notes) are also copied.

Functions Developers and **Power Functions Developers** can copy Functions owned by or shared with them. You cannot copy Functions from published libraries.

Follow the steps below to create a copy of an existing Function:

1. Open the Function you wish to copy.
2. From the **Actions** menu, select **Copy Function**.

   The Function card of the application displays the copy of the selected Function.

> By default, the ID of the newly created Function displays in the following format: **function-[index]**, where **index** is used to ensure the ID of the newly created Function is unique.

You can edit the newly created Function as you like.

Using Application Logging for BPM Directives and Epicor Functions

You can create custom loggers in both Business Process Management (BPM) directives and Epicor Functions. Typically you create the logger in a functions library that you then call from a BPM directive. The logger generates the application logs. Application logging captures errors and tracks specific Kinetic activity. You define what an application log captures and tracks by creating a custom logger. You can then better locate the causes of issues or capture process calls.

After you resolve an issue or capture the data you need, either remove or comment out the logging code. Running instances of multiple logs which generate files in the same destination may cause log messages to write in different, unexpected sequences. Running multiple logs can also slow performance.

During this help article, we explore how to create custom loggers:

- Creating Default Logs
- Creating Specific Logs
- Creating Application Insights Logs
- Testing Application Logs

Creating Default Logs

You typically will set up the log to run using the default configuration. You manage these loggers on your Epicor server, so the system will send all the logger instances to the same folder destination(s). These default loggers gather general information from Kinetic.

Use the following code to create a default logger:

```
using logger = Ice.Logging.ApplicationLoggerBuilder.CreateDefaultBuilder(this.Session,
"LogId")
    .Build();

logger.LogInformation("Log message");
```

The CreateDefaultBuilder statement needs both a **Session** and a **Log Identifier** (LogId) parameter. This Session parameter is defined on the user account configuration. When the file log generates, this Session parameter helps build the directory path where the log file is stored. For example, the MANAGER user session is configured to write logs to the C:\EpicorData\Users\Manager folder.

The LogId parameter identifies the log, so it is the base name for the file. For example, if the LogId is MyLog, the log file name will be MyLog.log. If you generate an Application Insights log, the logger adds the MyLog parameter to the customDimensions.

If you need to add more settings to the log, call a **Configure** method. Place this method before the Build method. For example, to write the Date/Time before each log entry, enter the following configuration:

```
using logger = Ice.Logging.ApplicationLoggerBuilder.CreateDefaultBuilder(this.Session,
"LogId")
    .ConfigureFile(options => options.MessageOptions.TimestampFormat =
TimestampFormat.DateTime)
    .Build();
```

Creating Specific Logs

Customize your log by using the **CreateBuilder** statement. Only the information you specify in the logger writes to the log and it saves as flat text. You can create a specific log by entering code similar to the following example:

```
var logger = ApplicationLoggerBuilder.CreateBuilder()
    .SetMinimumLevel(LogLevel.Information)
    .AddFile(
        options =>
        {
            options.Session = this.Session;
            options.FileName = "Test.log";
            options.Folder = LogFolder.User;
            options.MessageOptions.QuoteValues = false;
            options.MessageOptions.ShowLogLevel = true;
            options.MessageOptions.TimestampFormat = TimestampFormat.Time;
        })
    .Build();
using (logger)
{
    logger.LogInformation("This is a test: {Value}", "test");
}
```

The parameters in this code do the following:

- The **SetMinimumLevel** method define what level this log will write. The default value is LogLevel.Information, however you can change the level to define the output. For example, if you enter LogLevel.Error, the log file only generates errors messages and WILL NOT generate logs with less precedence like LogLevel.Information.
- The **Session** property defines the current session. This property retrieves tenant, company, and user information.

- The **FileName** property defines the name of the file that will write to the server. This file always writes to the server log folder, which is typically the server data folder that has the tenant identifier. This path is similar to the following:
    - C:\EpicorData\TenantId\CompanyId\Log\UserId

  You can also specify **subfolders** with the FileName property. To specify subfolders, use this format:
    - folder1\folder2\folder3\log.txt
- The **Folder** property can use the following options:
    - **Company** - The log file writes to the Company log folder.
    - **LegacyUser** - This is a legacy option you SHOULD NOT use. The log file writes to the Company+Log+User folder.
    - **LegacyProcessesUser** - This is a legacy option you SHOULD NOT use. The log file writes to the Company+Processes+User folder.
    - **User** - The default option, the log file writes to the User log folder.
- The **MessageOptions** class defines what information you want in each log entry. You can use the following options:
    - **QuoteValues** - Indicates if the log will add double quotes around string values. This can make the string text easier to read. This is what the text looks like with and without QuoteValues:

      ```
      This is a value: test
      This is a value: "test"
      ```

    - The **TimestampFormat** property adds the time to each log entry. You can pull this value from any time zone. The typical options you use are local time, UTC time, and company time. To use UTC time, enter the following code:

      ```
      options.MessageOptions.TimestampTimeZone = TimeZoneInfo.Utc;
      ```

      To use company time, enter the following code:

      ```
      options.MessageOptions.TimestampTimeZone =
      Ice.Lib.CompanyTime.GetTimeZoneForCompany(this.Session.CompanyID);
      ```

    - The **ShowLogLevel** property displays any additional information you want in each entry. Using the above example, it writes the text included with the logger.LogInformation code:

```
This is a value: test
```

- The **FileSizeLimitBytes** property defines how large each file can be. When a log file reaches this limit, the logger creates a new log file that uses the same name and adds a number to the end of it. For example, when a log.txt file reaches this limit, the logger creates a new log_001.txt file.
- The **TruncateFile** property indicates that if a log file with the same name exists in the same folder, the logger removes the contents from the existing file and writes the new contents to it. You can only use this property when you DO NOT use the FileSizeLimitBytes property.

  If the existing log file is in use, you cannot truncate it. This error message displays in the log file:

  ```
  The truncate attempt of the log file has been unsuccessful.
  ```

Creating Application Insights Logs

Application Insights logs include a structure so you can search and filter on their log entries. You can search, filter, and chart data in the same way that you do in a SQL query. It does this by saving each piece of data individually. You create Application Insights loggers by entering code similar to the following:

```
const string connectionString = "InstrumentationKey=aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa;IngestionEndpoint=https://centralus-0.in-
.ap-
plication-
insights.azure.com/;LiveEndpoint=https://centralus.livediagnostics.monitor.azure.com/";
using var logger = ApplicationLoggerBuilder.CreateBuilder()
    .AddApplicationInsights(
        options =>
        {
            options.ConnectionString = connectionString;
        })
    .Build();
```

This log is now set up to generate each piece of data separately, so you can search on each one. For example, if you run the following code:

```
logger.LogInformation("This is a test: {Value}", "test");
```

The logger saves each part of this code as a separate item you can find in a search:

```
MessageTemplate: This is a test: {Value}
Value: test
message: This is a test: "test"
```

You can then use the Kusto query language to search for matching entries. For example, if you want to find the test value, enter the following code:

```
traces
| where customDimensions.Value == "test"
```

Using Different Contexts

If you need to create logs that sort or filter using different contexts, configure these contexts based on the kind of loggers you need. For example, MRP is a context. Because of this context, the MRP log writes to a separate log file. You can then open this MRP log file and only see MRP log entries.

Because Application Insights logs generate to a single location, you can find the data you need to review. Enter the contexts you need in the logger code. For example, this code creates both an MRP context and a JobId context for Job 314159:

```
const string connectionString = "InstrumentationKey=aaaaaaaa-aaaa-aaaa-aaaa-
aaaaaaaaaaaa;IngestionEndpoint=https://centralus-0.in-
.ap-
plication-
insights.azure.com/;LiveEndpoint=https://centralus.livediagnostics.monitor.azure.com/";
using var logger = ApplicationLoggerBuilder.CreateBuilder()
    .AddContext("Context", "MRP")
    .AddContext("JobId", 314159)
    .AddApplicationInsights(
        options =>
        {
            options.ConnectionString = connectionString;
        })
    .Build();
```

After you enter the logger code, you can also add more context to the entries by entering code similar to the following:

```
var additionalContext = new Dictionary<string, object>
{
    ["PartId"] = "2718"
};
using (logger.BeginScope(additionalContext))
{
    logger.LogInformation("Some log text");
}
```

The logger includes this additional context in all log entries until the BeginScope code disposes.

Now that you have added these contexts in the logger, you can find them using Kusto query language. This example returns all the log entries that use the MRP context.

```
traces
| where customDimensions.Context == "MRP"
```

You could also search for the JobId 314159 and narrow the search to only show entries for a specific time range. The entries will then be similar to what generates in default log files.

Writing Exception Logs

You can set up Application Insights logs to track exceptions. These log entries then use the **exception** type instead of the **trace** type. The following example creates an exception log:

```
logger.LogInformation(exception, "Processing failed with {Value}", "3.14159");
```

Now search for exceptions using a Kusto query. This query will use the exceptions value. For example, this query searches for the exception type, message, and method that caused the exception.

```
exceptions
| where type == "System.ArgumentNullException"
| where outerMessage == "Value cannot be null. (Parameter 'NAME')"
| where method == "Ice.Logging.TestByHand.Test_by_hand"
```

The logger does not write the formatted message, but the customDimensions does have the MessageTemplate and any values you specify in it. Use the following query to search for the message template:

```
exceptions
| where customDimensions.MessageTemplate == "Processing failed with {Value}"
```

Testing Application Logs

You can use application logging in application code, Business Process Management (BPM) directives, and Epicor Functions. The easiest way to test your log is by creating an Epicor Function for it:

1. Launch **Epicor Functions Maintenance**.
2. Create an Epicor Function library.
3. Select the **Custom Code Functions** check box.
4. Add the following assemblies from the **Server\Assemblies** directory:
   - **Microsoft.Extensions.Logging.dll**
   - **Microsoft.Extensions.Logging.Abstractions.dll**
5. Now create a custom code function.
6. Add a **message** input string parameter to write to the log.
7. Now add this using code to the function:

```
using Microsoft.Extensions.Logging;
using Ice.Logging;
```

Epicor Functions DO NOT support some C# code, so you must enter these using statements.

8. Enter your logger. For example:

```
var logger = Ice.Logging.ApplicationLoggerBuilder.CreateDefaultBuilder(this.Session,
"LogId")
  .ConfigureFile(options => options.MessageOptions.TimestampFormat =
TimestampFormat.DateTime)
  .Build();

using (logger)
{
  logger.LogInformation(this.message);
}
```

9. When you run a log using Application Insights, you define these application settings for the log in the **Server/AppServer.config** file. For example:

```
<configuration>
  ...
  <ApplicationLog minimumLevel="Information">
    <File enabled="true" />
    <ApplicationInsights
      enabled="true"
      connectionString="InstrumentationKey=aaaaaaaa-aaaa-aaaa-aaaa-
```

```
aaaaaaaaaaaa;IngestionEndpoint=https://centralus-
0.in.applicationinsights.azure.com/;LiveEndpoint=https://centralus.livediagnostics.mon
itor.azure.com/" />
  </ApplicationLog>
</configuration>
```

If you DO NOT configure Application Insights settings in this file, the log file generates using the default settings.

10. Test the different cases you want to track in the logger.

Data Management Tool (DMT)

The Data Management Tool (DMT) is a utility you use to import data into your current Kinetic database. You typically use this tool when you are upgrading to a new Kinetic version and you want to make sure your data moves into the database without errors. You can also use this tool to import legacy data from another system into your Kinetic database.

The DMT is included as part of **Kinetic Power Tools**. When you install this tools package and indicate your user account can access the Data Management Tool, you can launch the DMT from the Kinetic Power Tools menu.

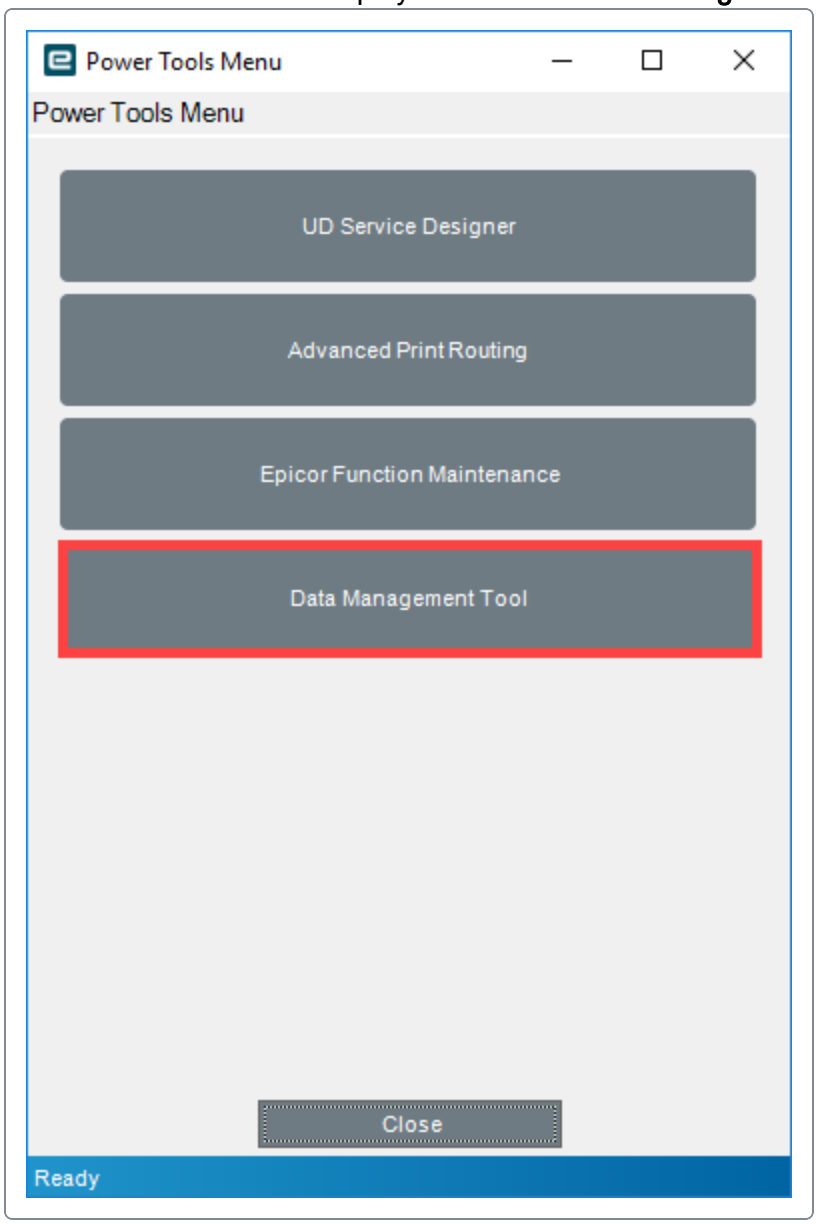
Using the Data Management Tool (DMT)

The **Data Management Tool**, or DMT, enables you to import and validate data into a target database. You import records by adding, deleting, or updating data. This ensures system security, data integrity, and performance.

Logging into the Tool

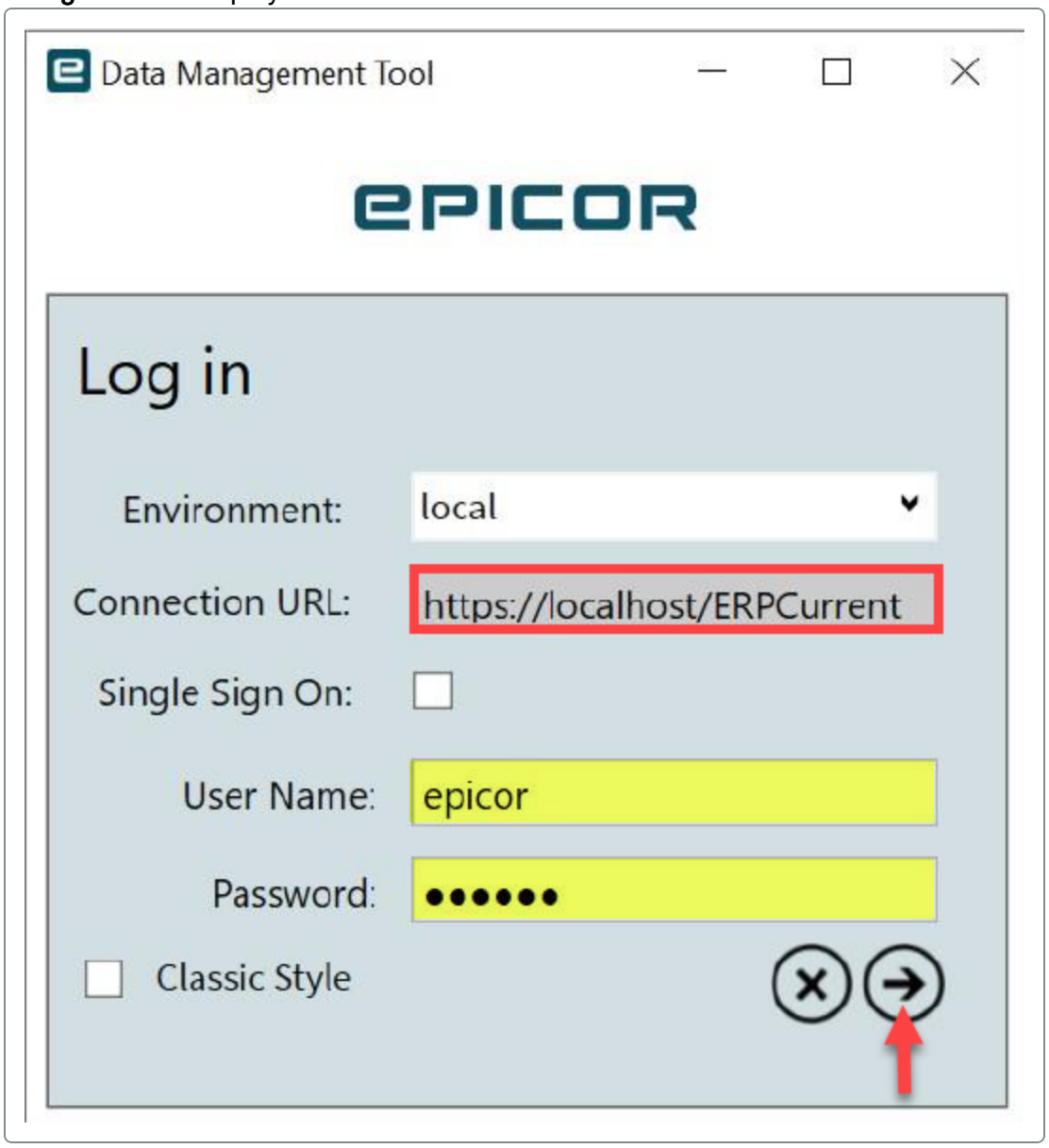
1. Launch the **Kinetic Power Tools** client.
2. Log in with your Kinetic user account.

