ThingWorx Manufacturing and Service Apps Customization Guide

8.2

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

Contents

About This Guide 5

Customizing the ThingWorx Manufacturing and Service Apps 6
- Using Duplicate Files for Customization 7
- Comparing Your Customizations with the Default 8
- Viewing User Interface Components In Use 9
- Using Tags and Descriptions to Find Entities 10
- Upgrade and Customizations 13

Using the Launch Point Configuration Thing to Link to Customized Mashups 15
- Example Customization Using the Launch Point Configuration Thing 16
- Changing the Tiles in the Main Application Console 18
- Changing the Logo and Text on the Welcome Page 25
- Changing or Adding New Logos in the Application Headers 26
- Other Customizable Launch Point Mashups 27

Changing Labels in the Application 32

Changing the Logos in the Application Console 34
- To change the PTC logos in the application console: 35
- To keep the PTC logos and add an additional logo in the master console: 35

Customizing the Schedulers 36
- To configure the launch schedule for a scheduler: 37
- To enable or disable the scheduler: 37
- To set the age of the data to keep after each purge: 38

Customized KPI Evaluation 39

Creating Custom Roles 41

Adding Custom Subtypes 43
- Creating a New Thing Template to Use as a Subtype 44
- Updating an Existing Thing Template to Use as a Subtype 45
- Define the List of Types Displayed When Creating New Equipment 48
- Defining a Display Alias for your SubType 48
- Defining Launch Points for Custom Subtypes 50

Customizing the Tag Picker Common Component 52
- Disable Preservation of the Last Selection 53
- Browse Data from Custom Connectors 53
- Using the Tag Picker Common Component in a Mashup 55

Customizing Equipment Structure Relationship Rules 57

Customizing Tab Pages in the Configuration and Setup Tile 61
- Changing the Main Mashup 62

Adding Tab Pages in the Configuration and Setup Main Mashup 62
Granting Access Control to the Tab Page..... 64
Modify the Existing Tab Pages 65
Adding a Custom Notification Handler..... 69
Implementing the NI TestStand Connector 73
Prerequisites..... 74
Download the NI TestStand Connector 74
Set the Environment Variable..... 74
ThingWorx Composer Configurations..... 75
Setup the Edge Micro Server (EMS) Environment 76
Install the NI TestStand Plugin 76
Adding Steps in a Test Stand Sequence 79
Customizing Asset Advisor..... 87
Customizable Asset Advisor Mashups..... 88
Anomaly Detection and Asset Advisor 92
Example: Displaying Anomaly Count in Asset Summary Mashup 98
Deprecated Entities, Services, and Properties 105

About This Guide

This guide describes how to customize ThingWorx Manufacturing Apps and ThingWorx Service Apps in ThingWorx Composer.

This guide assumes that prerequisite products are installed and configured, including a KEPServerEX with connected devices. For more information, see the Product Requirements section in the *ThingWorx Manufacturing Apps Setup and Configuration Guide* or the *ThingWorx Service Apps Setup and Configuration Guide* located at the PTC Reference Documents website.

Related Documentation

It may be useful to refer to the following documents located at the PTC Reference Documents website under the product categories: ThingWorx Manufacturing Apps Family and ThingWorx Service Apps Family.

- *ThingWorx Manufacturing Apps Setup and Configuration Guide*
- *ThingWorx Service Apps Setup and Configuration Guide*
- *What's New in ThingWorx Manufacturing Apps*
- *What's New in ThingWorx Service Apps*

Comments

PTC welcomes your suggestions and comments on its documentation. To submit your feedback, you can send an email to documentation@ptc.com. To help us quickly address your concern, include the name of the PTC product and its release number with your comments. If your comments are about this book, include the *ThingWorx Manufacturing and Service Apps Customization Guide* book title as well.

1

Customizing the ThingWorx Manufacturing and Service Apps

Using Duplicate Files for Customization 7
Comparing Your Customizations with the Default 8
Viewing User Interface Components In Use 9
Using Tags and Descriptions to Find Entities 10
Upgrade and Customizations 13

As starter apps, the ThingWorx Manufacturing and Service Apps can be customized in multiple ways to tailor the applications for your specific requirements. These customizations are made using the ThingWorx Composer. Common customizations include:

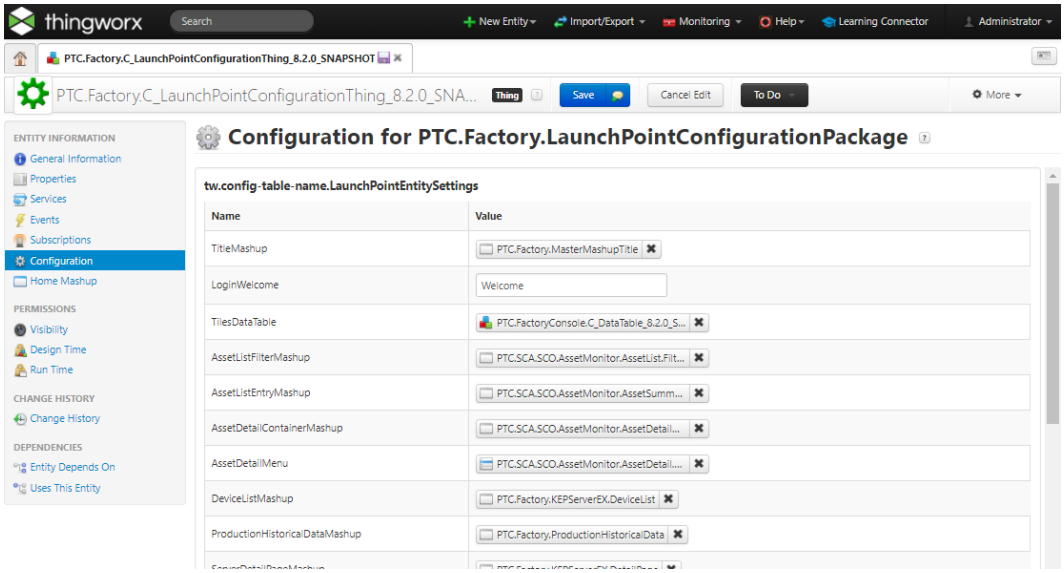
- Customizing the behavior of the apps by editing certain entities delivered within the extension, for example, using custom KPI evaluation formulas, or setting the launch time of certain schedulers.
- Defining custom user roles and custom equipment sub-types.
- Customizing the welcome sign-in screen.
- Adding or changing the logos displayed in the window headers and footers.
- Customizing the tiles within the main console, or adding new tiles to the console.
- Customizing various mashups within the apps to tailor the information displayed.

The ThingWorx Composer is accessed at the following URL: `http://<hostname>:<port>/Thingworx/Composer/index.html`.

Using Duplicate Files for Customization

Certain entities within the ThingWorx Manufacturing and Service Apps are not editable. PTC provides duplicates of these entities for use in customizations, such as the launch point configuration thing, mashups, and the data table. If you have a standard license, and do not have access rights to create or duplicate entities, you can edit these duplicates to customize the apps.

For each release, PTC provides a configuration thing named `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`, for example `PTC.Factory.C_LaunchPointConfigurationThing_8.2.0`. This gives you the ability to link customized mashups to certain launch points, rather than using the default mashups delivered with the application.