3. The **Power Tools Menu** displays. Press the **Data Management Tool** button.



If this button IS NOT active, the DataManagementTool license might not be active in your Kinetic environment. Your user account may also need rights to the Data Management Tool. Contact your system administrator to verify that the Data Management Tool license is active and request that you need **DMT User** rights added to your user account.

4. A **Login** screen displays:



5. Select the **Connection URL** file for the environment to which you want to connect.
6. Enter your **Username** and **Password**.

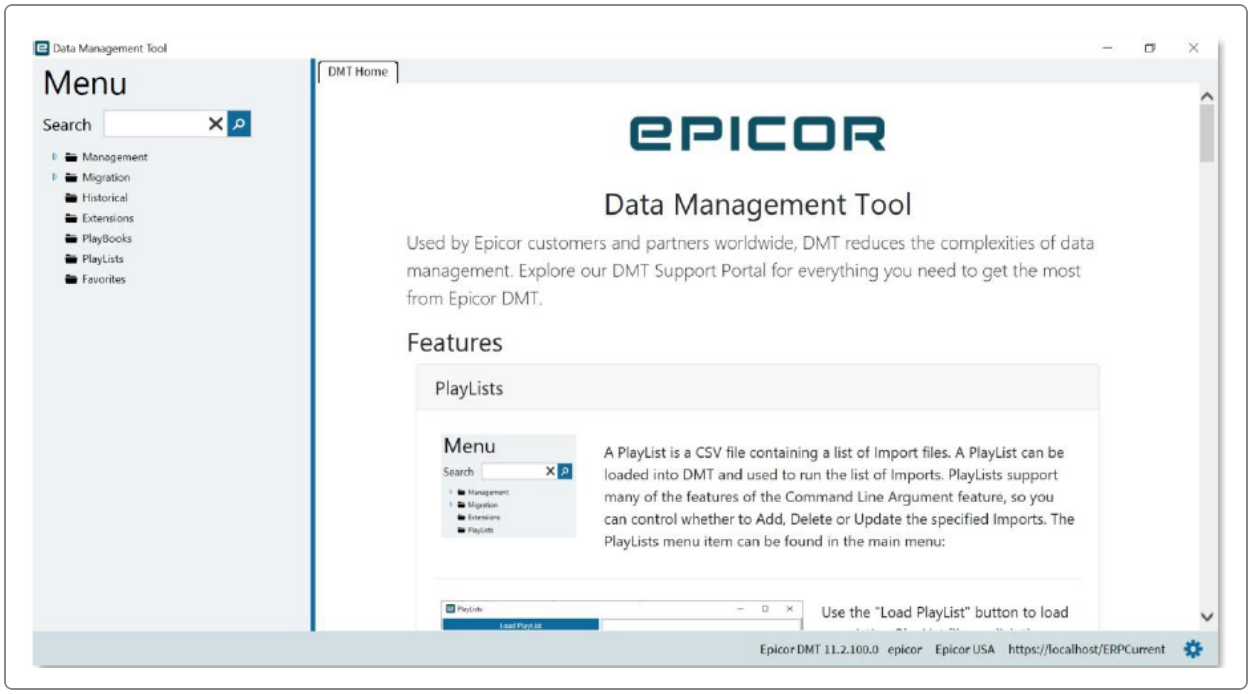
> The DMT remembers your choices. The next time you run the DMT, these default values populate the Log in window fields.

7. Click the **Right Arrow** icon.

The **DMT** will check that you have valid license. If it finds a license, the **Main Menu** will open.

Reviewing the Main Menu

The Main Menu displays all the **Imports** you can use to manage your data using the DMT:



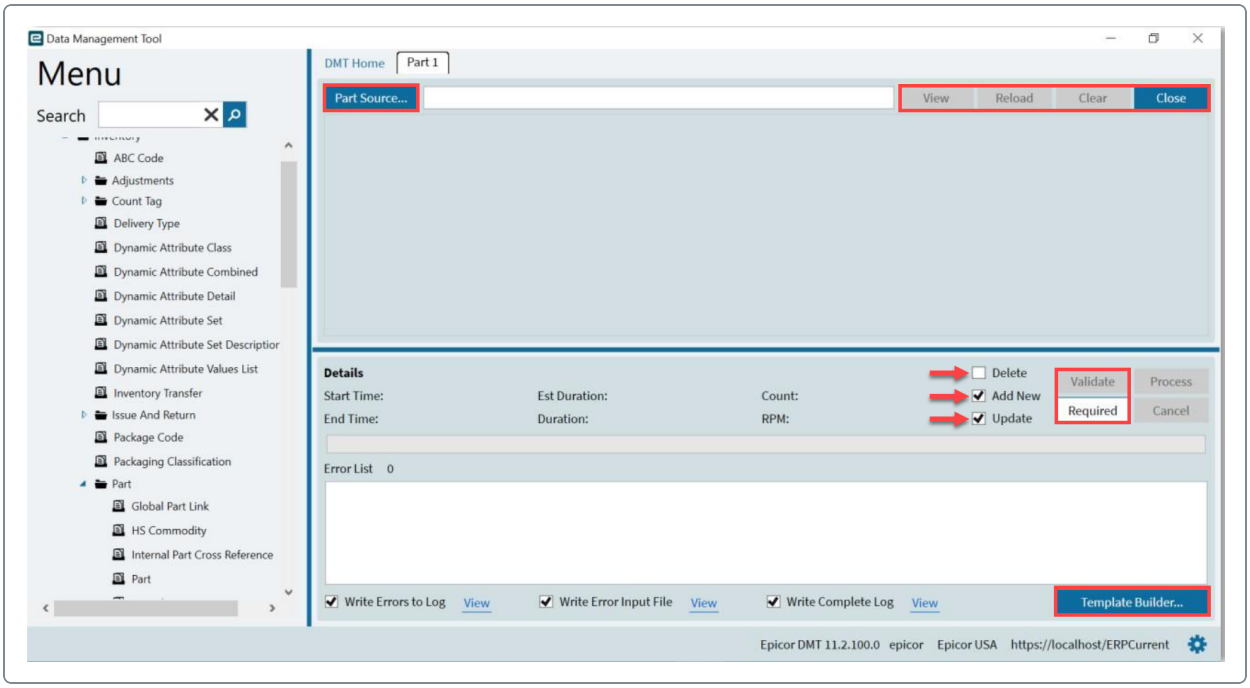
The **Status Bar** shows the DMT version, current user, company, and the connection URL for the selected environment.

Although the DMT will use this company as the default for the Imports, you can override this value by entering a different company ID in the **Import** data.

Reviewing the Imports

The **DMT** provides Imports for several entities. When you select a menu, such as **Inventory**, the DMT displays the available Imports and this menu item's additional sub-menus.

The **Part** import displays below as an example:



All Imports have a similar interface:

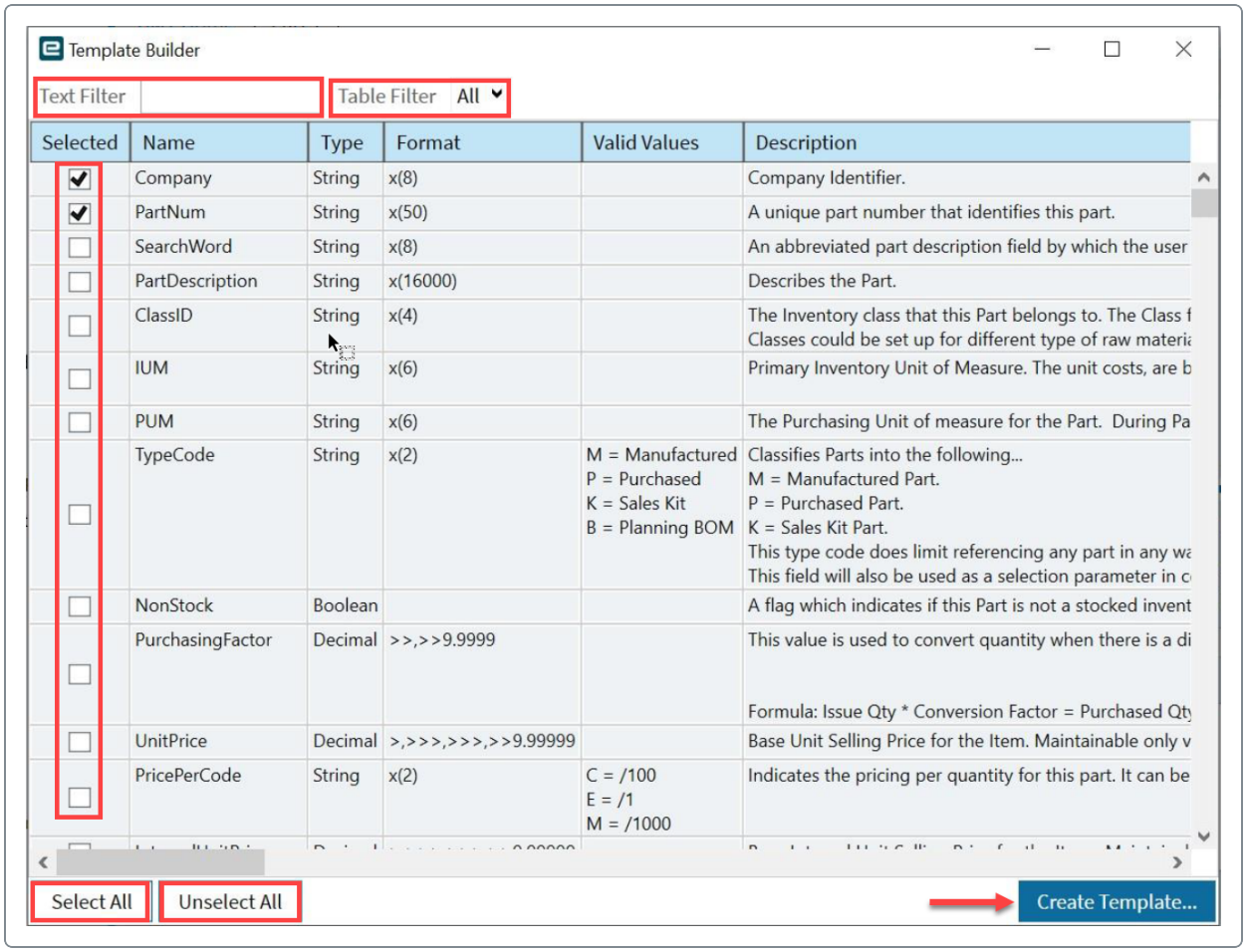
- Select**Source...** (in this example, Part Source) to select the file (xls, xlsx, or csv) containing the data you want to process. This data then displays for your reference.
- Press **View** to open the data source file for editing. If you make any changes, **Save** the file then press **Reload** to load the revised data into the DMT.
- Press **Clear** to unload any loaded data.
- Select **Close** to close the process.
- Select **Validate** to make sure the column names are correct.
- Press **Required** to list the minimum columns the import needs to run.
- Use **Template Builder** to build a csv file by selecting required columns. You use this csv file for the import.
- Select **Delete**, **Add New**, and/or **Update** to control which functions the process will perform.

While the import runs, its statistics display, including an estimate of the total time it will take to process the file.

You can run multiple imports at the same time; these imports are available from the **Imports** drop down menu.

Using the Template Builder

The template builder can help you prepare a data source for the DMT. When you open the template builder, DMT will load the data definitions for the data set that you are processing:



Do the following to create the template:

1. **Select** the fields you wish to include in the template.
2. The template selects any columns required for that import.
3. Use the **Select All** and **Unselect All** buttons to help select the columns you need.
4. Review the information. You can see the details of the column data types, formats, and valid values. You also see a description of the data.
5. To help you find the columns you want to include, use the **Text Filter** and **Table Filter**.
6. When you ready, select **Create Template**.

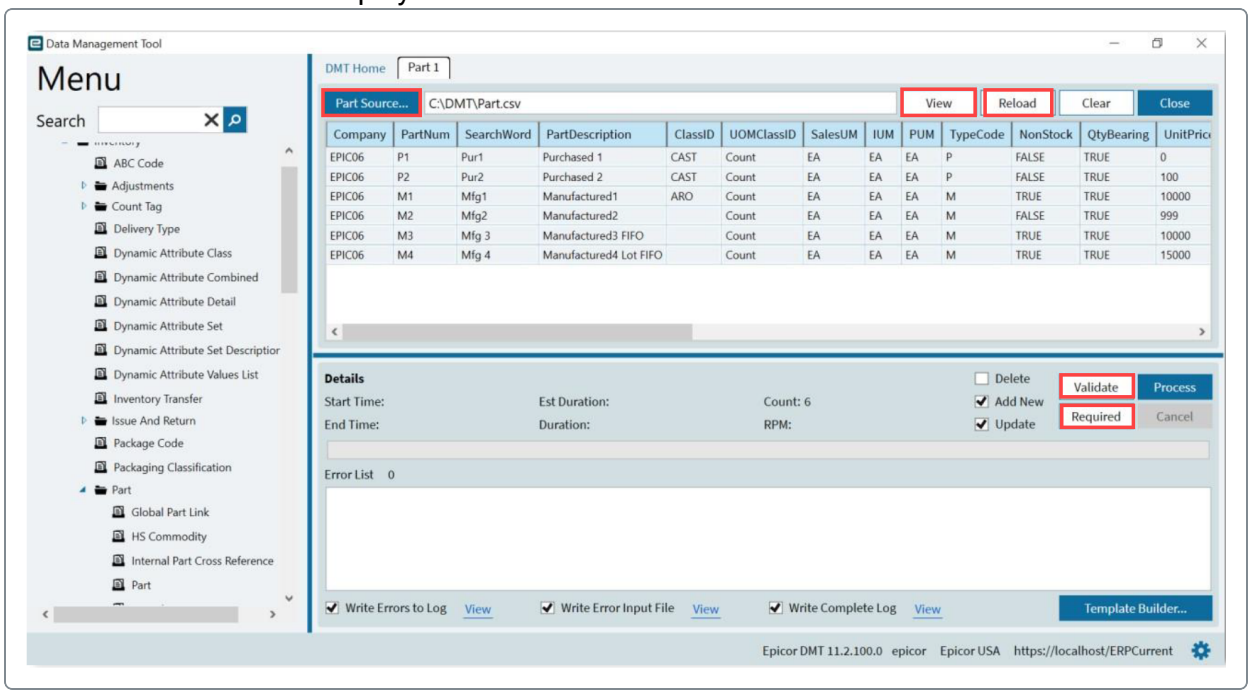
7. The DMT asks you to **Save** the file.
8. Open the file using your preferred editor to populate the template with your data.

While you do not need to use the template builder to create the data source file, it can help you identify column names and formats you want in the data file.

Loading Your Data

After you create your data source either manually or with the Template Builder, you can then load your data into the DMT. To do this:

1. Select the **Source...** button (Part Source displays in this example).
2. Select the file. The data displays in the DMT for review:



3. Press **Required** to see a list of the mandatory columns for the selected process.
4. Press **Validate** to make sure your column headers are correct.

> Using these buttons is optional, but they can help you prevent errors that can affect every row in your data.

5. If you need to make data changes, press **View**. This opens the data source for the registered application for the file type (such as Excel).
6. When you finish editing, **Save** your changes.
7. Now press **Reload**.

Your revised data now displays in the DMT.

Processing Your Data

You next process the data. Use the following check boxes to select the operations you want to apply while you process the data:

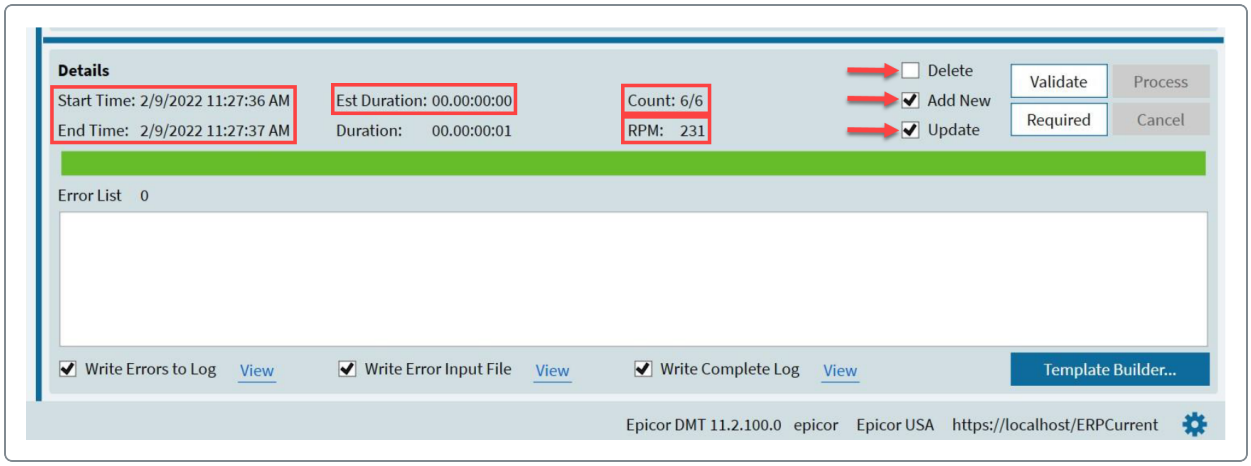
- **Delete**- Removes the listed rows from the Kinetic database.
- **Add New** - Adds the rows as new data to your Kinetic database.
- **Update** - Refreshes the Kinetic data with the selected database records.

Selecting both Add New and Update will update the data for any row that already exists and add a record for any row that DOES NOT already exist.

Depending on the import you selected, some of these options may not be available.

Statistics

While the process is running, DMT will display real-time statistics and a progress bar.

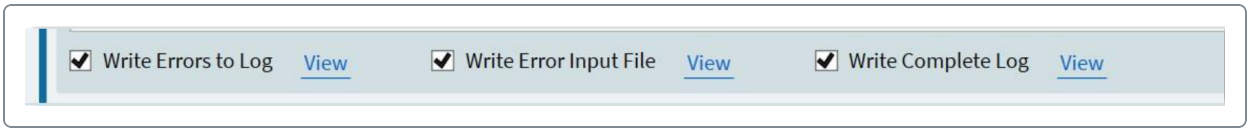


The **RPM** value shows how many records per minute the DMT is processing. the DMT multiplies this value with the total record **Count** to calculate the **Estimated Duration** for the whole process. You can also view the **Start Time**. When the process finishes, the **End Time** will display as well.

As the DMT processes data, it uses Kinetic business objects. This makes sure the tool runs and checks for errors in same way as Kinetic. If the process runs into issues, the they display in the **Error List**. This list also shows a count of the number of rows that contain errors.

Generating Output Files

You select which output types DMT will produce for each process.



- **Write Errors to Log** - Generates a text file that lists the errors in the error list. This file saves as **SourceFileName.Errors.txt**. The DMT only generates this file if it encounters errors.
- **Write Error Input File** - Creates a copy of the original data that contains only the rows that have errors. This file saves as **SourceFileName.Errors.Reprocess.csv**. The DMT only generates this file if it encounters errors.
- **Write Complete Log** - Creates a summary of the process that includes how many records were processed as well as the details of your installation and environment. This file saves as **SourceFileName.CompleteLog.txt**.

You open these files in the DMT by selecting the adjacent **View** link.

If errors occur, use the Error Input file to correct the errors. You can then use this file as a data source for a subsequent process. The DMT then only reprocesses the rows that have errors.

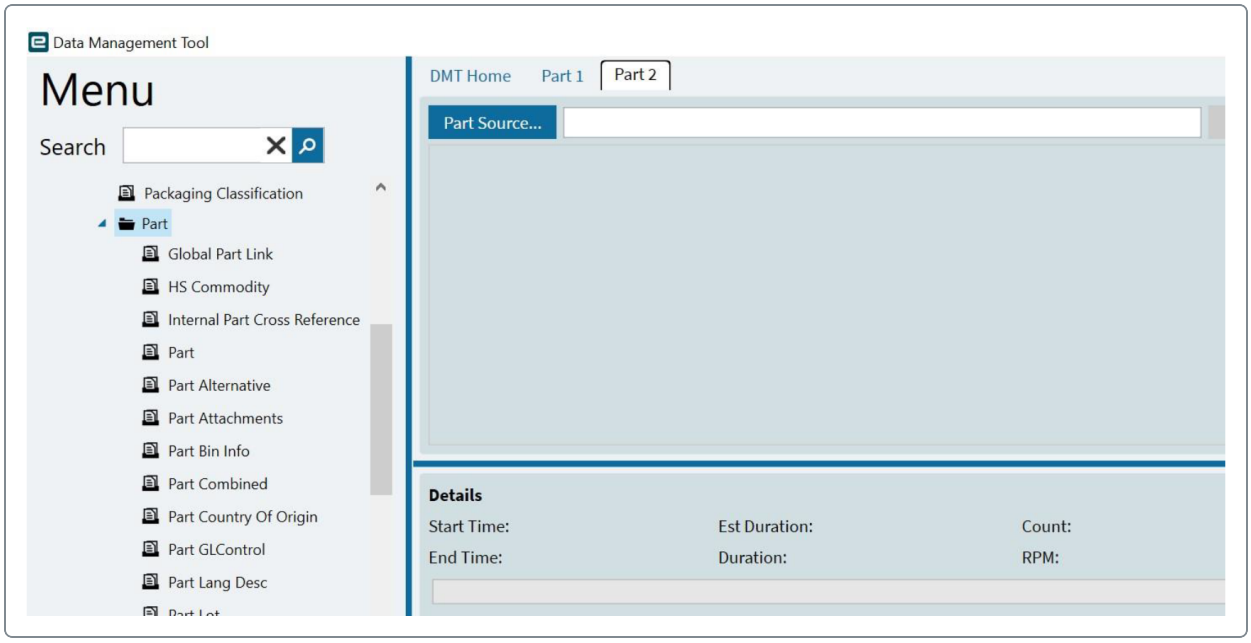
Reviewing Other Features

This section details some additional DMT features.

Concurrent/Multiple Imports

You can run the DMT using multiple clients or within the same client by using multiple, concurrent threads. You can select these separate updates in the Menu. Depending on your hardware, this option can help you gain higher throughput against the database.

For example if you can process many rows, split the source file into several smaller files and then process them in parallel.



Running from the Command Line

You can run the DMT from a command line. Use this alternate process to control how you import Parts, Bills of Materials, and Bills of Operations. You can import this data as a scheduled task.

Learn more about running the DMT from a command line by reviewing the **Using the DMT Command Line** help article and the **Command Line Tools Guide**.

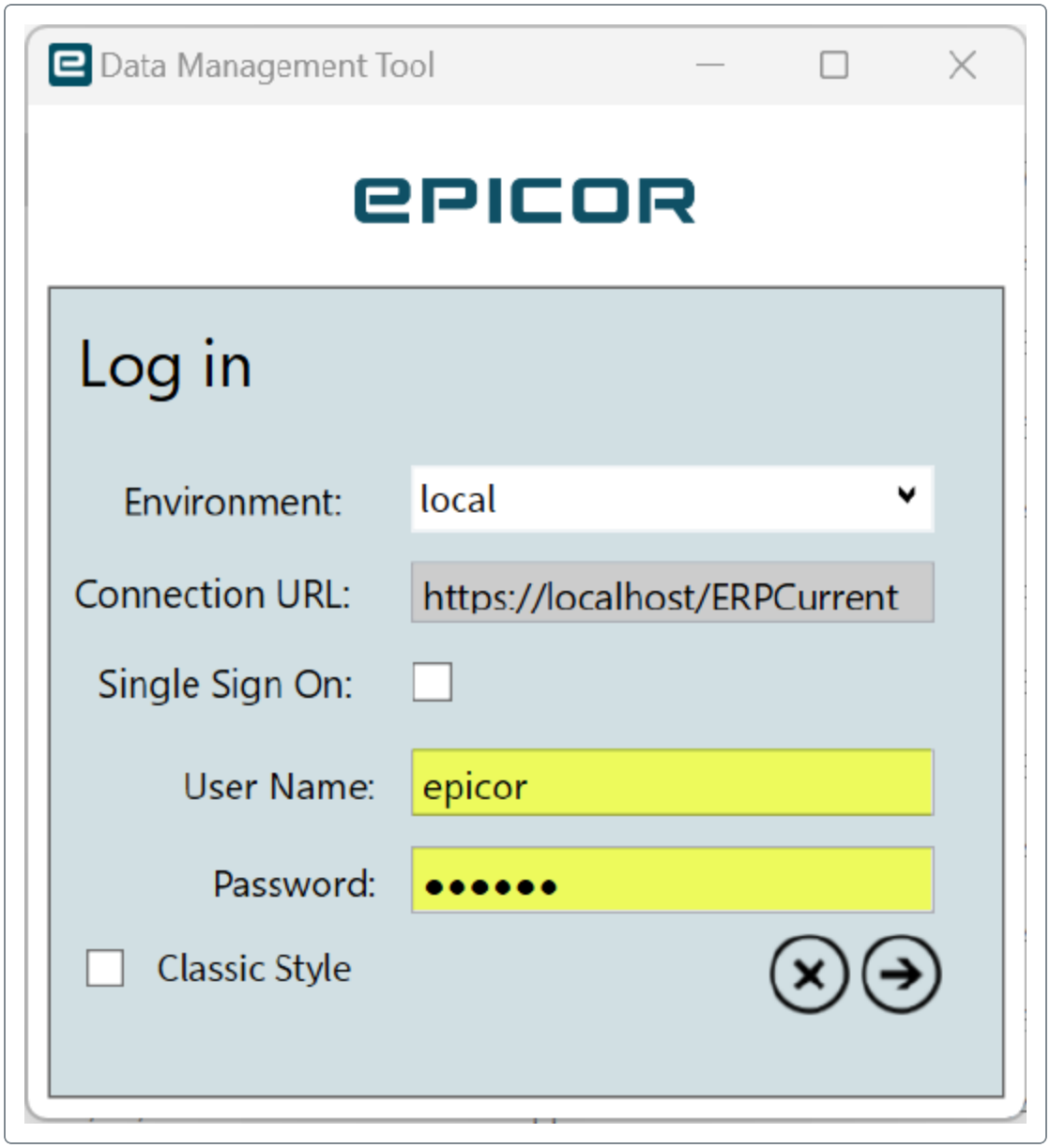
Authentication Modes

You can authenticate the DMT using the **Basic**, **Windows**, **Azure**, and **Epicor Identity Provider** modes.

Basic Authentication Mode

When you authenticate using the Basic mode, you enter your account credentials:

- If you run the DMT by launching the application, enter the **User** and **Password** values in these fields:



- If you run the DMT from the Command Line, you enter these values in the **-User** and **-Pass** options:

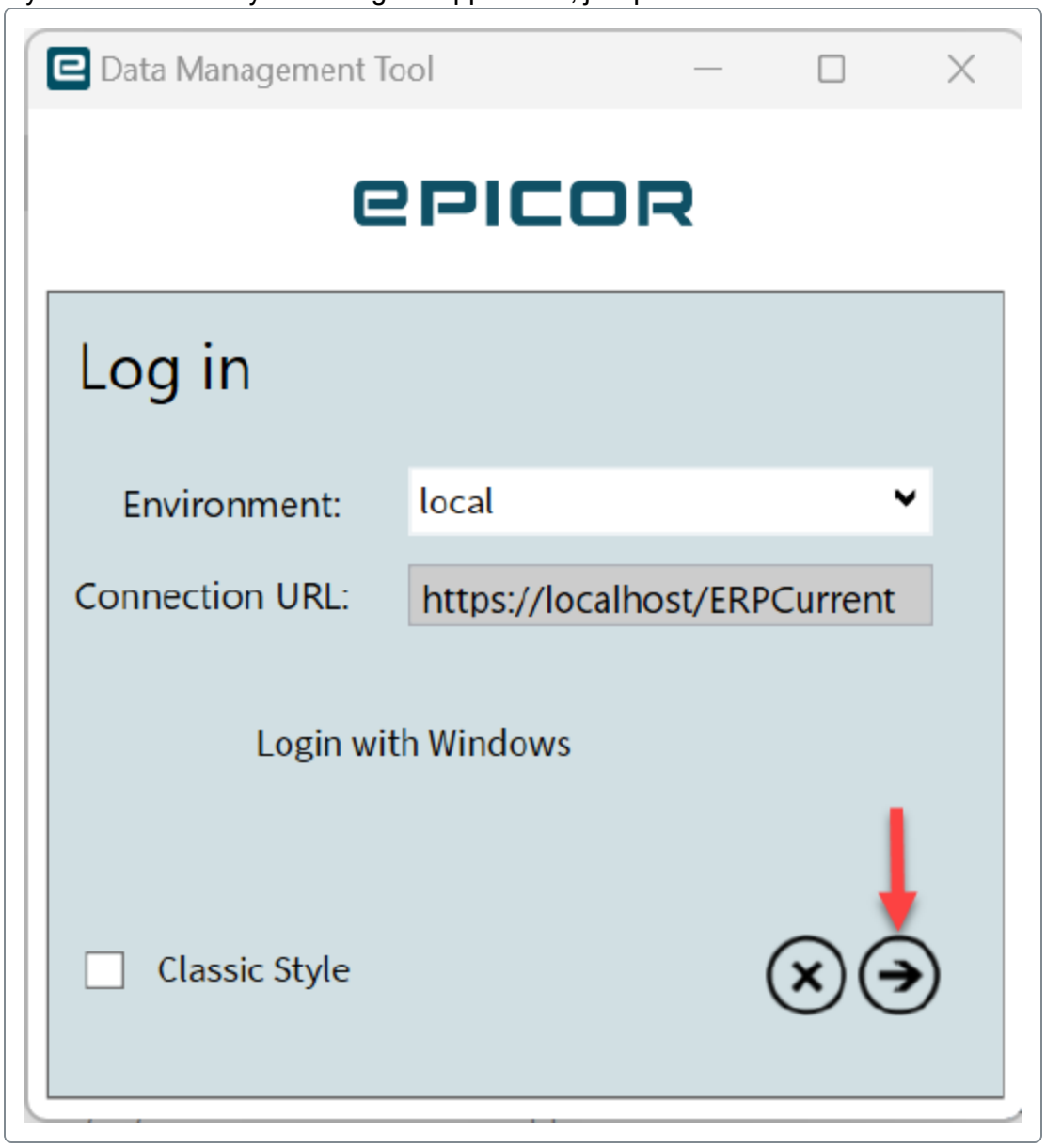
```
C:\_projects\ERP\Current\Deployment\Client>DMT.exe -User=epicor -Pass=epicor -Import="Customer" -Source="C:\DMT Template
s\DMT Customer Templates\Customer.csv" -Add -Update

C:\_projects\ERP\Current\Deployment\Client>
```

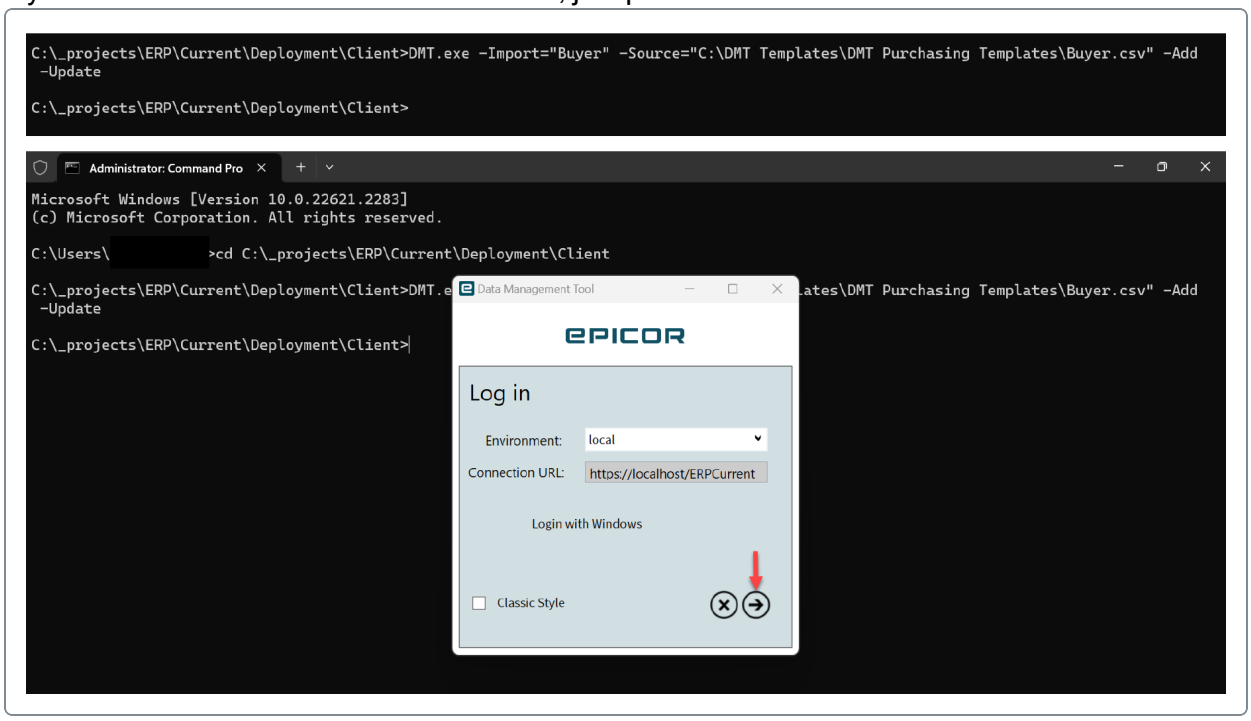
Windows Authentication Mode

When you authenticate using Windows, you DO NOT enter your User and Password. The DMT instead uses the values from your Windows credentials.