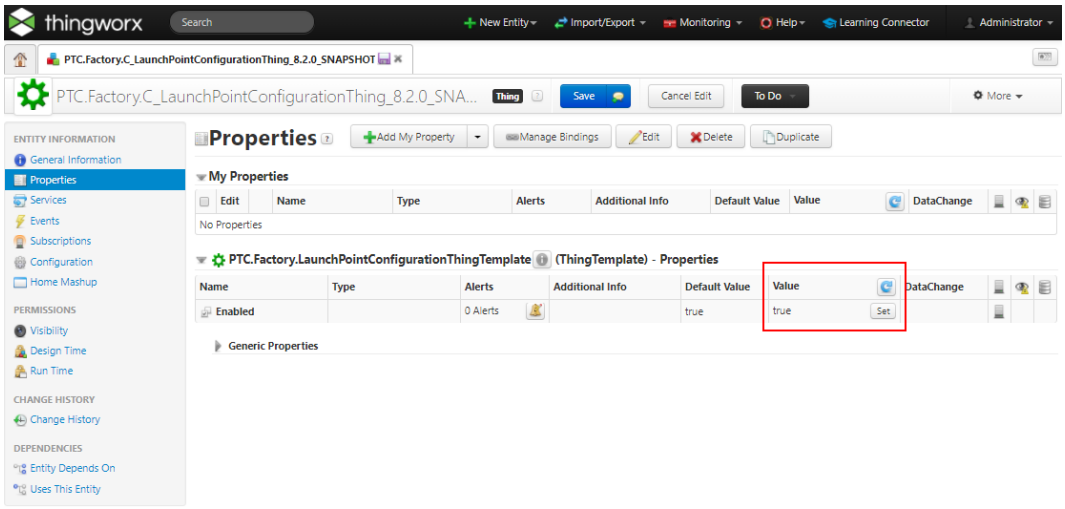


To link to a customized mashup from a launch point in the `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` configuration thing, customize the corresponding duplicate mashup for that launch point in the configuration table, and then change the configuration table to link to the new customized version.

Comparing Your Customizations with the Default

If you want to see the default application after changing the launch point settings in `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`, go to **Properties** for `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` in ThingWorx Composer and change the value of the **Enabled** property to false. Change the property value back to true to return to your customized version. This is the easiest way for you to switch between the customized version and the default version.

You can also open `PTC.Factory.LaunchPointConfigurationThing` to see the default configuration settings there. However, be sure to use only `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` for customization.

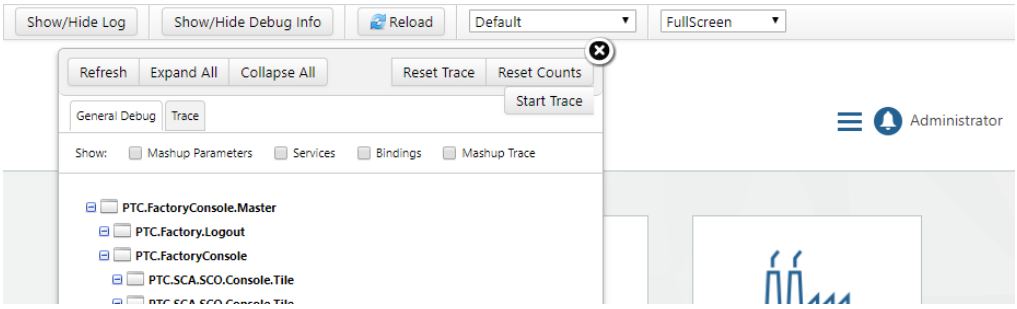


Viewing User Interface Components In Use

The ThingWorx debug toolbar is helpful for viewing and identifying the components that are in use in a particular section of the user interface. This can help you to understand which components you want to reuse or customize.

To turn on the ThingWorx debug toolbar:

1. While viewing the apps in a browser, click CTRL+ALT+F9 to turn on the ThingWorx debug toolbar.
2. Click **Show/Hide Debug Info**. The list of components used in the current page displays.



Using Tags and Descriptions to Find Entities

You may not always know the exact name of an entity which you are trying to find in ThingWorx Composer. The entities delivered with the ThingWorx Manufacturing and Service Apps use tags and descriptions to help you more easily find each entity in ThingWorx Composer. Filter by tag to narrow down the list of available entities, then review the descriptions to find a particular entity.

Descriptions

An entity's **Name** property is used as the identifier for the entity, and must be unique and immutable. The **Description** allows more flexibility in providing readable and meaningful information to help identify the entity and convey its function.

Tags

Tags identify the entity type and where the entity is used. An entity can have multiple tags, as needed. View an entity to see all tags applied to that entity.

The following list describes the types of tags used for ThingWorx Manufacturing and Service Apps entities.

- Application tags identify the application to which an entity belongs. Each entity has a single application tag.
    - PTC:sca-common—entities shared among smart connected applications.
    - PTC:sca-mfg—entities used in the manufacturing applications.
- Entity type tags match the entity's type, allowing you to easily filter for particular types of entities. Each entity has a single type tag.
    - PTC:DataShape
    - PTC:ExtensionPackage
    - PTC:Group
    - PTC:Mashup
    - PTC:MediaEntity—Media entities can have one or more additional sub-type tags:
        - PTC:Icon
        - PTC:Image
    - PTC:Menu
    - PTC:ModelTagVocabulary
    - PTC:Network
    - PTC:Organization
    - PTC:Project

    - PTC:Resource
    - PTC:StateDefinition
    - PTC:StyleDefinition
    - PTC:Thing
    - PTC:ThingPackage
    - PTC:ThingShape
    - PTC:ThingTemplate
    - PTC:Widget
- Logical grouping tags identify an entity's association with other entities, for example, entities which are used together in a particular functional area. An entity can have multiple logical grouping tags. Example logical grouping tags include:
    - PTC:Administration
    - PTC:AlertManagement
    - PTC:Common
    - PTC:Device
    - PTC:KPI
    - PTC:Mashup
    - PTC:MediaEntity
    - PTC:Menu
    - PTC:Trend

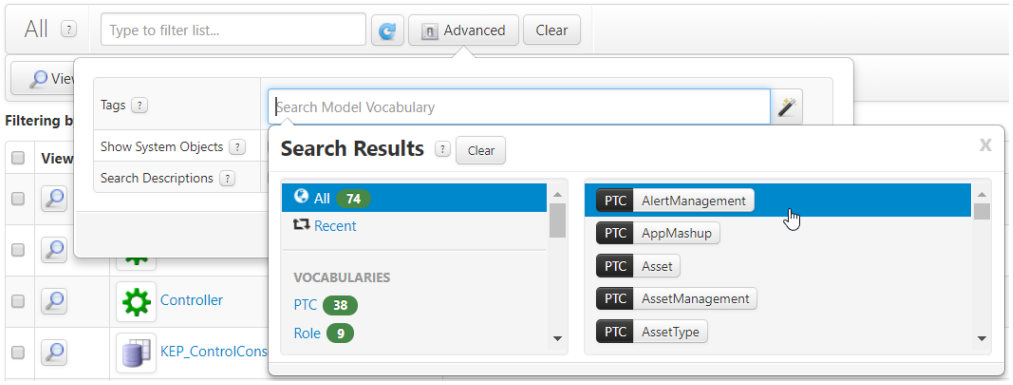
Note

PTC recommends using similar tagging and description conventions with your own custom entities.

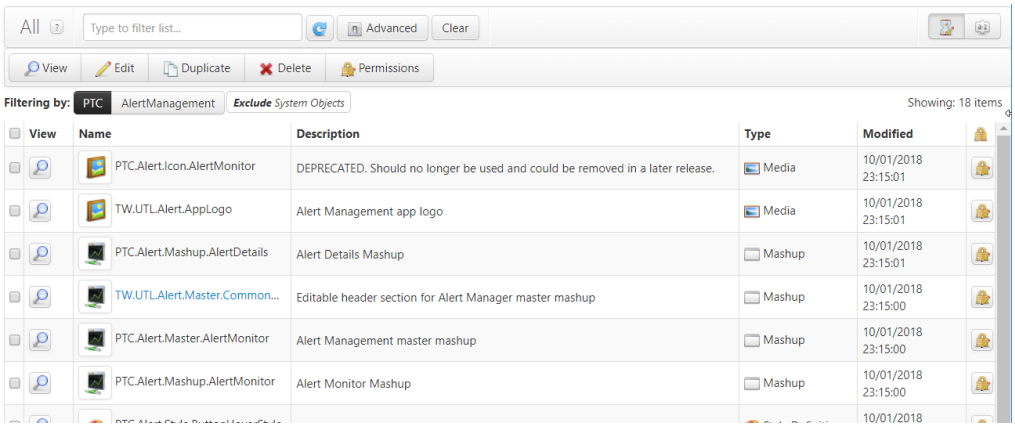
Example

To find an entity that is used in alert management:

1. Click **Advanced** next to the filter field.
2. Click  , and select the PTC:AlertManagement tag from the list of defined tags.



3. Click **Done**. The list filters to show entities tagged as being used in alert management.
4. Use the descriptions to identify the entity you want to view.