- If you run the DMT by launching the application, just press the Arrow button:

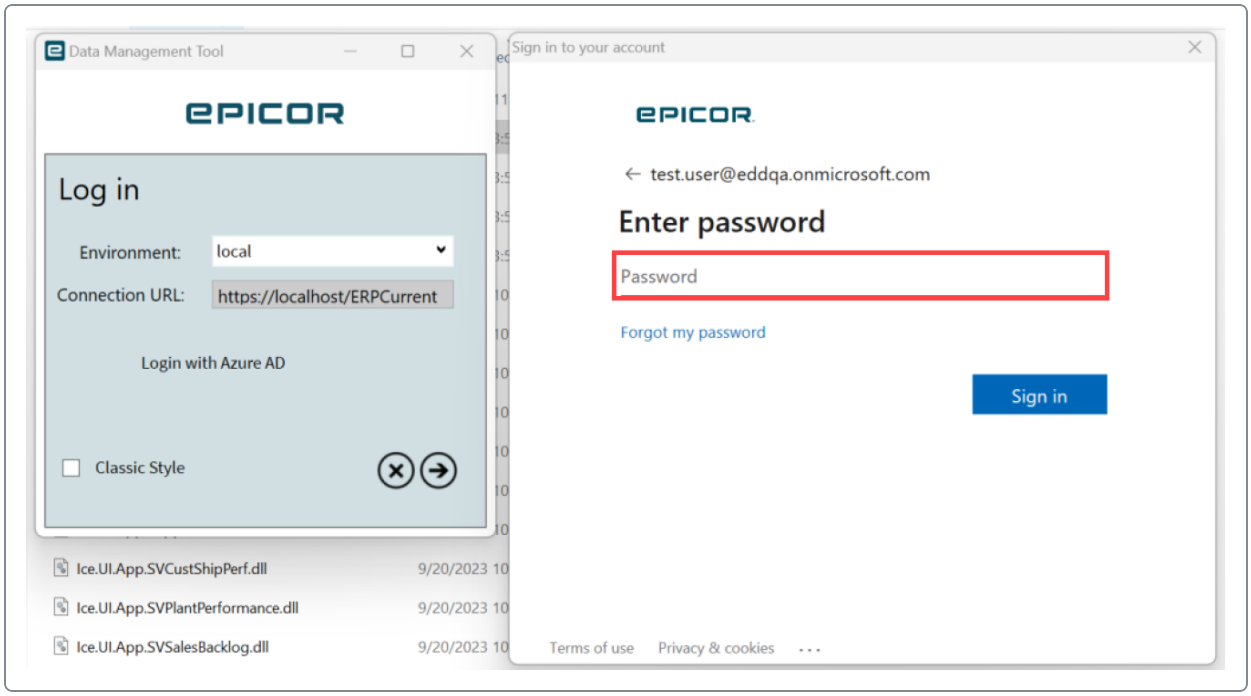


- If you run the DMT from the Command Line, just press the Arrow button:



Azure Authentication Mode

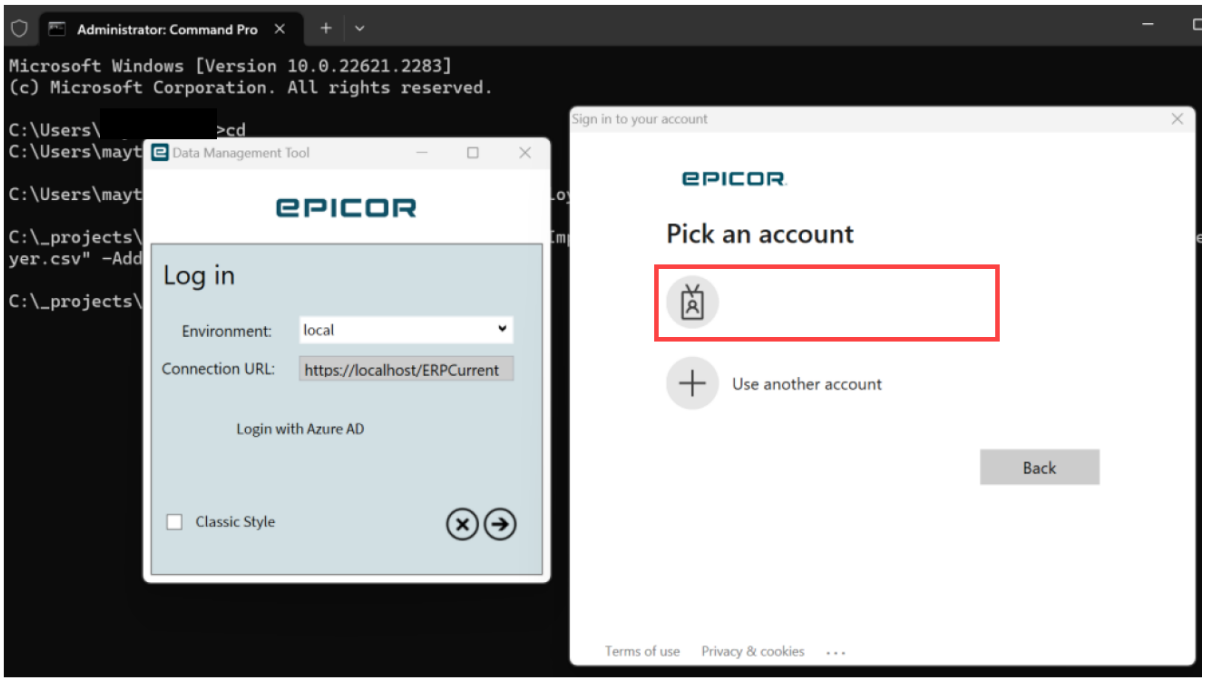
- When login through DMT using Azure Authentication, a window displays that asks for your credentials:



- If you run the DMT from the Command Line, this window also displays that asks for your credentials:

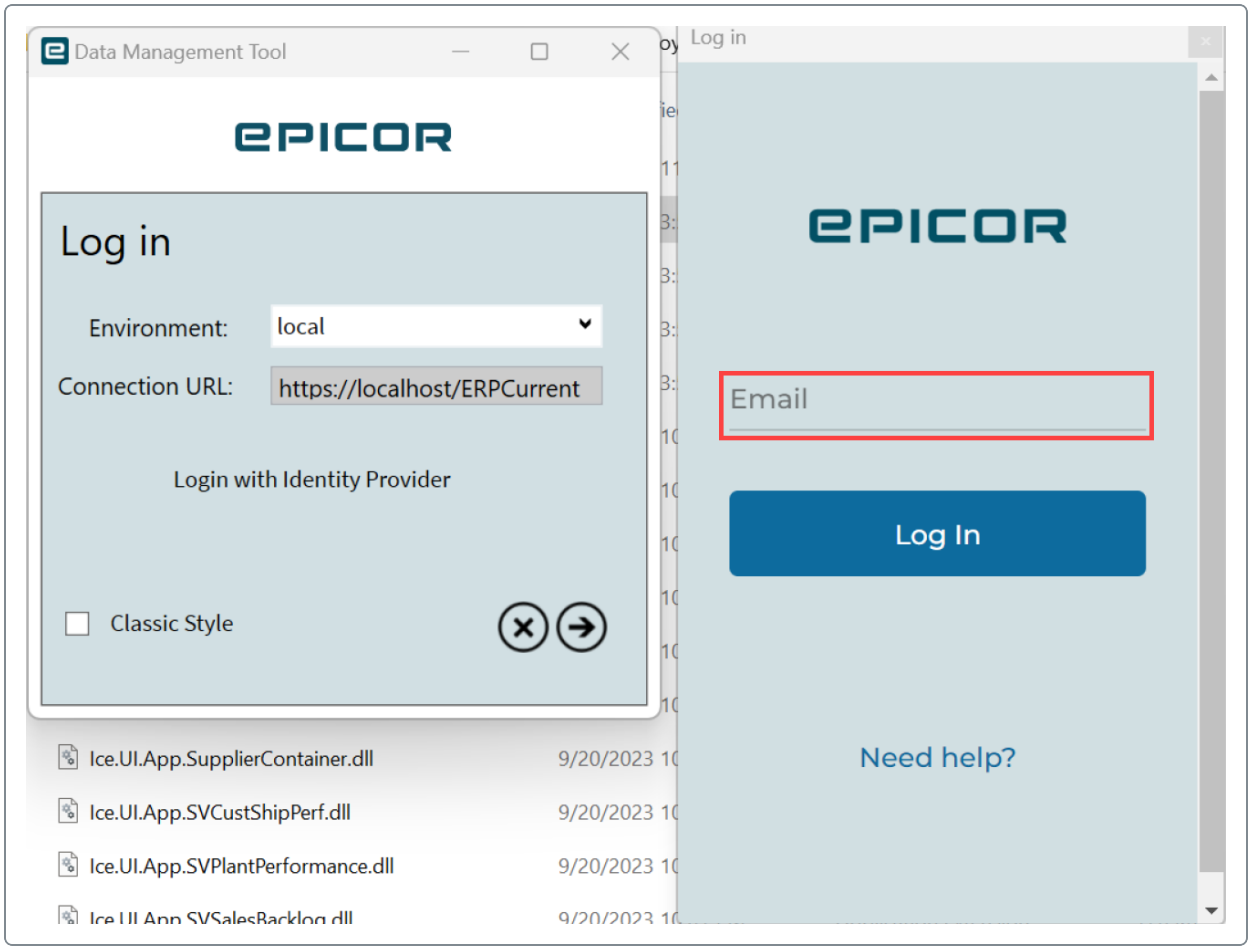
```
C:\_projects\ERP\Current\Deployment\Client>DMT.exe -Import="Buyer" -Source="C:\DMT Templates\DMT Purchasing Templates\Buyer.csv" -Add -Update

C:\_projects\ERP\Current\Deployment\Client>
```

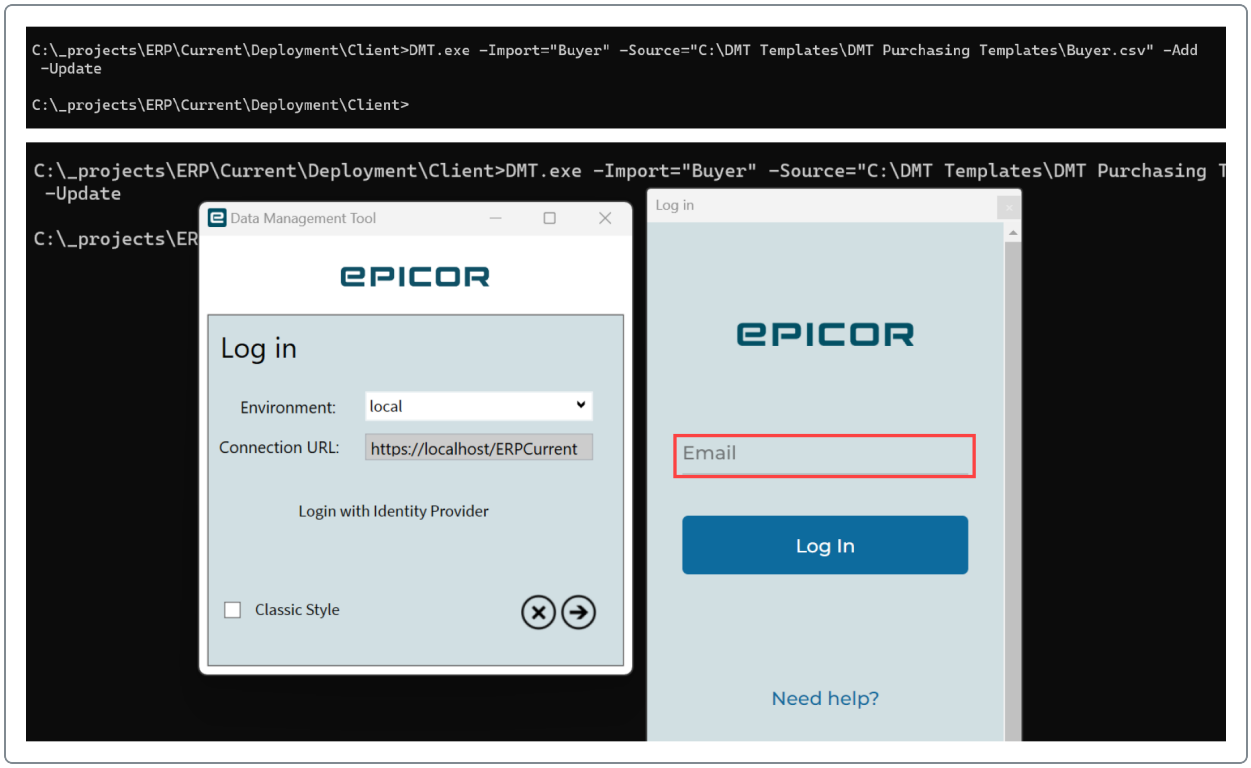


Identity Provider Authentication Mode

- When you login through the DMT using Identity Provider Authentication, a window displays that asks for your credentials:



- If you run the DMT from the Command Line, a window also displays that asks for your credentials:

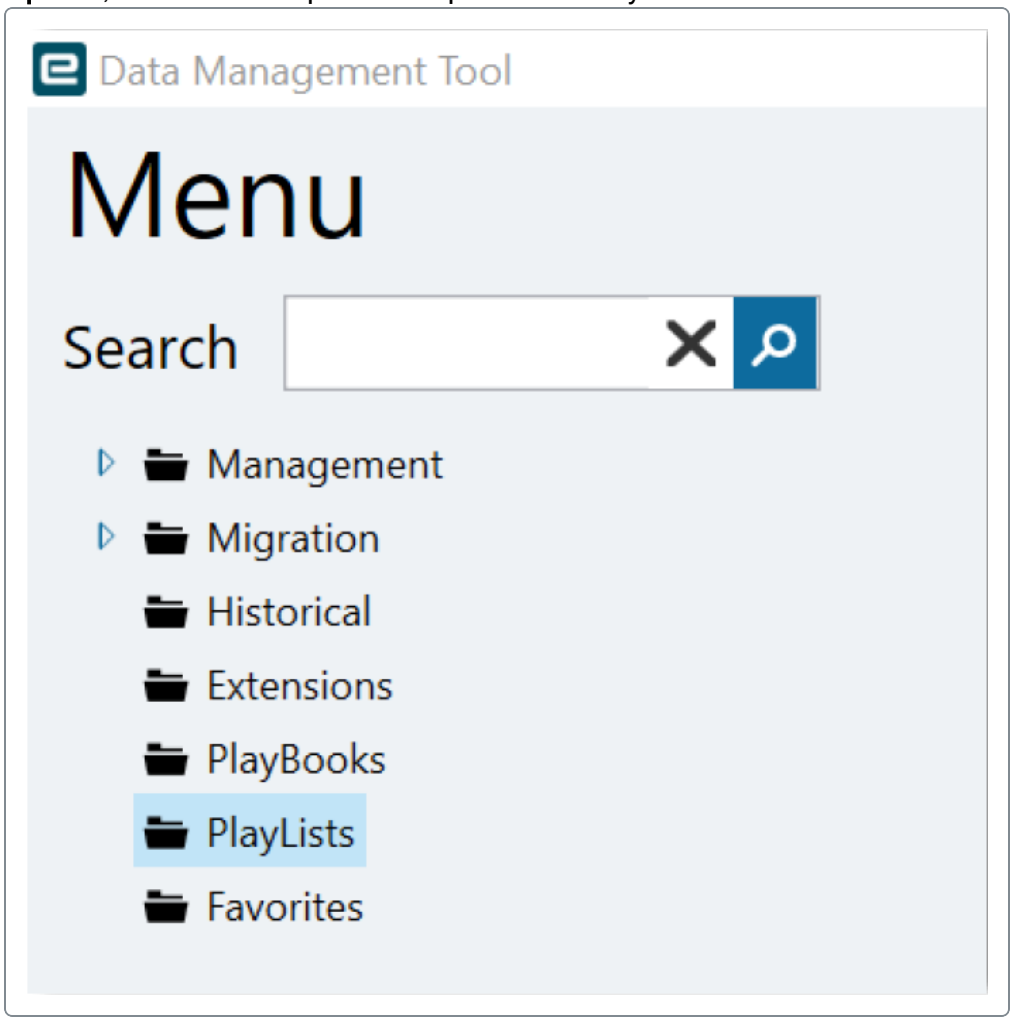

C:_projects\ERP\Current\Deployment\Client>DMT.exe -Import="Buyer" -Source="C:\DMT Templates\DMT Purchasing Templates\Buyer.csv" -Add -Update
C:_projects\ERP\Current\Deployment\Client>
C:_projects\ERP\Current\Deployment\Client>DMT.exe -Import="Buyer" -Source="C:\DMT Templates\DMT Purchasing T -Update
C:_projects\ER
Data Management Tool
EPICOR
Log in
Environment:
local
Connection URL:
https://localhost/ERPCurrent
Login with Identity Provider
Classic Style
Log in
EPICOR
Email
Log In
Need help?

Creating PlayLists and PlayBooks in the Data Management Tool (DMT)

This help article will show you how to create and run **PlayBooks** and **PlayLists** using the **Data Management Tool (DMT)**.

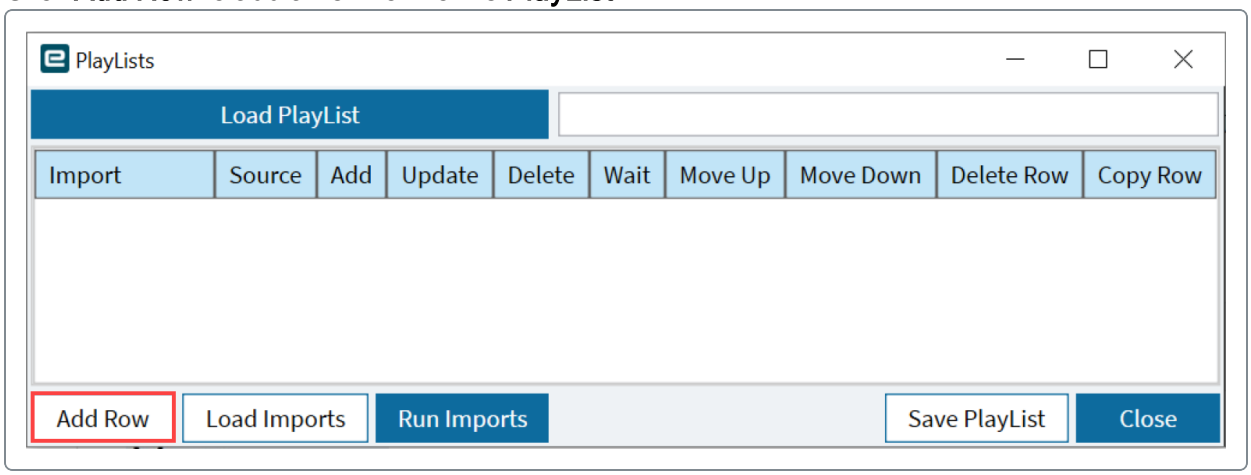
PlayLists

A **PlayList** is a CSV file containing a list of **Import** files. A **PlayList** can be loaded into DMT and used to run the list of **Imports**. PlayLists support many DMT features, as you can control whether to **Add**, **Update**, or **Delete** the specified Imports. The PlayLists menu item is on the **Main Menu**:



Creating a PlayList

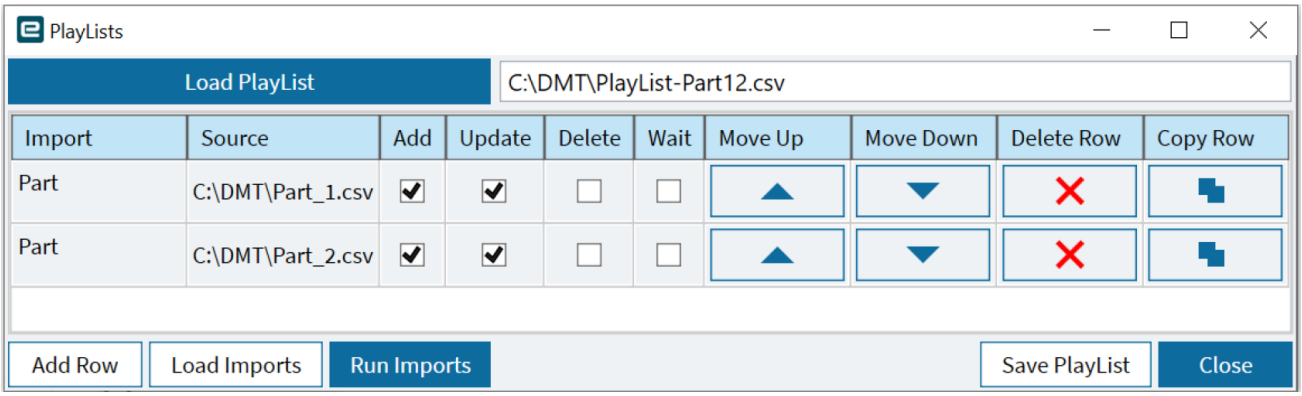
1. Click **Add Row** to add a new row to the **PlayList**.



2. Click the **Import** field to display a drop-down list of **DMT Imports**. Select the required Import or start typing to display matching Imports.
3. Click the **Source** field to display a file dialog. Select the required Import file and click **Open**.
4. Set the **Add**, **Update**, and **Delete** checkboxes as you need them to appear in the Import.
5. If DMT should wait for this Import to finish before starting the next Import, select the **Wait** checkbox.
6. Repeat steps 1-5 to add more Imports.
7. To change the order of Imports in the PlayList, use the **Move Up** and **Move Down** buttons.
8. To remove an Import from the PlayList, use the **Delete Row** button.

> This does not delete the file; it just removes the line from the PlayList.

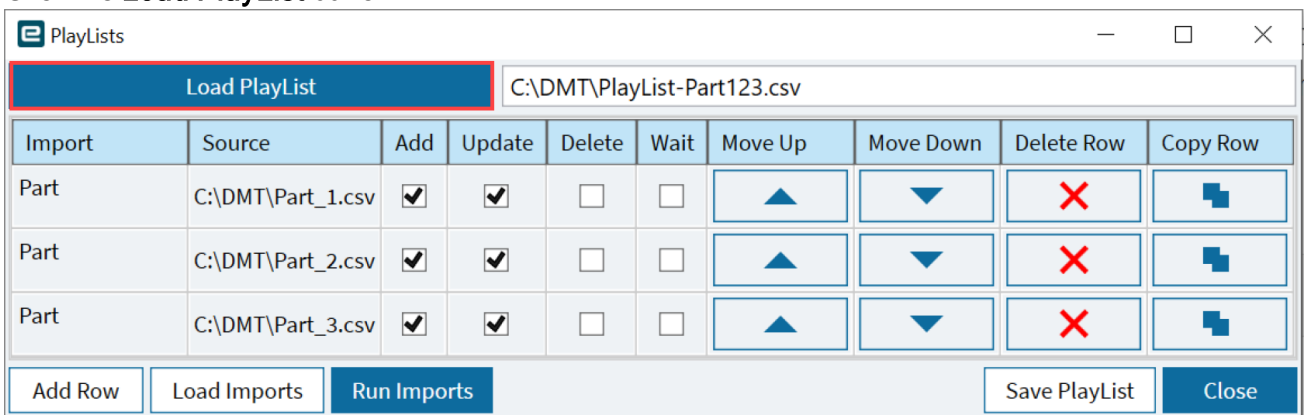
9. When the PlayList is ready, click **Save PlayList**.
10. The **Save As** file dialog appears. Enter a **Name** for this PlayList.
11. Press **Save**.



This saves the **PlayList** and its name appears in the **Load** textbox.

Loading an Existing PlayList

1. Click the **Load PlayList** button.



2. The **Open** file dialog appears. Select the **PlayList** you wish to open.
3. Click **Open**.

The selected **PlayList** is loaded into the **PlayLists** window. From here you can **Modify**, **Load**, **Run**, and **Save** the PlayList.

Modifying a PlayList

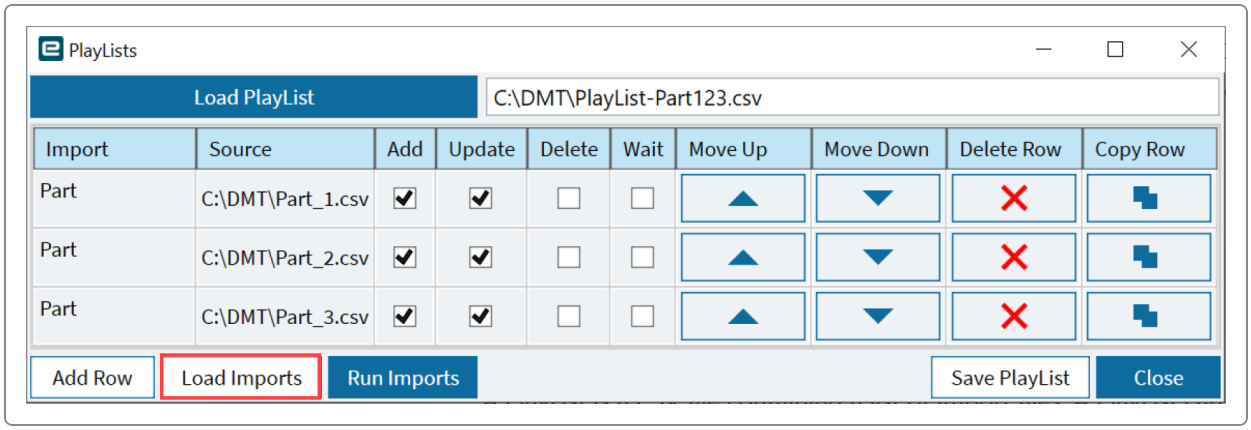
1. To change the order of Imports in the PlayLists, use the **Move Up** and **Move Down** buttons.
2. To remove an Import, select the **Delete** check box.

> This DOES NOT delete the import file, it just remove the line from the PlayList.

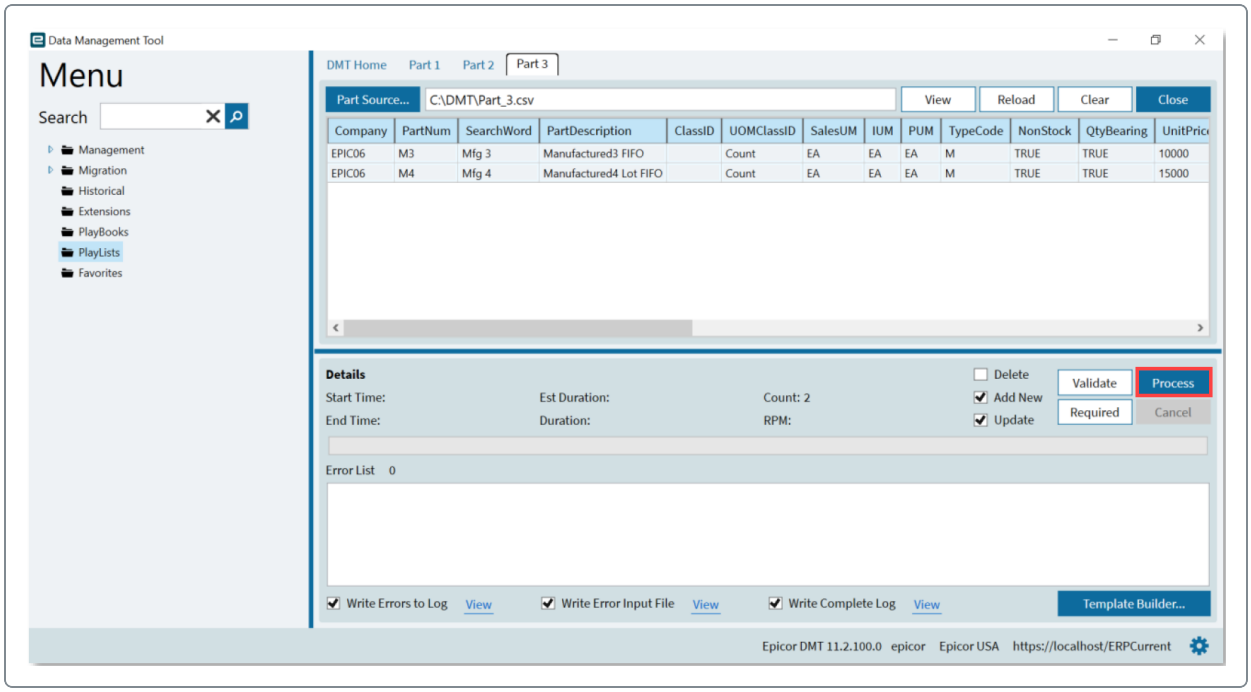
3. When ready, click **Save PlayList**.
4. The **Save As** file dialog appears. Enter a **Name** for this PlayList.
5. Click **Save**.

Loading all the Imports in a PlayList

The **Load Imports** button pulls in all the Imports in the PlayList but DOES NOT run them:

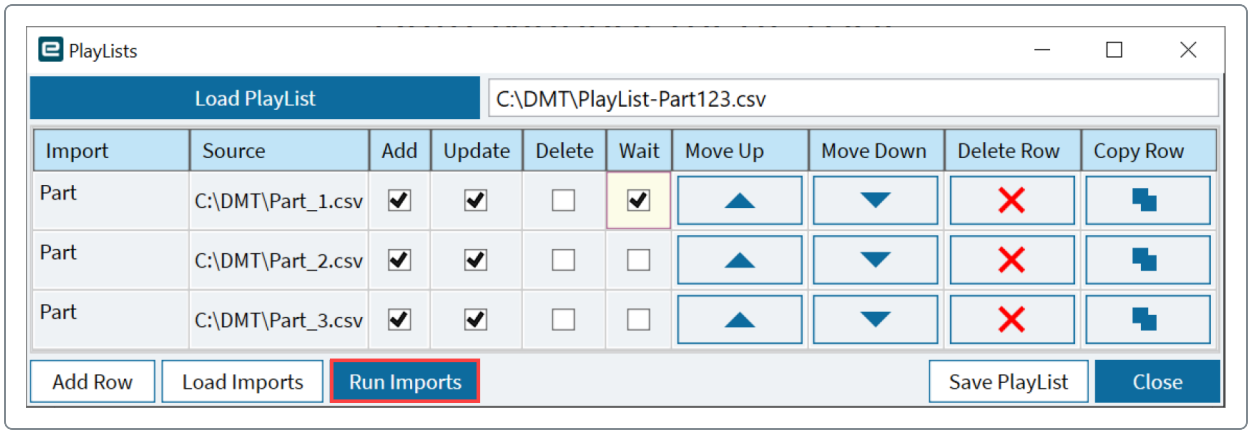


You can now manually run the Imports. Do this by clicking the **Process** button:



Run all the Imports in a PlayList

The **Run Imports** button runs all the Imports in the PlayLists using the order in which they appear on the list:



If you select the **Wait** check box, then DMT will wait for this Import to finish before it starts the next Import process. In the above example, the first Import will run. When it finishes, the second and third Imports will run concurrently.

PlayList Command Line Arguments

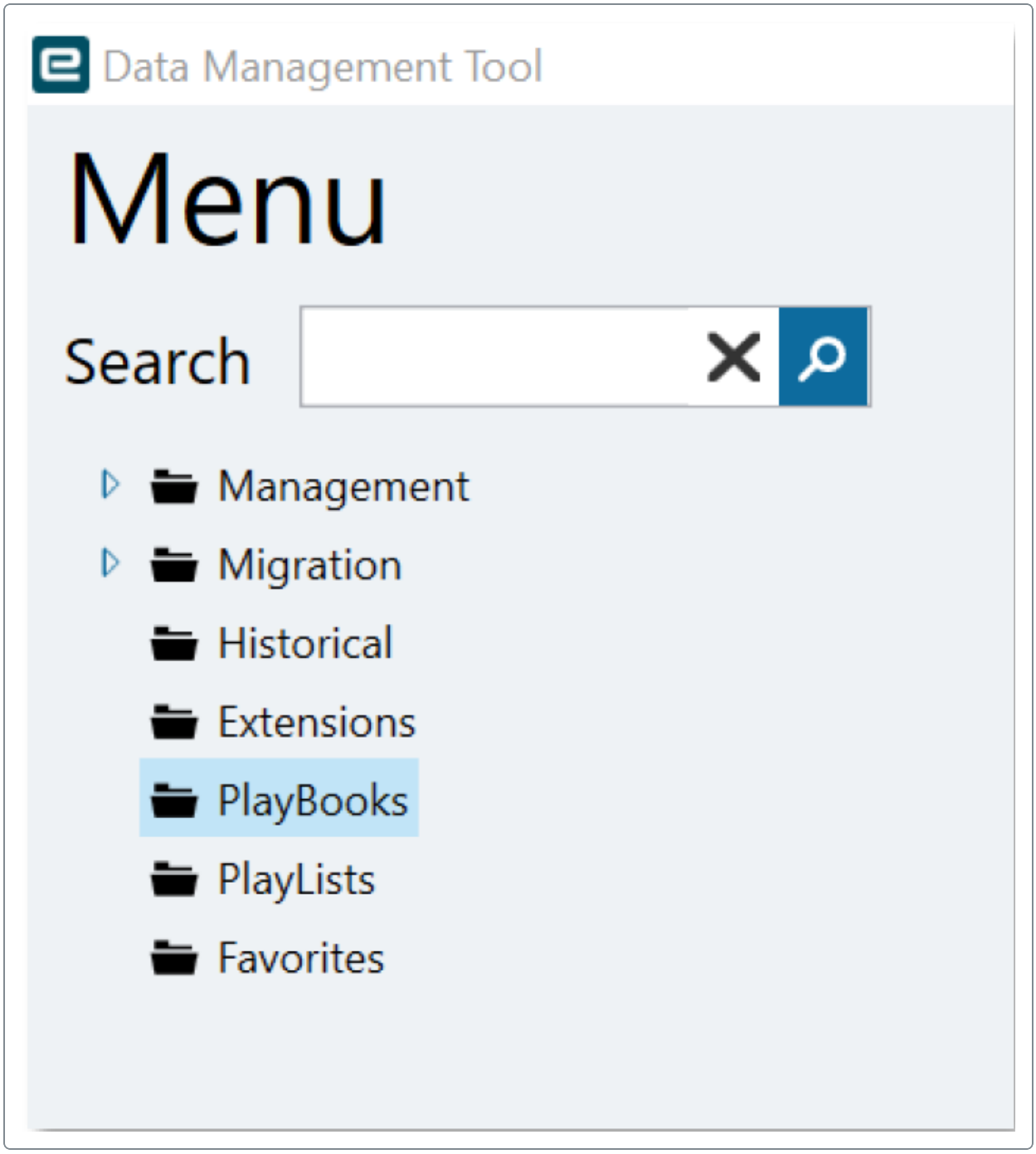
Playlists can be run from the command line using the **-PlayList** argument. The following command will launch DMT and run all the Imports in the specified PlayList:

```
DMT.exe -User=manager -Pass=password -PlayList="C:\DMT\PlayList-Part123.csv" Con-
figValue=local.sysconfig
```

The **Wait** command is not implemented in Command Line mode, so all Imports will run at the same time.

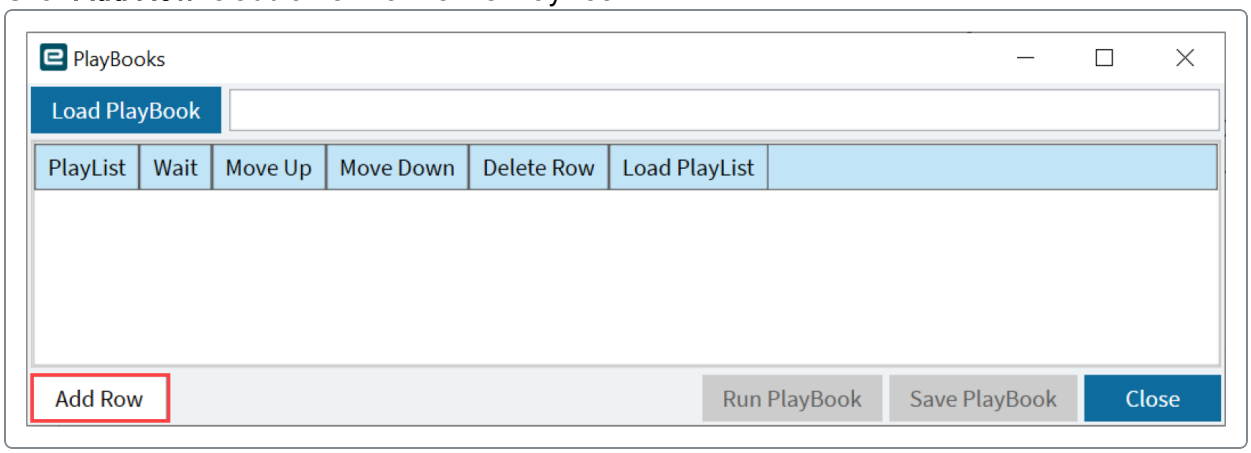
PlayBooks

A **PlayBook** is a text file that contains a list of PlayList files. You load a PlayBook into DMT to both load or run PlayLists. The PlayBooks menu item is on the **Main Menu**.