Upgrade and Customizations

When upgrading to a new version of ThingWorx Manufacturing and Service Apps, different customizations are impacted differently.

Direct Edits to Extension Entities

Direct edits to entities provided with the extensions, such as the schedulers and media entities, are overwritten during an upgrade. If these customizations are wanted in the new release, they will need to be re-implemented after the upgrade is complete. These customizations are clearly identified when they are discussed.

Localization Table Changes

The localization tables are overwritten when an upgrade is installed. To keep your localization table modifications, export the customized localization table before performing an upgrade, and import it back after the upgrade is complete.

Launch Point Configuration Thing and Duplicate Mashup Changes

Changes made in the release-specific launch point configuration thing (`PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`) and release-specific mashup duplicates are retained, but not automatically applied in the upgraded extension.

For each new release update, a new `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` is delivered with the application. After you see the new changes in the application, you can compare your old `PTC.Factory.C_LaunchPointConfigurationThing_[OldReleaseVersion]` with the new `PTC.Factory.C_LaunchPointConfigurationThing_[NewReleaseVersion]`, and modify the launch point settings as needed. Your customized `PTC.Factory.C_LaunchPointConfigurationThing_[OldReleaseVersion]` and any other customized mashups are not overwritten during upgrade.

Tile Changes In the Data Table

Changes made for tiles in the release-specific data table (`PTC.FactoryConsole.C_DataTable_[ReleaseVersion]`) are retained, but not automatically applied in the upgraded extension. If you want those changes in the upgraded extensions, the following steps need to be performed after completing the upgrade:

1. Modify the configuration table of the launch point configuration thing `PTC.Factory.C_LaunchPointConfigurationThing_[NewReleaseVersion]` to point to the new data table of `PTC.FactoryConsole.C_DataTable_[NewReleaseVersion]`.
2. Modify the data table of `PTC.FactoryConsole.C_DataTable_[NewReleaseVersion]` with all changes that were made in the `PTC.FactoryConsole.C_DataTable_[OldReleaseVersion]`.

2

Using the Launch Point Configuration Thing to Link to Customized Mashups

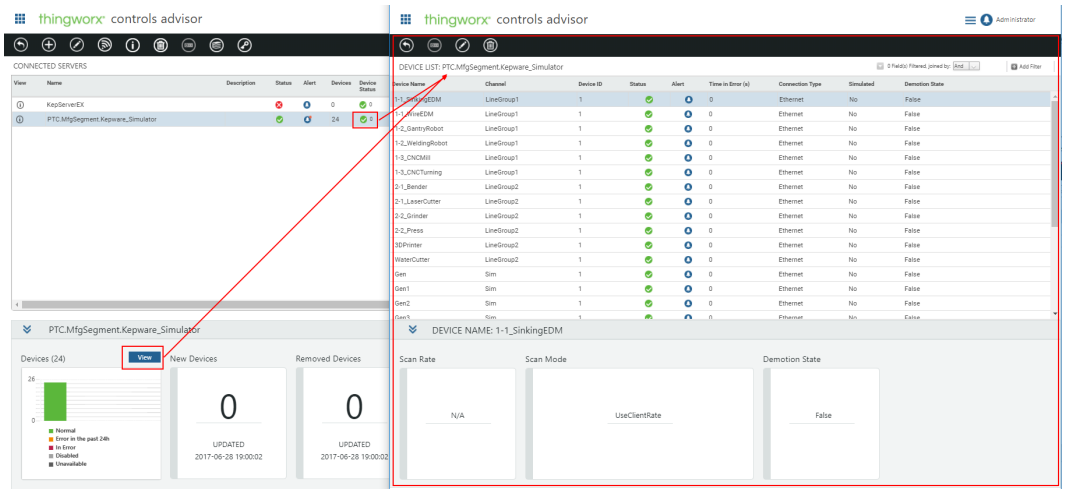
Example Customization Using the Launch Point Configuration Thing 16
Changing the Tiles in the Main Application Console 18
Changing the Logo and Text on the Welcome Page 25
Changing or Adding New Logos in the Application Headers 26
Other Customizable Launch Point Mashups 27

This chapter discusses using the launch point configuration thing to link to customized mashups, including an example, common customizations using the launch point configuration thing, and identifying other customizable launch point mashups.

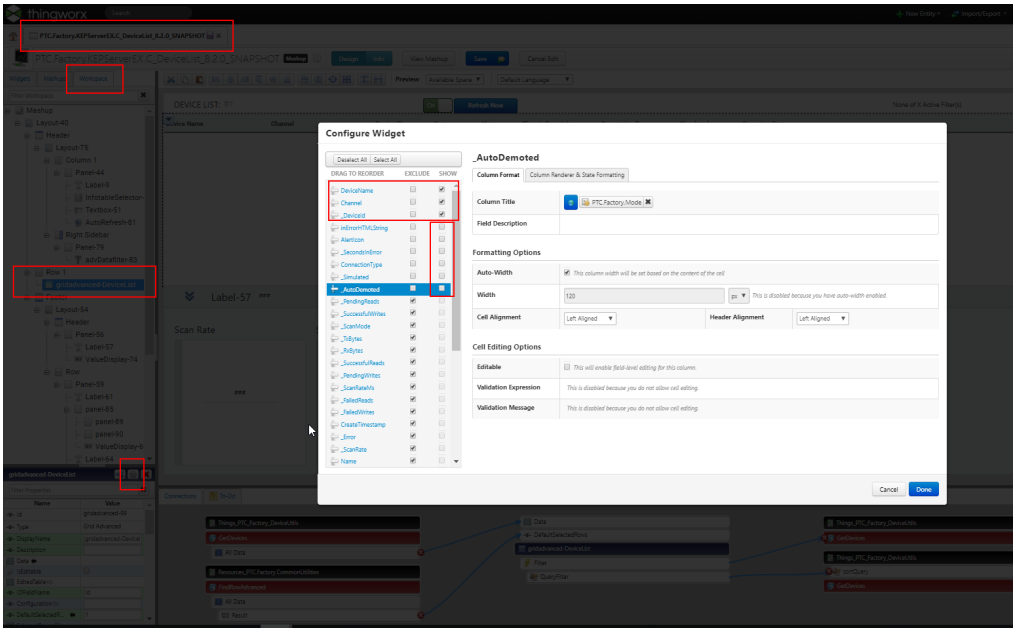
Example Customization Using the Launch Point Configuration Thing

In this example, we assume that you want to create a new customized mashup for the device list page, that launches when you click the **View** button or the **Device Status** link instead of launching the default mashup. This is accomplished by editing `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`.

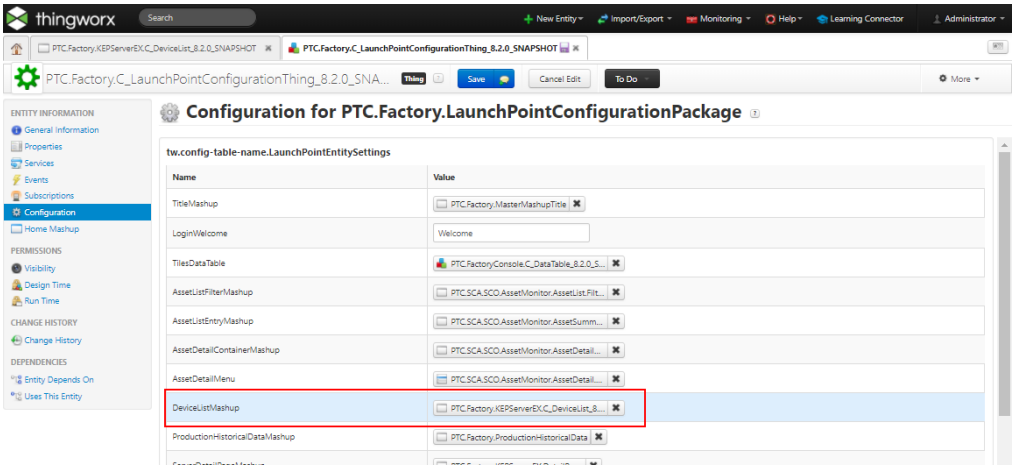
The following figure shows the default device list mashup. The launch points for this mashup are the **View** button and the link under **Device Status** in the Controls Advisor main mashup.



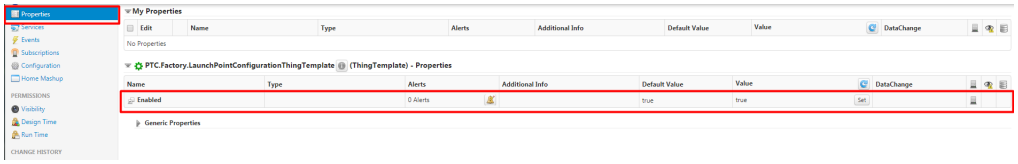
1. Use the out-of-box duplication of `PTC.Factory.KEPServerEX.C_DeviceList_[ReleaseVersion]` or duplicate the `PTC.Factory.KEPServerEX.DeviceList` mashup.
2. Edit the `gridadvanced-DeviceList` widget to keep only the **Device Name**, **Channel**, and **Device ID** columns displayed in the grid.



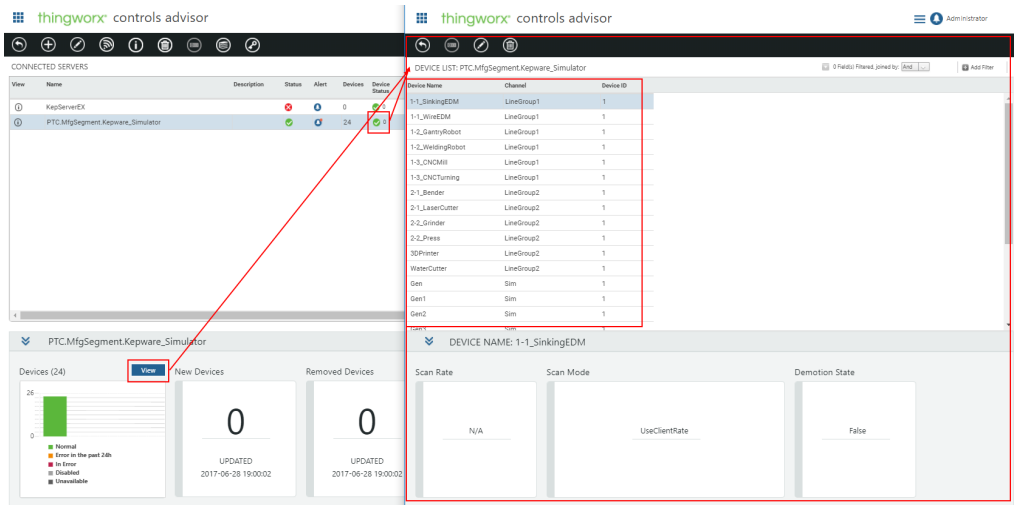
3. Open `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` in ThingWorx Composer. In **Configuration**, change the value for **DeviceListMashup** to the name of the customized mashup edited in the previous step.



4. Go to **Properties**, and ensure that the **Enabled** property is set to true. This is the default value.



5. In the application, verify that the customized mashup can be opened from the same launch points.



Changing the Tiles in the Main Application Console

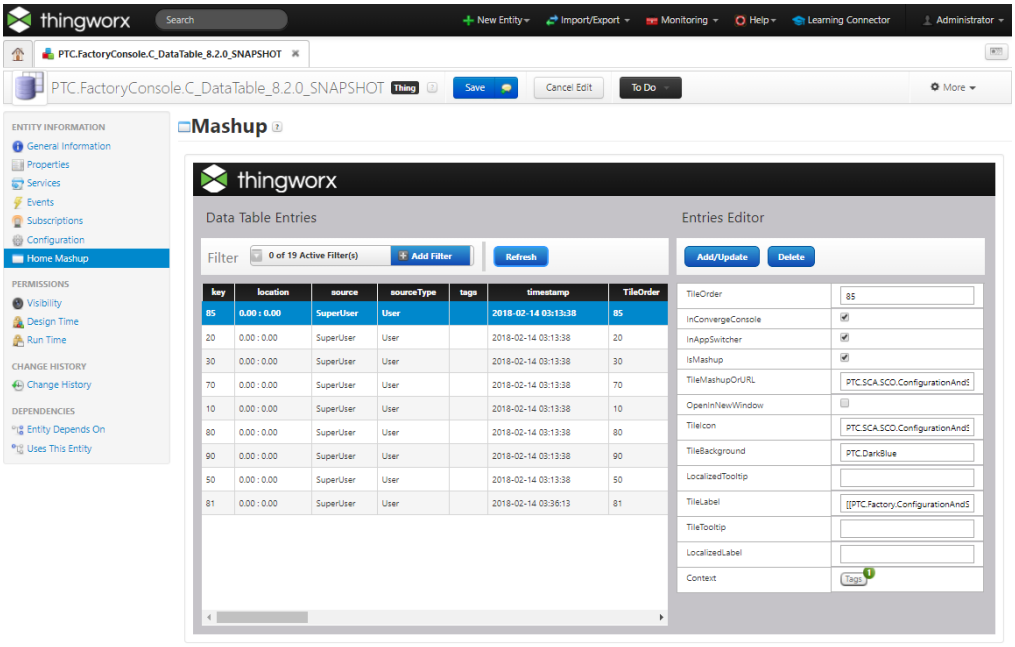
You can add a tile to the console, or update a current tile to point to and launch a new customized mashup. The tiles in the application are controlled by `PTC.FactoryConsole.DataTable`. Use the duplicate `PTC.FactoryConsole.C_DataTable_[ReleaseVersion]` that is provided with the extension for your customizations.

Note

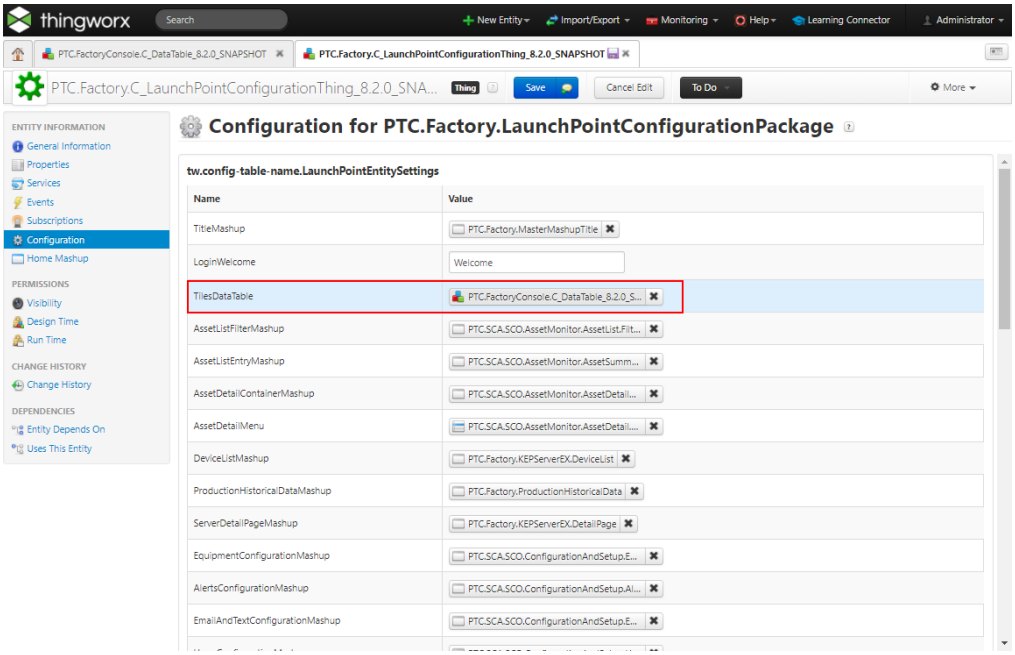
Tile icons are 80 pixels high with varying widths. If you add custom tiles, ensure that the tile icons adhere to the 80 pixel height measurement.

To change the tiles in the main application console:

1. Modify the data table `PTC.C_FactoryConsole.DataTable_[ReleaseVersion]` using the **Home Mashup** of this datatable.



2. Change the **TilesDataTable** launch point value in `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` to `PTC.FactoryConsole.C_DataTable_[ReleaseVersion]`.