Data Management Tool
Menu
Search
Management
Migration
Historical
Extensions
PlayBooks
PlayLists
Favorites

Creating a PlayBook

1. Click **Add Row** to add a new row to the PlayBook.

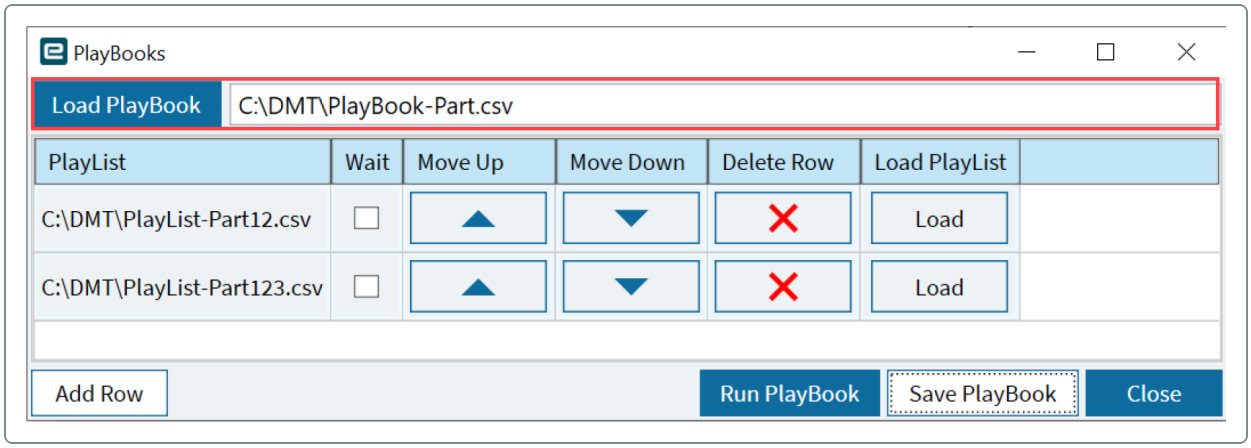


2. Click the **PlayList** field to open a file dialog.
3. Select the required **PlayList** and press **Open**.
4. Repeat steps 1-3 to add more PlayLists to the PlayBook.
5. To change the order of the PlayLists in the PlayBook, use the **Move Up** and **Move Down** buttons.
6. To remove a PlayList from the PlayBook, use the **Delete Row** check box.

> This does not delete the PlayList file; it just removes the line from the PlayBook.

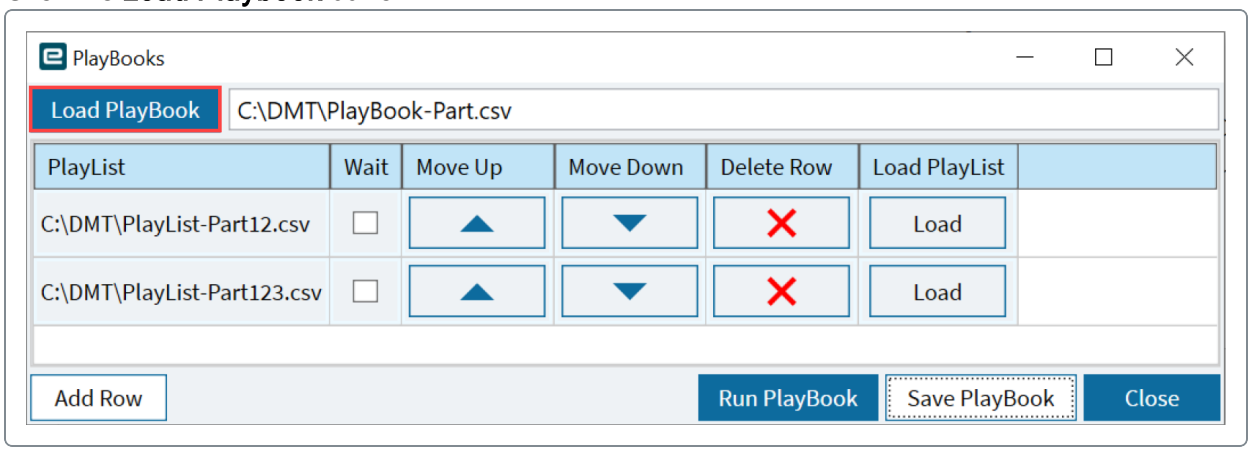
7. When ready, click **Save PlayBook**.
8. The **Save As** file dialog appears. Enter a **Name** for this PlayBook.
9. Press **Save**.

This saves the **PlayBook** to the location you selected. Its name appears in the **Load PlayBook** textbox:



Loading an Existing PlayBook

1. Click the **Load Playbook** button.



2. The **Open** file dialog box appears. Select the **PlayBook** you wish to open.
3. Press **Open**.

Modifying a PlayBook

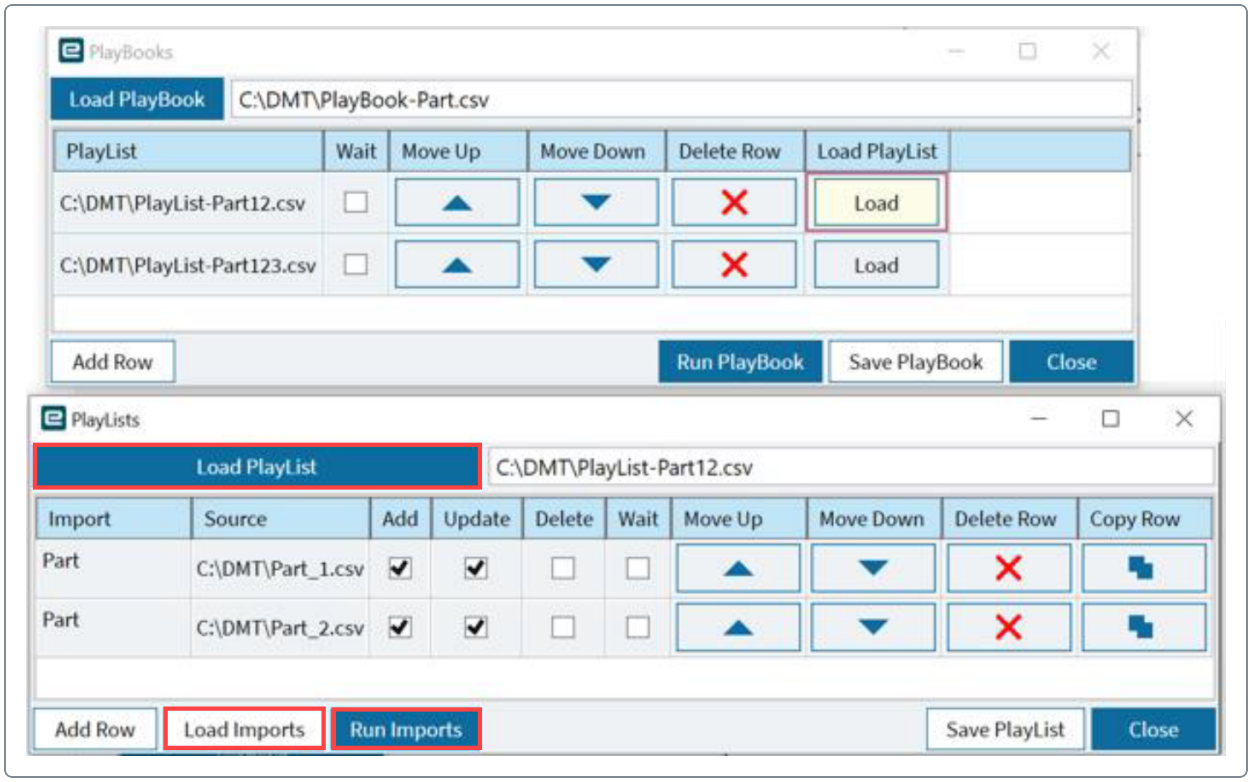
1. To change the order of PlayLists in the PlayBook, use the **Move Up** and **Move Down** buttons.
2. To remove a PlayList from the PlayBook, select its **Delete** check box.

> This does not delete the PlayList file; it just removes the line from the PlayBook.

3. When the PlayBook is ready, click **Save PlayBook**.
4. The **Save As** file dialog appears. Enter a **Name**.
5. Press **Save**.

Loading a PlayList from a PlayBook

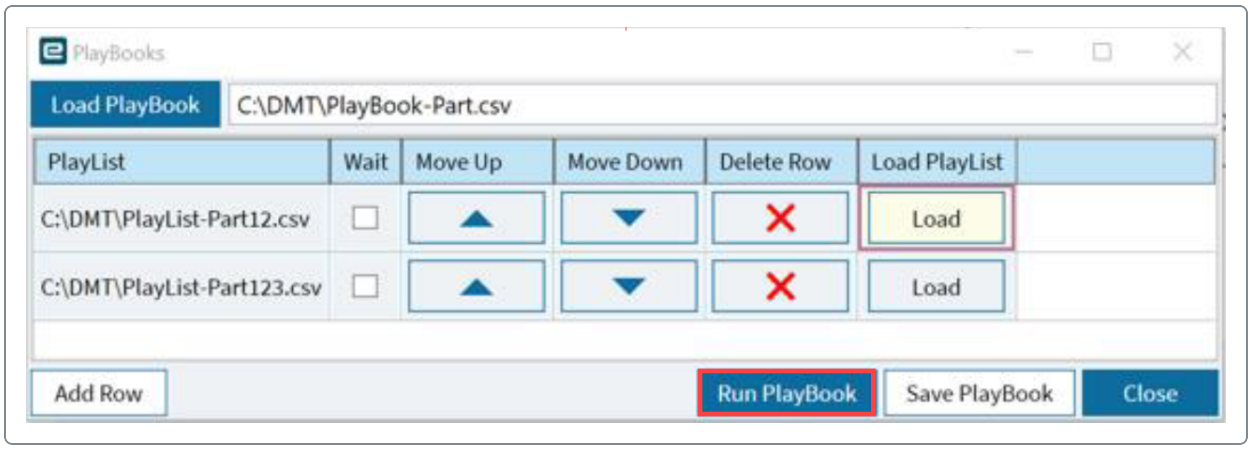
Press the **Load PlayList** button to launch the **PlayLists** window. This loads the selected **PlayList** in the window:



You can then either **load** or **run** the imports from this window.

Run all the PlayLists in a PlayBook

Click the **Run PlayBook** button to run all Imports in all PlayLists that are in the current PlayBook:



PlayBook Command Line Arguments

The **PlayBook** command line argument feature is not implemented yet.

Using the DMT Command Line Interface

While you can run the Data Management Tool (DMT) using its standalone interface (UX), you can also run this tool from the command line. This help article documents how you run this tool.

You can also run other system management tools from the command line. Learn about these tools by reviewing the **Command Line Tools User Guide**.

In this article, we will cover:

Overview

This lab will guide you through using the **Data Management Tool (DMT)** via the command line. Depending on which parameters are supplied, you can use DMT via the command line to automate imports, export data, perform BAQ exports or split files.

Command Line Import

You can start DMT with command line arguments to automate imports. This also allows for more complex automated sequencing of imports to save you time once the imports have been set up.

An example of running DMT from the command line:

```
DMT.exe -User=manager -Pass=manager -Import="Part" -Source="C:\Imports\Part.csv" -Con-
figValue=local.sysconfig -Add -Update
```

The name of the **Import** is the same as that of the menu item within DMT .

By creating and saving these commands in a batch file you can run imports and conditionally check for errors.

Below is an example where **Product Group** will be imported then, if no errors occurred, **Part** will be imported.

```
DMT.exe -User=manager -Pass=manager -Import="Product Group" -NoUI -Source-
e="C:\Imports\ProdGrps.xls" -Add -ConfigValue=local.SysConfigIF ERRORLEVEL 1
GOTO ERROR
DMT.exe -User=manager -Pass=manager -Import="Part" -NoUI -Source="C:\Imports\Part.csv" -Add
-ConfigValue=local.sysconfig
IF ERRORLEVEL 1 GOTO ERROR
ECHO "FINISHED"
GOTO END
```

The above example includes the additional **-NoUI** parameter. This parameter is required if you want to schedule DMT to be run from a scheduled Windows task on Windows Server 2008 and above.

Executing `DMT.exe /?` displays help for running DMT from the command line.

Usage: DMT.exe [options]

- **-User**: Specifies the username DMT will use to connect.
- **-Pass**: Specifies the password DMT will use to connect.
- **-ConfigValue**: Specifies which config file should be used. If not specified, then default.sysconfig is assumed.
- **-sso**: Specifies that Single Sign On is enabled on Epicor and hence auto-login is required.
- **-NoUI**: Runs DMT without a User Interface. You can run it from a scheduled task.
- **-DisableUpdateService**: Disables the DMT update service.
- **-Add**: Indicates that records should be added.
- **-Update**: Indicates that records should be updated.
- **-Delete**: Indicates that records should be deleted.
- **-Trace**: Enables tracing in Epicor ERP.
- **-LoadViaOdbc**: Forces DMT to use ODBC to read import data. Use it if errors occur on data load.
- **-Delimiter**: Sets delimiter for import files.
- **-Import**: Specifies the Import to run.
- **-Source**: Specifies the import source file.
- **-Converter**: Specifies the location of an assembly containing a method to convert the data before importing.
- **-NoCompleteLog**: When set to TRUE, it disables the Complete Log.
- **-NoErrorLog**: When set to TRUE, it disables the Error Log.
- **-NoErrorInput**: When set to TRUE, it disables the Error Input Log.
- **-PlayList**: Executes PlayList through command line with specified import source file.
- **-PlayBook**: Executes PlayBook through command line with specified import source file.
- **-LogDetailedException**: Specifies if the detailed exception should be included in the error log file. Valid values are True or False. Helpful when reporting errors to Kinetic.

Command Line File Splitter

You can use DMT to split a CSV file into multiple files. You must supply three arguments.

This should follow the syntax:

```
DMT.exe -split -input=[InputFile] -files=[FileCount]
```

Example:

```
DMT.exe -split -input="C:\DMT\Part.csv" -files=3
```

This will save the data from **Input.csv** in three new CSV files in the same folder. These files follow this format:

```
C:\DMT\Part_X.csv
```

Options Include:

- **-split**: Required to run the split tool
- **[InputFile]**: The file you wish to split
- **[FileCount]**: How many new CSV files you wish to create (must be a number)

If the paths exist in the same folder as the application, these paths can be relative.

Command Line Export

The DMT command line can also be used to export the data format for each **Import** type.

- **-ExportData**: Exports format of all imports to specified file.
- **-ExportDataFormat**: Set to XML to export imports to XML rather than CSV.

Execution of PlayList through Command Line

```
DMT.exe -User=manager -Pass=manager -PlayList="C:\Desktop\PlayList.csv"
```

Execution of PlayBook through Command Line

```
DMT.exe -User=manager -Pass=manager -PlayList="C:\Desktop\PlayBook.csv"
```

Logging Detailed Exception

```
DMT.exe -User=manager -Pass=manager -Import="Part" -Source="C:\Imports\Part.csv" -Con-
figValue=local.sysconfig -Add -Update -LogDetailedException=True
```

Example Error File (with -LogDetailedException=True):

```
2022-02-09T13:28:53 PartNum: P1 Record already exists.
Record already exists.
at Epicor.DMT.Lib.PartImporter.Import(ImportData data, ExtendedBackgroundWorker worker) in
C:\_projects\ERP\Current\Source\Client\Lib\DMTLib\Inventory\Part.cs:line 103
```

Running the DMTDiff Utility for the Data Management Tool (DMT)

DMTDiff is a command line utility that allows you to compare changes between groups of files. This is useful when you are exporting large amounts of data to be imported into Kinetic via the **Data Management Tool (DMT)** ahead of time.

Hence you extract your data to CSV, then you import this into Kinetic via DMT. You retain the original load files. After a period, you extract the data again, utilize the DMTDiff tool to compare it to the previously loaded data and it will provide you with just the changes, which you can load.

The utility also allows the ability to split and merge data files to aid in the importing and comparison of large or multiple data sets.

This help article will guide you through a lab using DMTDiff via the command line to compare changes between two .CSV delimited files or to split larger .CSV files into smaller ones.

Command Line

DMTDiff can be started with command line arguments to automate the process. This also allows for more complex automated sequencing of the process. The format of the command line is as follows:

```
DMTDiff.exe –base=[BaseFilePath] –cmp=[ComparisonFilePath] –out=[OutputFilePath]
```

The utility requires three parameters along with optional switches which are explained later. The required parameters are as follows:

- **–base=[BaseFilePath]** – Path of the original file that you wish to compare.
- **–cmp=[ComparisonFilePath]** – Path of the file you wish to compare against the base file. Rows that are different in the comparison file will be written into the output file.
- **–out=[OutputFilePath]** – Path for the output file that will be generated.

An example command line would be:

```
DMTDiff.exe –base=c:\DMTDiff\BaseFile.csv –cmp=c:\DMTDiff\CmpFile.csv –out-
t=c:\DMTDiff\NewFile.csv
```

Relative Paths

The required arguments can be provided as actual paths as shown in the example above. They can also be provided using relative paths when the data files exist in the same folder where the utility is

running.

For example:

```
DMTDiff.exe -base=BaseFile.csv -cmp=CmpFile.csv -out=NewFile.csv
```

If the utility is run from the **C:\DMTDiff** directory (the same location as the above data files), then the example above will work.