The following three duplicate mashups are delivered with the application as they are linked with the main tiles. Use these duplicate mashups for customizing, then change the data table to link certain tile to customized mashups.

- Asset Advisor tile: `PTC.SCA.SCO.AssetMonitor.AssetList.C_AssetListContainerMashup_[ReleaseVersion]`
- Production Advisor tile: `PTC.Factory.C_PlantStatus_[ReleaseVersion]`
- Controls Advisor tile: `PTC.Factory.KEPServerEX.C_ListServers_[ReleaseVersion]`

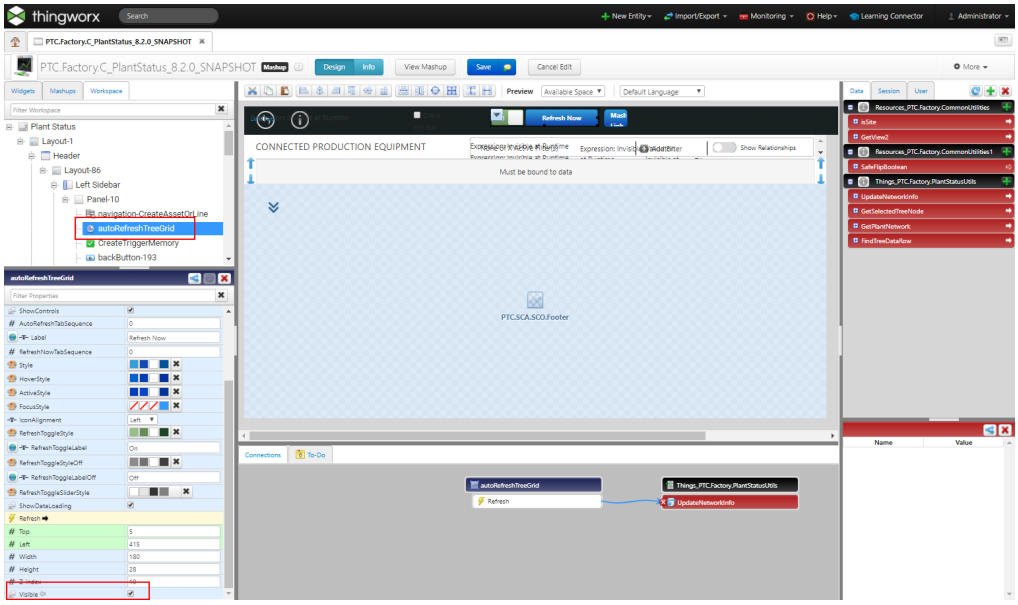
Example

Following is a simple example modifying the `PTC.FactoryConsole.C_DataTable_[ReleaseVersion]` to:

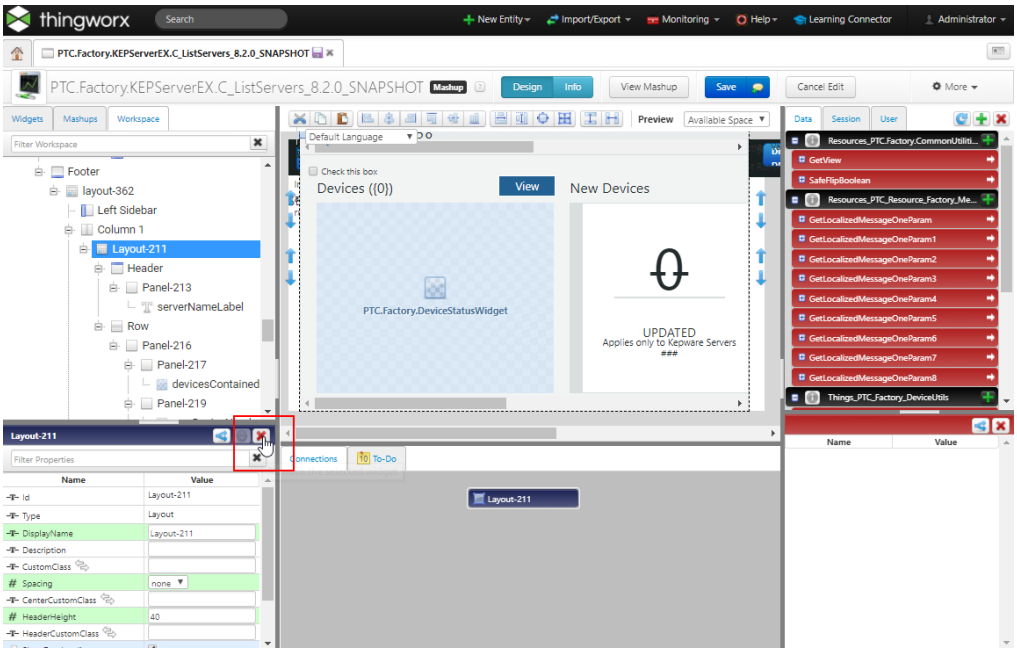
- Add a new tile with the **Production Advisor** label which launches a mashup with the **Refresh Now** button made visible.
- Update the current **Controls Advisor** tile to launch the main mashup without the contained mashups for device information at the bottom of the page.

Procedure:

1. Edit `PTC.Factory.C_PlantStatus_[ReleaseVersion]`, select `autoRefreshTreeGrid`, and select the **Visible** checkbox. Click **Save**.



2. Edit `PTC.Factory.KEPServerEX.C_ListServers_[ReleaseVersion]`, remove the `Layout-211` widget in the `Footer` layout, and click **Save**.

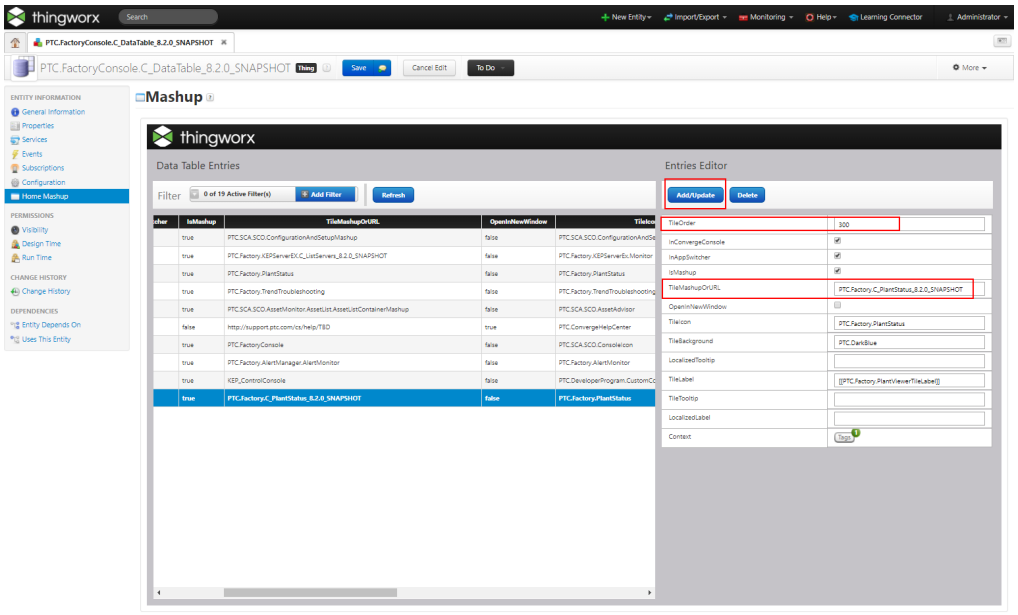


3. Complete the following steps to add the new **Production Advisor** tile.

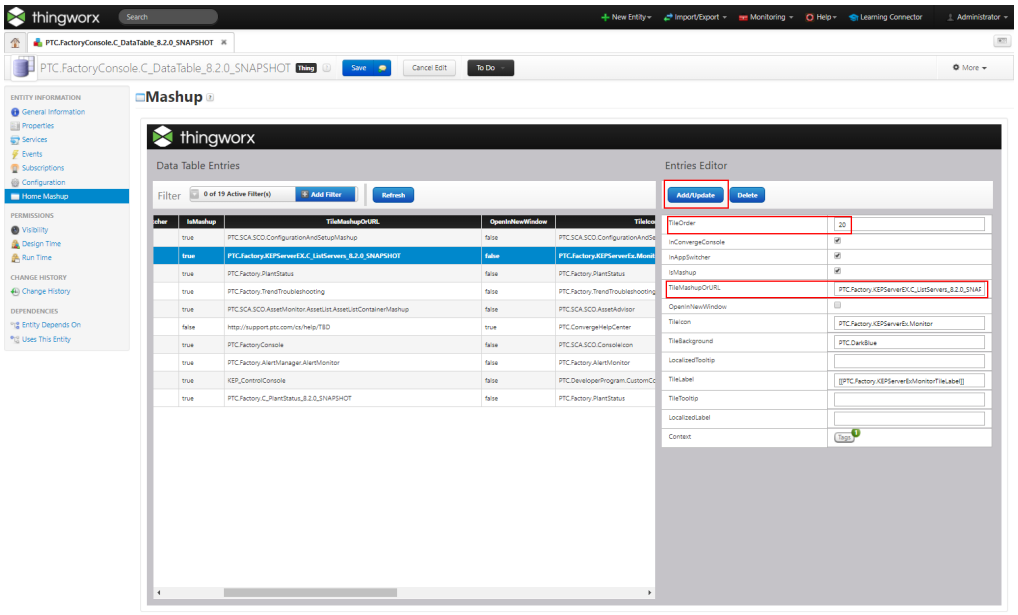
   a. From the `PTC.FactoryConsole.C_DataTable_[ReleaseVersion]`, open **Home Mashup**.

   b. In the **Entries Editor** on the right side, set **TileMashupOrURL** to `PTC.Factory.C_PlantStatus_[ReleaseVersion]`, and choose a unique **TileOrder**.

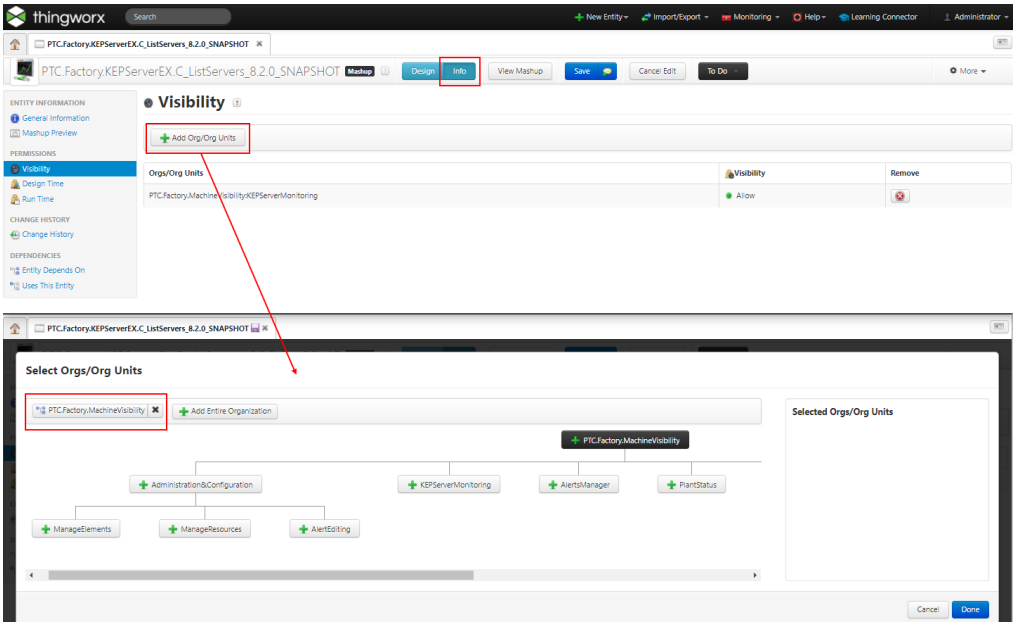
c. Click **Add/Update**.



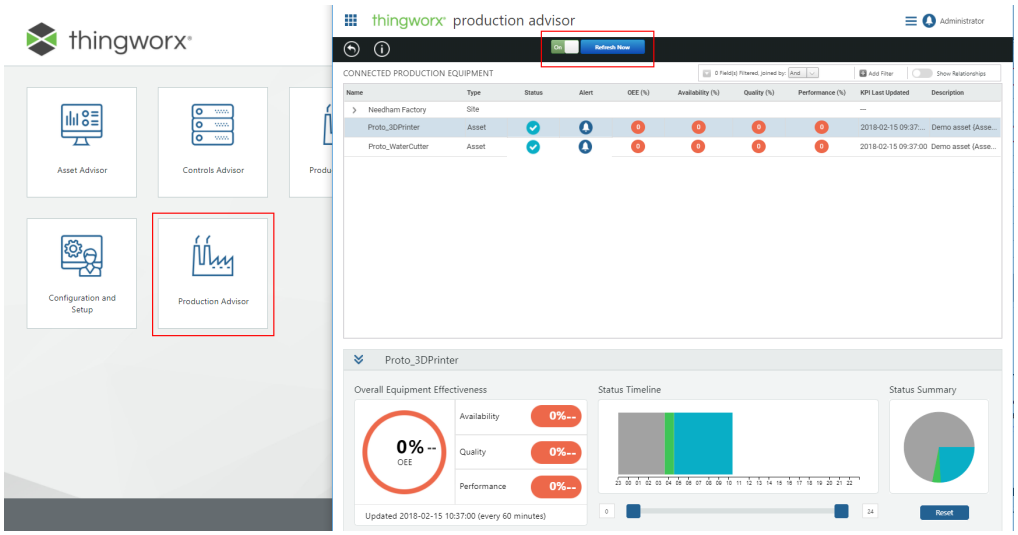
4. From `PTC.FactoryConsole.C_DataTable_[ReleaseVersion]`, open **Home Mashup**. Update the entry whose current **TileMashupOrURL** is `PTC.Factory.KEPServerEX.ListServers` with a **TileOrder** of 20. Change **TileMashupOrURL** to `PTC.Factory.C_KEPServerEX.ListServers_[ReleaseVersion]`, and click **Add/Update**. This updates the existing **Controls Advisor** tile.



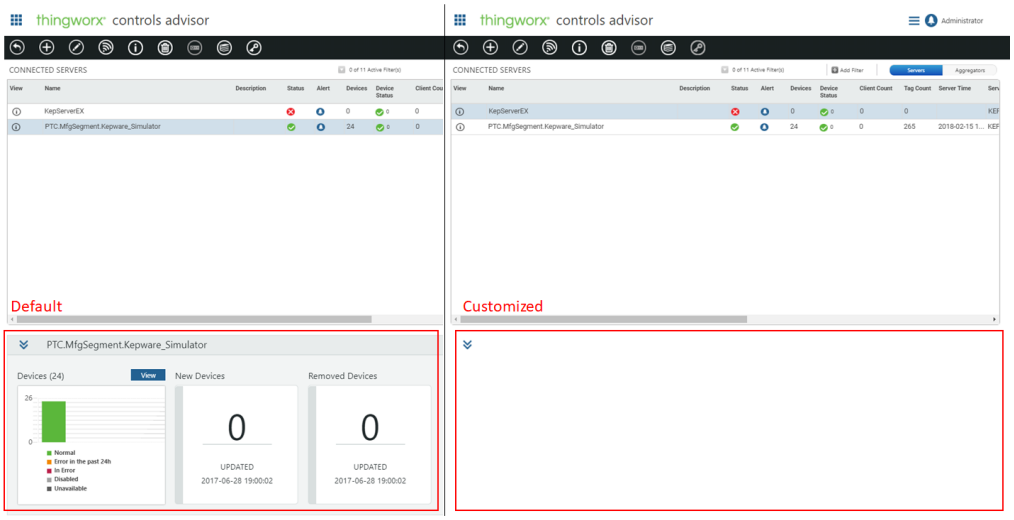
5. Note that the role-based visibility of a tile is the same as its **TileMashupOrURL**. Open `PTC.Factory.KEPServerEX.C_ListServers_[ReleaseVersion]` in ThingWorx Composer, and then choose **Info** ▸ **Permissions** ▸ **Visibility**. The current visibility is visible for `PTC.Factory.MachineVisibility:KEPServerMonitoring`, which is the same as the default mashup `PTC.Factory.KEPServerEX.ListServers`. As a result, this **Controls Advisor** tile in the console is seen by a Controls Engineer in the application console, but not by a Production Engineer. You can click **Add Org/Org Units** to add a new visibility control. Then remove the current visibility control in ThingWorx Composer. In this way, the role-based visibility of the mashup and the tile are customized at the same time.