Note in this example the output file will also be written to this directory.

File Merge

The utility can accept multiple .CSV files as part of a single input. To do this the files need to be renamed to start with a similar pattern. You then need to include a ***** suffix in the parameter as per example below to define this pattern.

For example:

```
DMTDiff.exe -base=c:\DMTDiff\BaseFile*.csv -cmp=c:\DMTDiff\CmpFile.csv -out-
t=c:\DMTDiff\NewFile.csv
```

The above example will load in all .CSV files in the folder that begin with 'BaseFile'. For example, it will load the following files:

- c:\DMTDiff\BaseFile-A.csv
- c:\DMTDiff\BaseFile-B.csv
- c:\DMTDiff\BaseFile-C.csv
- c:\DMTDiff\BaseFile-D.csv

CSV Split Tool

The utility can also be used to split up a single .CSV into separate smaller .CSV files. To do this you need to pass the **-split** argument, an input file path, and an integer argument.

The format of the command line is as follows:

```
DMTDiff.exe -split -input="[InputFilePath]" -files=[FileCount]
```

For example:

```
DMTDiff.exe -split -input="C:\DMTDiff\Input.csv" -files=5
```

This above example will create 5 new .CSV files containing subsets of the data rows from the original. The output files are saved in the same location as the input file and will begin with the same name followed by a counter (note if an output file already exists it will automatically get overwritten).

Using the example above this will output the following 5 files:

- c:\DMTDiff\Input-1.csv
- c:\DMTDiff\Input-2.csv
- c:\DMTDiff\Input-3.csv
- c:\DMTDiff\Input-4.csv
- c:\DMTDiff\Input-5.csv

Optional Switches

The default behavior for the utility is to run unattended. This means it should run through to the end without any additional prompts from the user. This includes any errors that may occur. You can however provide optional switches which allow the utility to be configured to behave differently from its default. Each additional switch can be added as a new parameter in the command line (Note switches are not case sensitive).

- **/?** - (Display Help) - This will display help text for the utility.
- **/O** - (Prompt Overwrite) - Checks if an output file already exists and prompts a user for a response.
- **/T** - (Execution Time) - Display a counter of how long the process has taken from start to finish.
- **/H** - (Hide Progress) - Hide the progress bars when loading the files. This will speed up the process slightly if a visual display is not required.

For example:

```
DMTDiff.exe -base=c:\DMTDiff\BaseFile.csv -cmp=c:\DMTDiff\CmpFile.csv -out-
t=c:\DMTDiff\NewFile.csv /o /t /h
```

Updating the Data Management Tool (DMT)

This article details how you update the **Data Management Tool (DMT)**.

Extensions

Epicor may release extension modules for **DMT** that add functionality and imports. All extensions use the following file convention:

```
Epicor.DMT.Extension.ExtensionName.dll
```

Place these extensions in the same file location as the **DMT.exe**. These extensions will automatically appear next time you run **DMT**.

Patches and Versions

The **DMT** version will match the current Kinetic's version. If the current version is 2026.100, the Data Management Tools will also be the 2026.100 version.

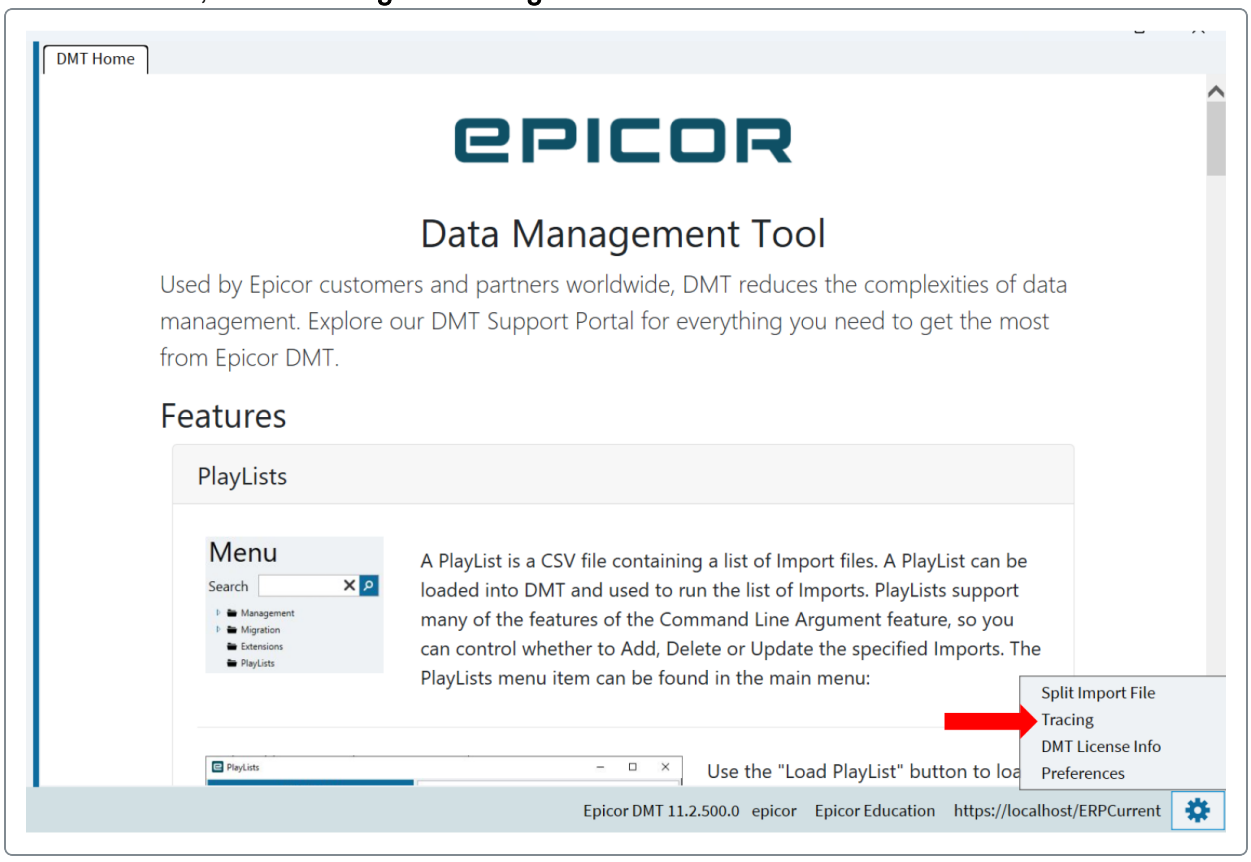
Troubleshooting the Data Management Tool (DMT)

You many encounter issues while you run processes with the Data Management Tool (DMT). This help article documents some ways you can correct these issues.

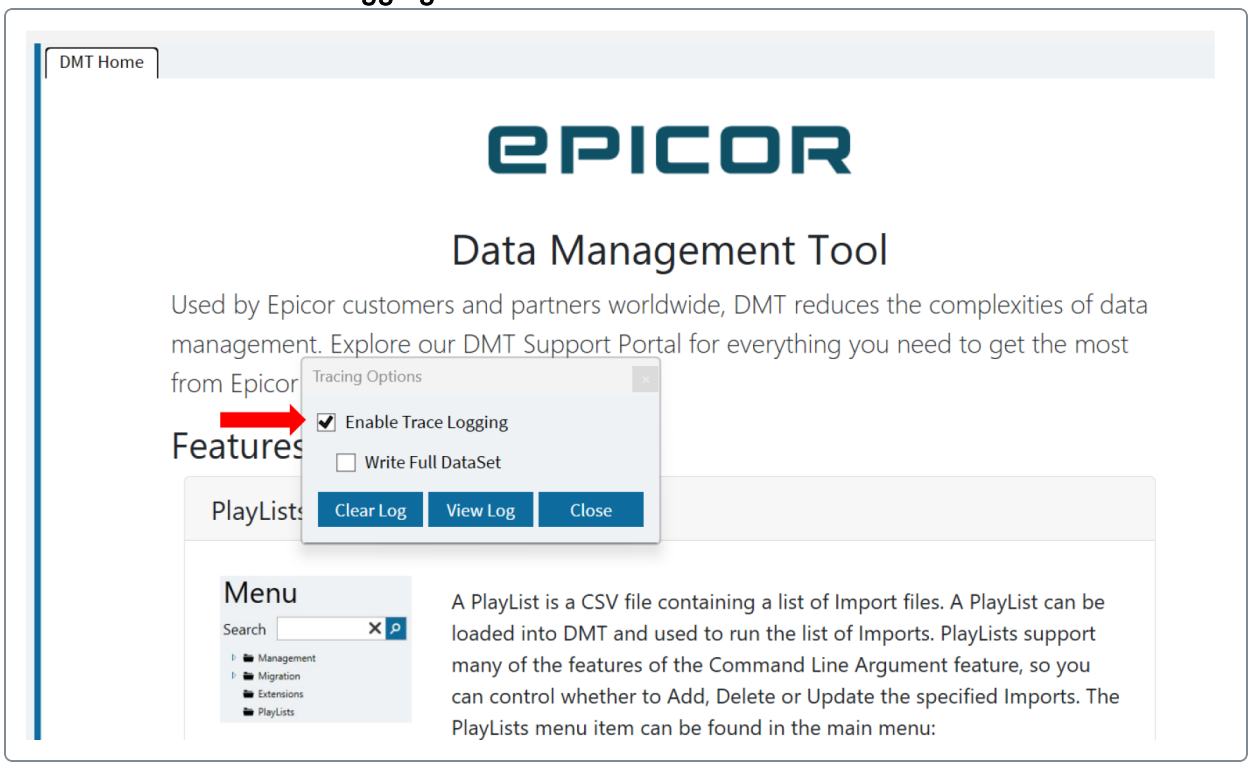
Enabling the Trace Log

Do the following to enable the trace log. If the DMT process is having an issue, this log can help to identify the cause of the problem.

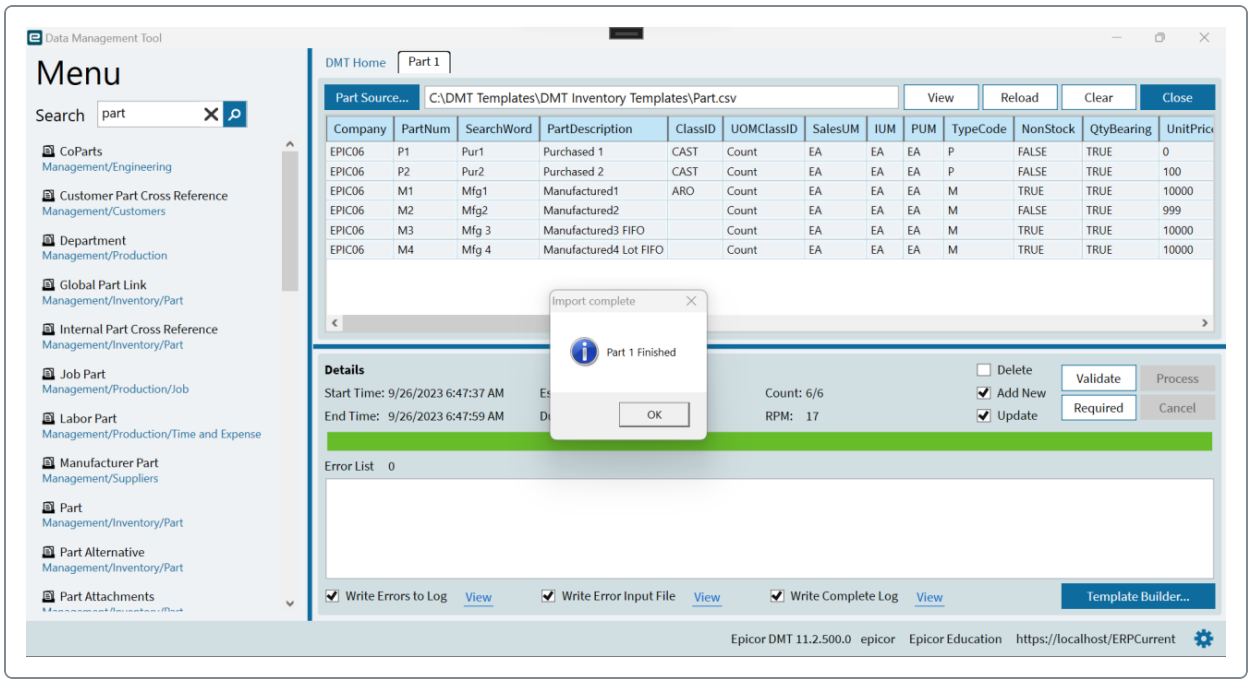
1. From the menu, select **Settings > Tracing**.



2. Select the **Enable Trace Logging** checkbox.

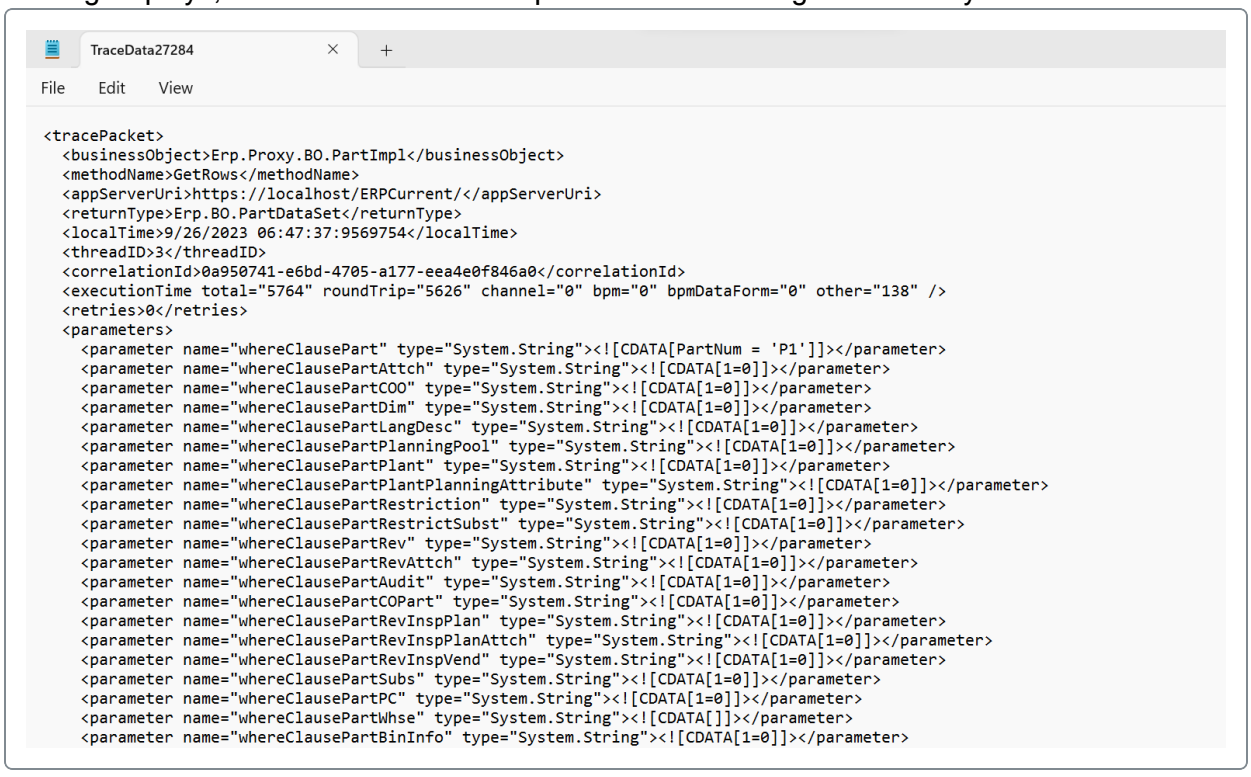


3. **Process** the data.



4. Click **View Log**.

5. The log displays; review it to see how the process ran and if it generated any errors.



6. **Save** the log if you need to review it later.

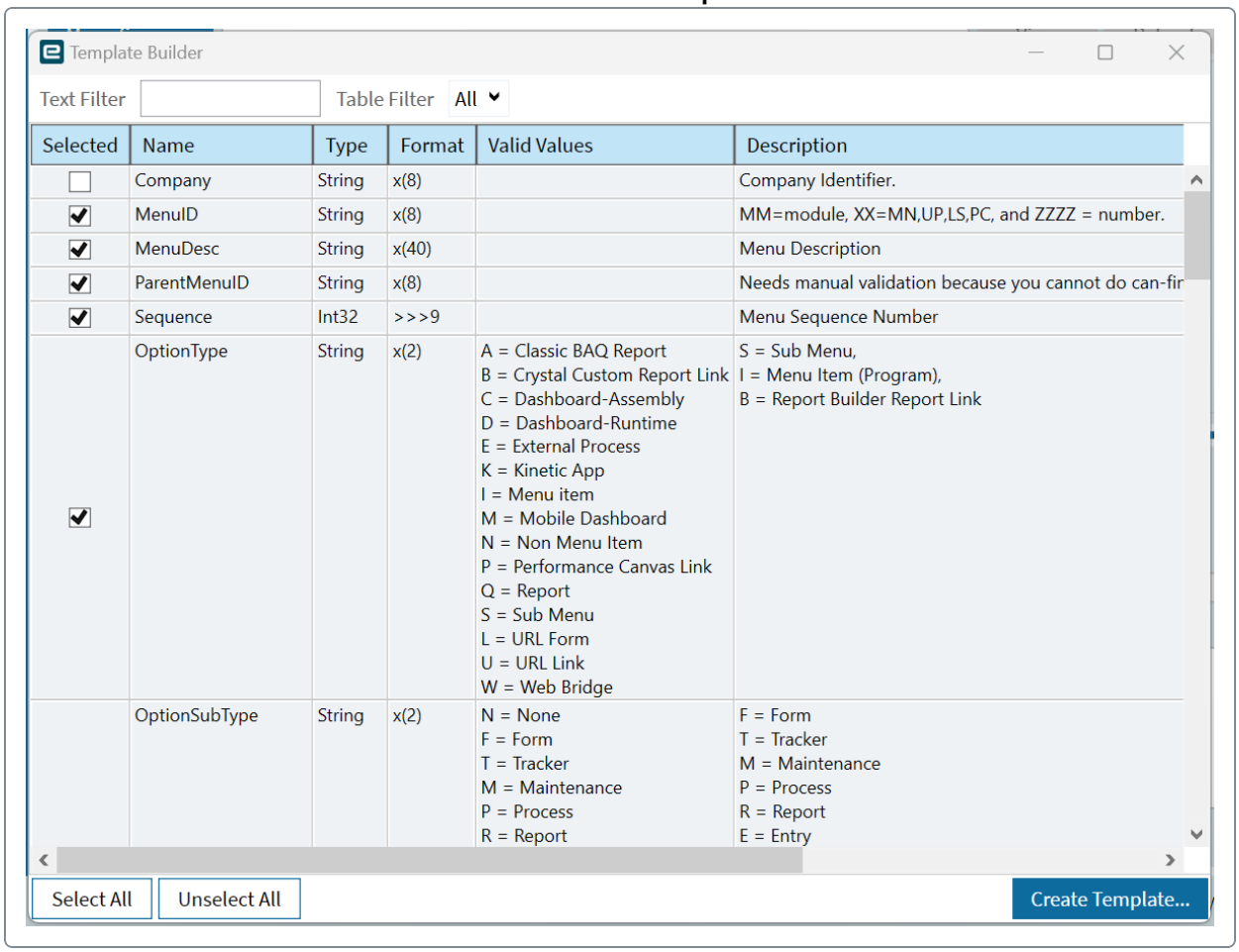
Setting Up the Template Builder

Before you build a template, make sure that this template matches the Kinetic database.