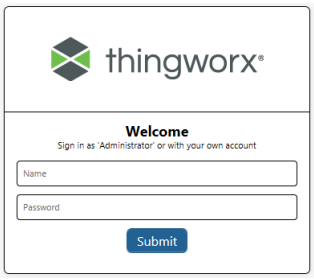
6. Now open the application console. There is a new **Production Advisor** tile that opens the customized mashup with the**Refresh Now** button displayed.



7. From the application console and click **Controls Advisor**. Note that the mashup opened does not have the bottom container mashup for device information.

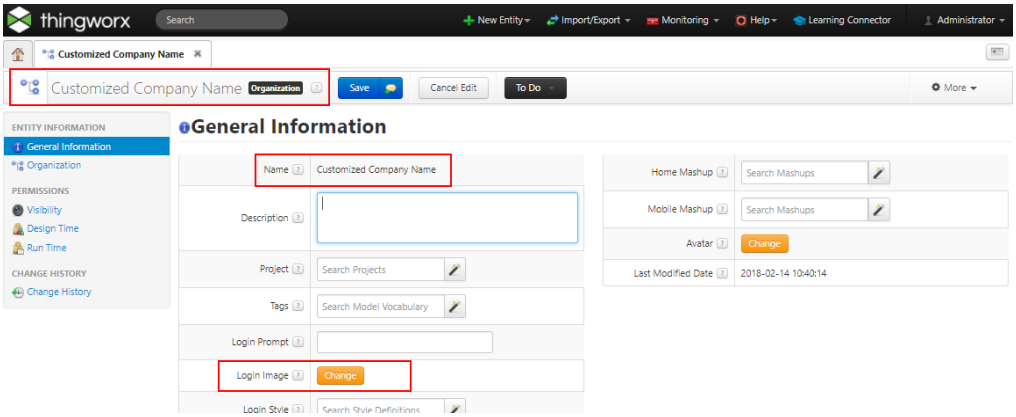


Changing the Logo and Text on the Welcome Page

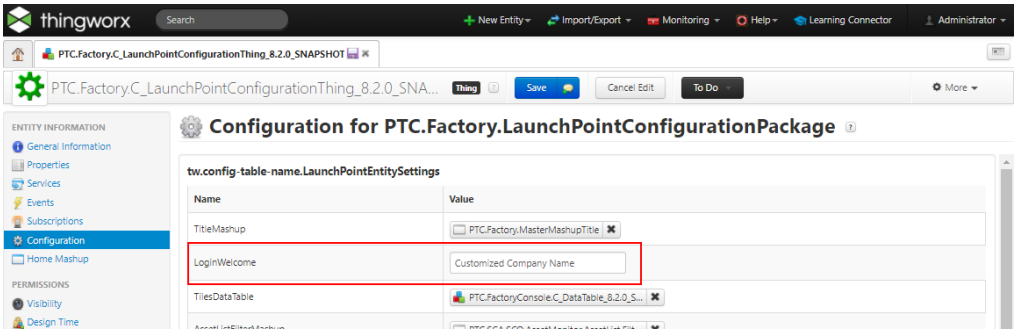


To change the logo and text on the **Welcome** page:

1. Create a new organization in the ThingWorx Composer. The default organization is Welcome and can be used as a template or reference for customization.
2. Choose the name and logo image.



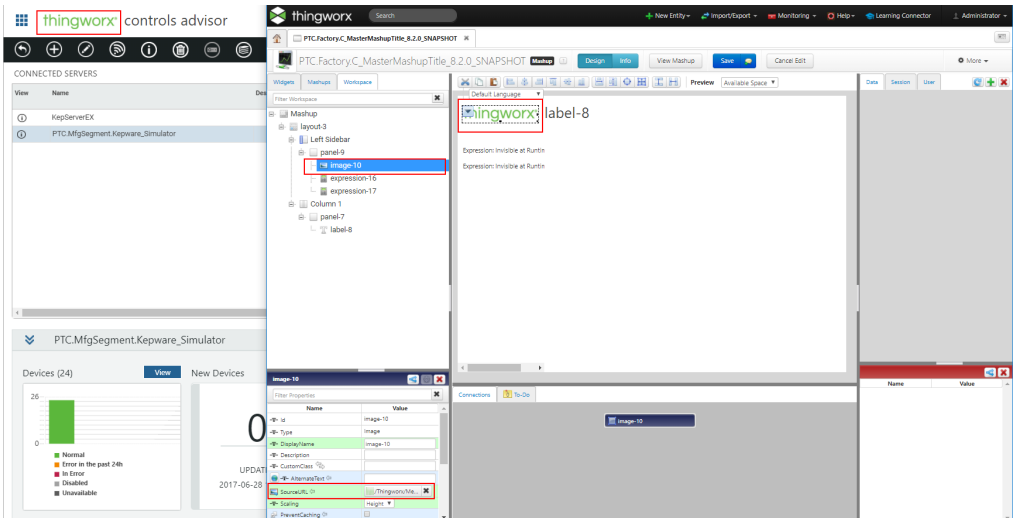
3. Change the **LoginWelcome** launch point value in `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` to the newly created organization entity name.



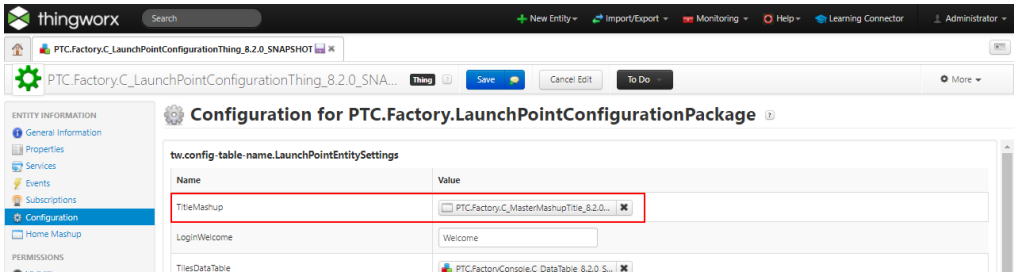
Changing or Adding New Logos in the Application Headers

To change or add logos in the application headers:

1. Edit `PTC.Factory.C_MasterMashupTitle_[ReleaseVersion]` in ThingWorx Composer.



2. Change the **SourceURL** of the image for the header in the mashup to any existing or new media entity.
3. You can do other styling changes on this mashup. For example, you can keep the ThingWorx logo, and add your company logo.
4. Change the **TitleMashup** launch point value in `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` to `PTC.Factory.C_MasterMashupTitle_[ReleaseVersion]`.



Other Customizable Launch Point Mashups

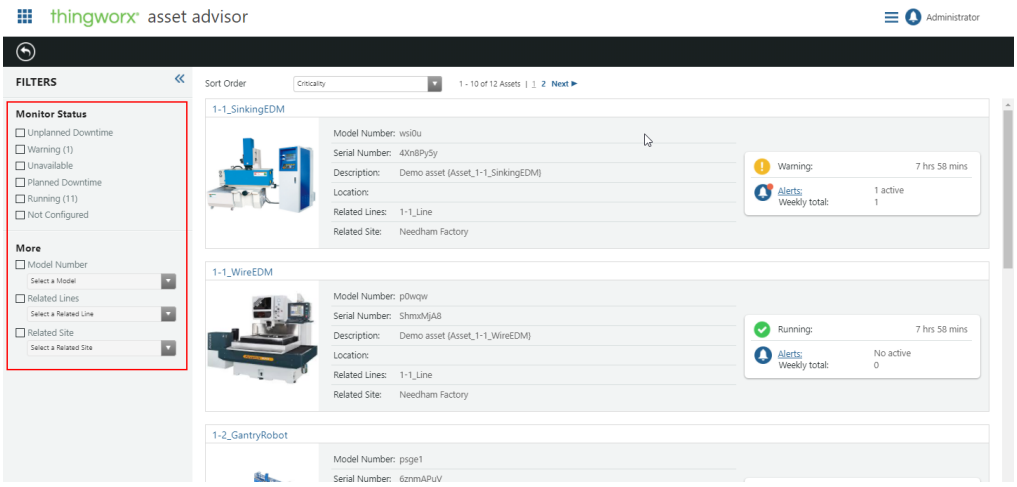
The customization for all launch point mashups follows the same general work flow:

1. Edit the provided duplicate of the mashup, for example `PTC.Factory.KEPServerEX.C_DeviceList_[ReleaseVersion]`. If you have the license to duplicate the original mashup, you can make the duplicate yourself using a different name.
2. Change the mashup name for the corresponding launch point key in `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` to the name of your customized mashup.