Valid Values

Check the values assigned to columns.

1. Review if the column name has valid values in the **Template Builder**.



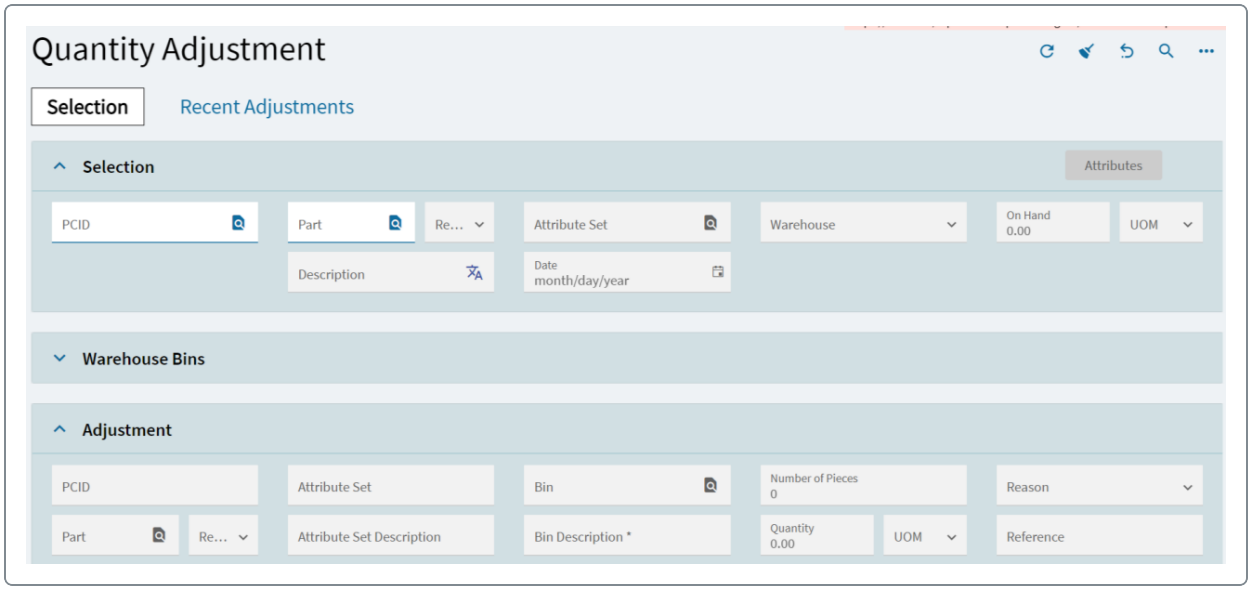
2. If it has valid values, you should use one option of the provided list.
3. Make sure that the value in the template for that column is written exactly as it is shown in the valid values. Do not write “a” instead of “A” for Option Type.

Column Order

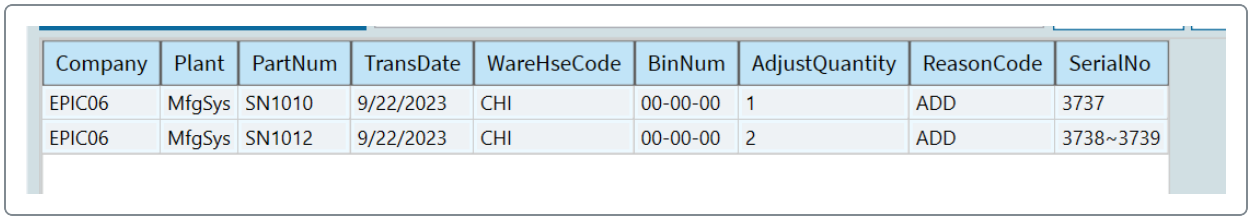
Verify that the database columns are in the same sequence as they are in Kinetic.

For example in Kinetic, when enter a Quantity Adjustment for a selected Part through Kinetic, you first select the **Part**, then you can add any information you want in the **Selection** section. After filling

that data, you move to the **Adjustments** section where you must specify the **Bin**, **Number of pieces**, **Quantity**, **Reason**, and other values.



After you identify what order you enter the data in Kinetic, make sure the template follows this required sequence:

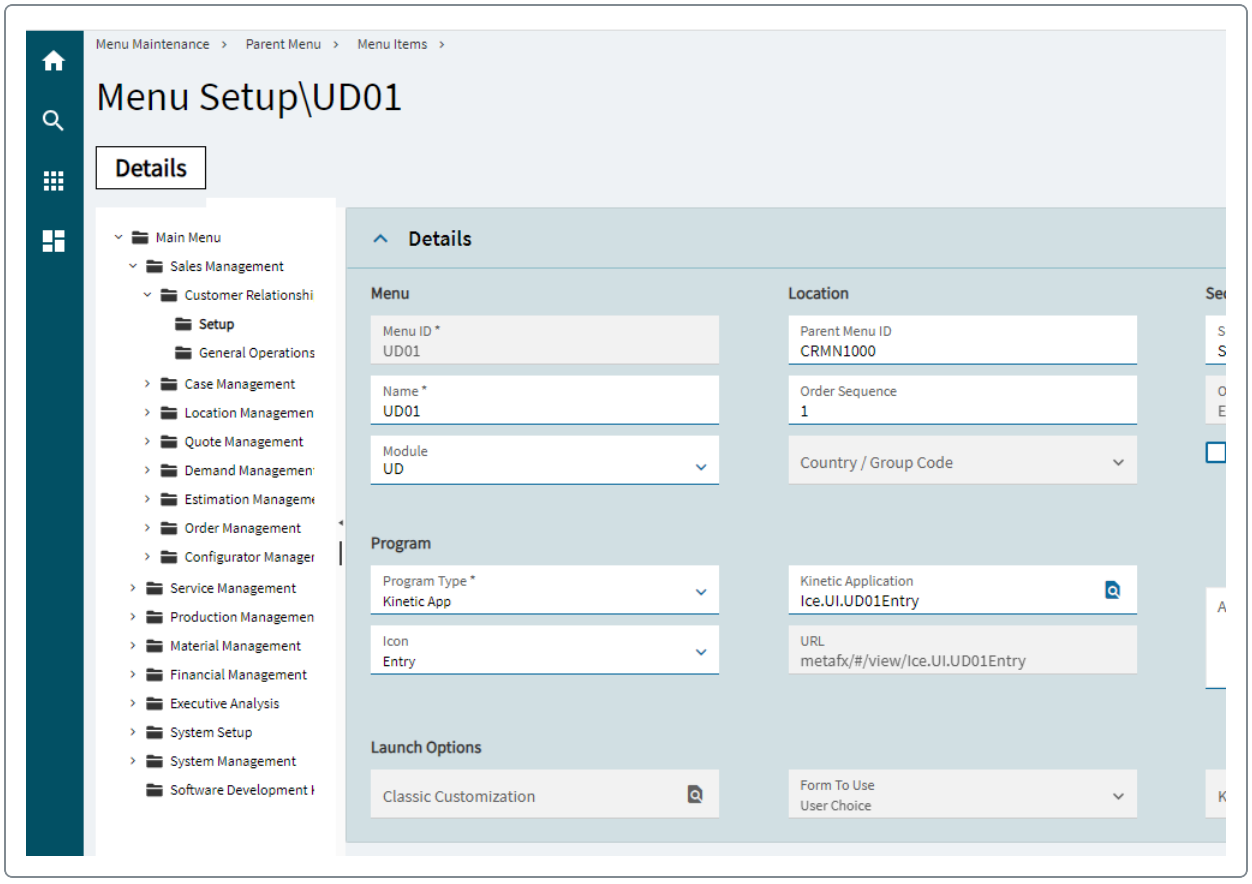


Company	Plant	PartNum	TransDate	WareHseCode	BinNum	AdjustQuantity	ReasonCode	SerialNo
EPIC06	MfgSys	SN1010	9/22/2023	CHI	00-00-00	1	ADD	3737
EPIC06	MfgSys	SN1012	9/22/2023	CHI	00-00-00	2	ADD	3738~3739

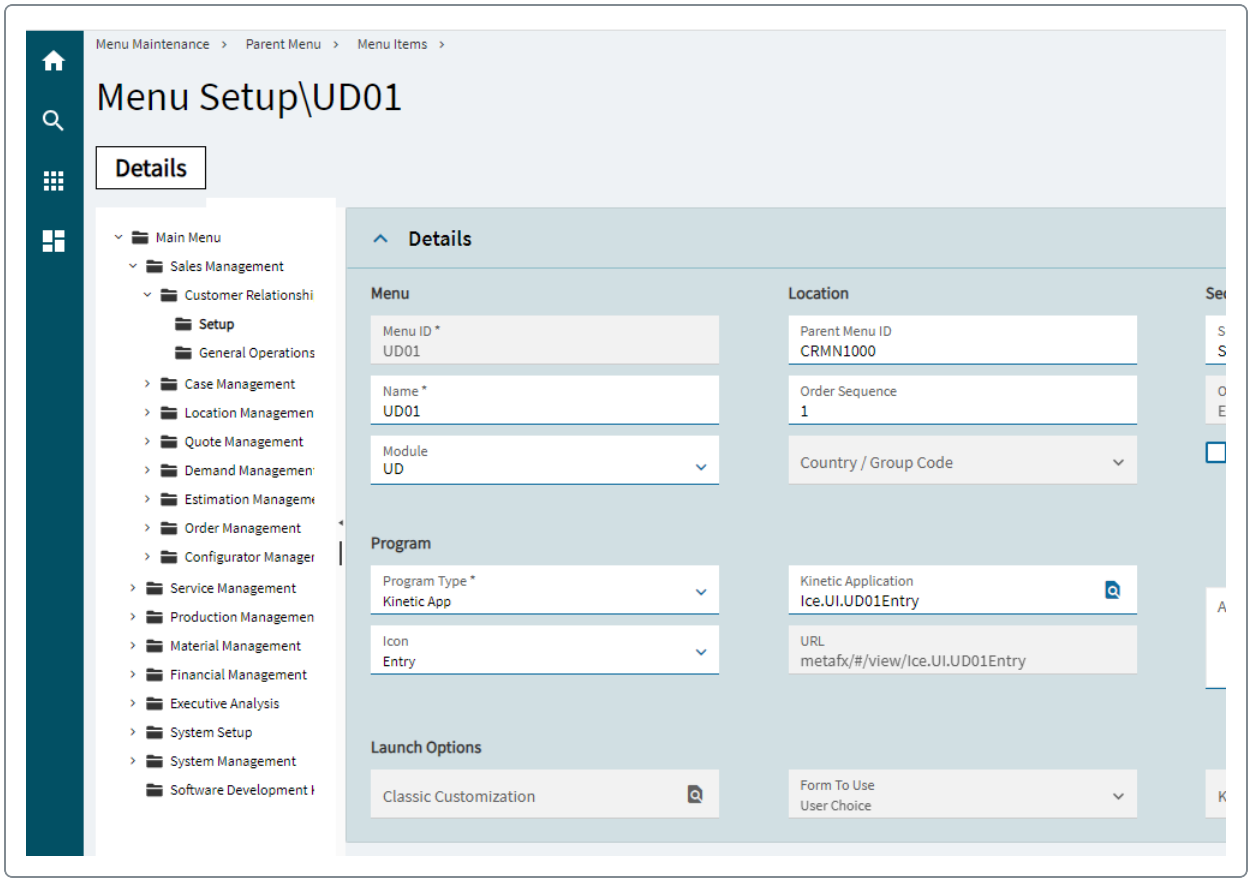
Accessing UD Tables

When you need to access User-Defined (UD) tables in the DMT, you must assign these tables to a Kinetic menu and users must be able to access these tables. This prevent users who DO NOT have access to the UD tables from entering invalid data.

For example, to be able to see the **UD01** menus in the DMT, there must be a menu in Kinetic that is assigned to **UD01Entry**.

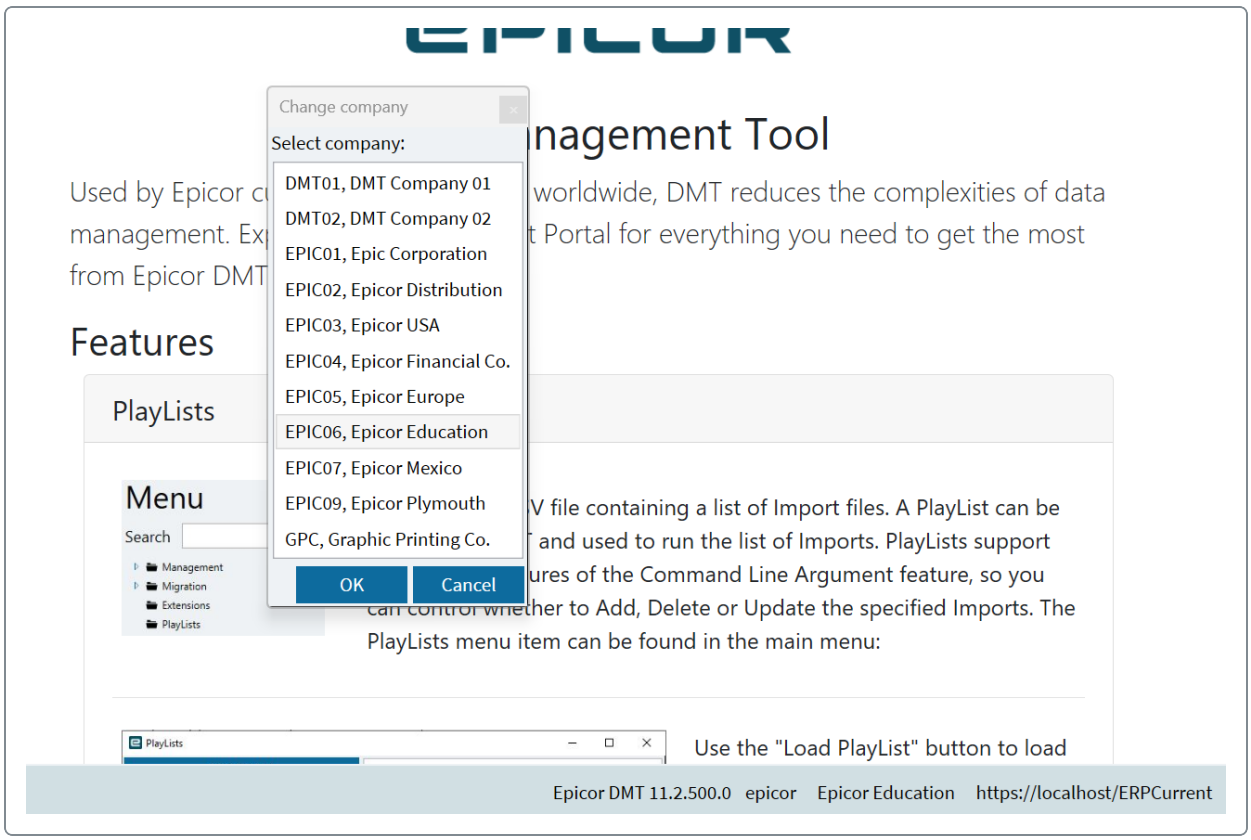


When the UD table is assigned to a Kinetic menu, the DMT will show all the menus linked to this UD table.



Changing the Company

If you see that a DMT menu is not taking the **Company** column into consideration, a possible solution can be to change the **Company** in the DMT with the company you are using in your template. Do this by double-clicking the **Company** shown in the DMT and then selecting the company.



Be aware that this is a workaround. This should not be necessary, as each DMT menu should let you add records for multiple companies. If this happens in one of the DMT menus, please report this issue to Kinetic Technical Support.

Running the Split Tool (DMTDiff Utility)

Processing a huge quantity of records using only a file can cause performance to slow down or even stop. To avoid this issue, you can either manually split the records into multiple files or use the **Split Tool** (DMTDiff Utility).

You can read more about this tool by reviewing the **Running the DMTDiff Utility in the Data Management Tool (DMT)** help article.

Using the DMT 64 bit Version

When you process a lot of records, it is recommended you use the **DMT64.exe** version. This will help you avoid getting the System Out Of Memory exception message. You can find this version under the deployment client folder as DMT64.exe. This version is available from 11.2.300 release and later.