The following sections list information for other customizable launch point mashups, including the launch point key, the original default mashup name and image, and the name of the provided duplicate mashup.

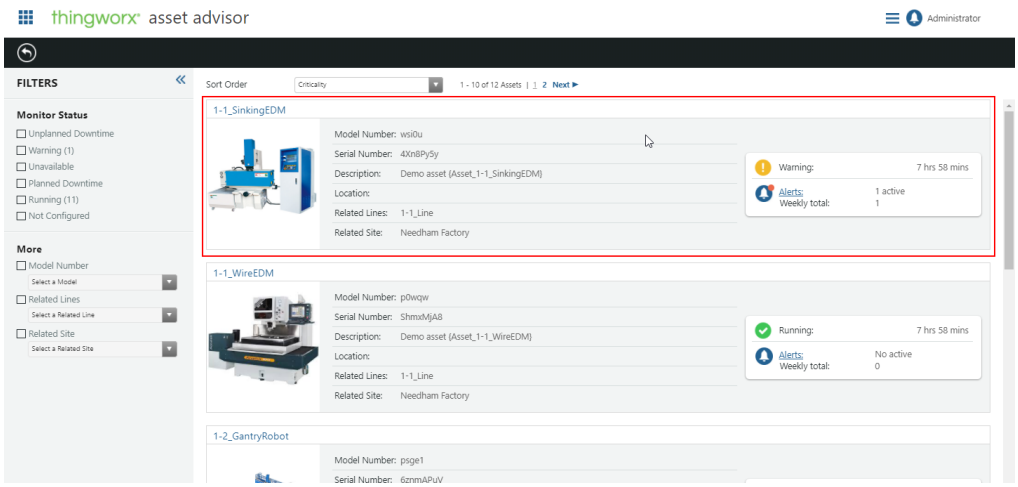
Asset List Filter Mashup

- Launch Point Key: `AssetListFilterMashup`
- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetList.FilterMashup`
- Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetList.C_FilterMashup_[ReleaseVersion]`
- Image:



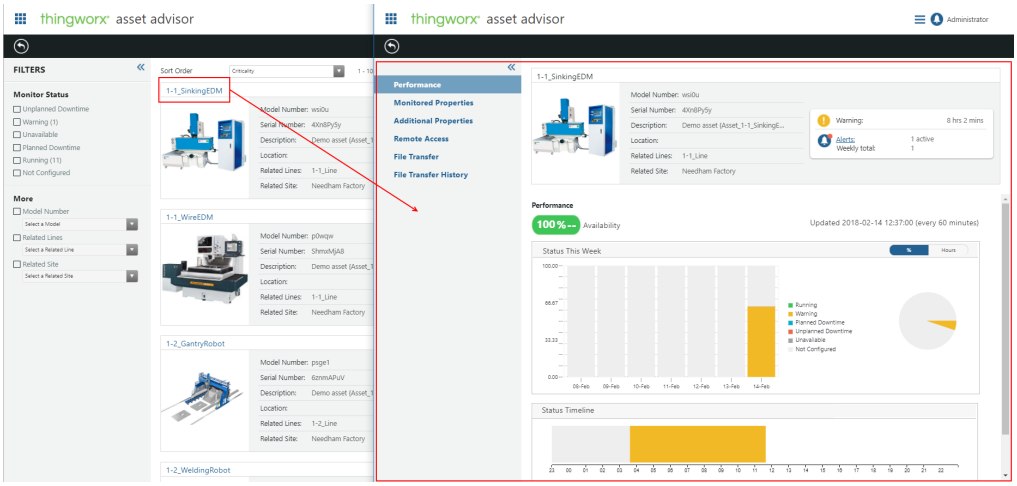
Asset List Entry Mashup

- Launch Point Key: `AssetListEntryMashup`
- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetSummaryMashup`
- Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.C_AssetSummaryMashup_[ReleaseVersion]`
- Image:



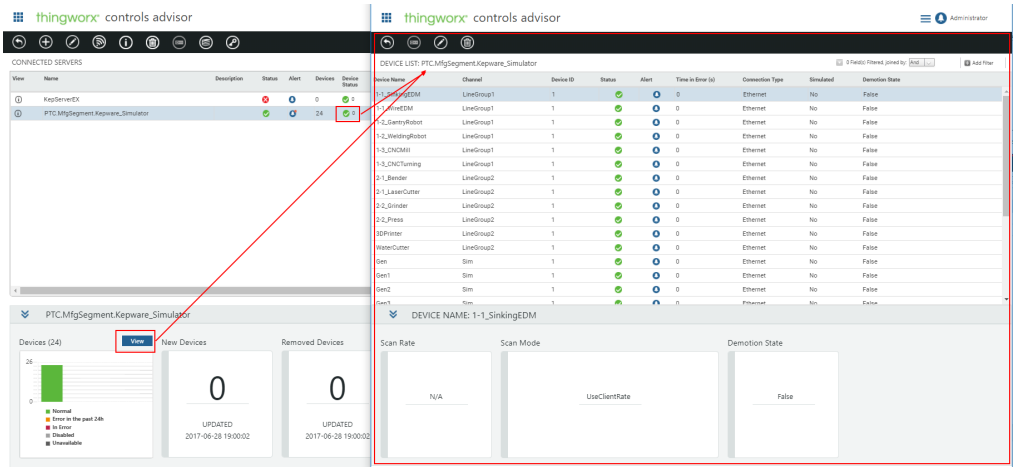
Asset Detail Container Mashup

- Launch Point Key: `AssetDetailContainerMashup`
- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetailContainerMashup`
- Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.C_AssetDetailContainerMashup_[ReleaseVersion]`
- Image:



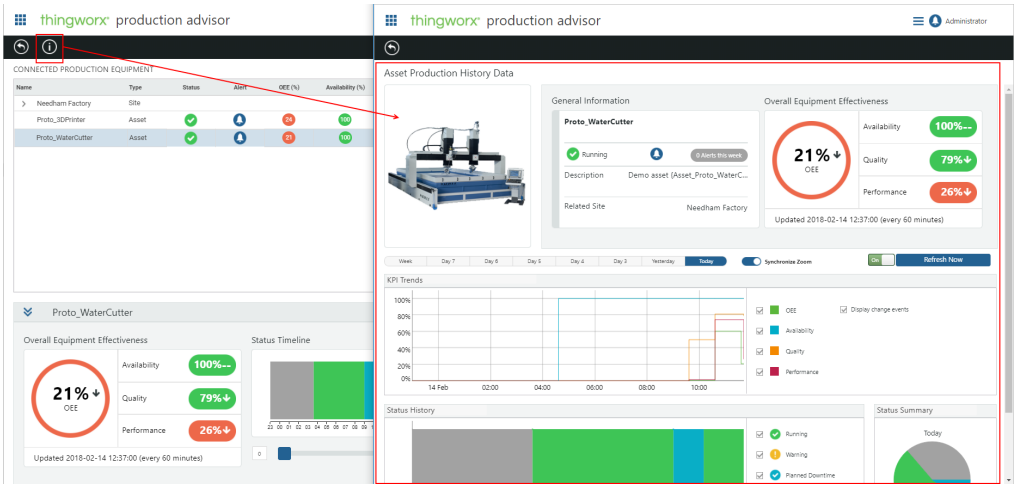
Device List Mashup

- Launch Point Key: `DeviceListMashup`
- Default Mashup: `PTC.Factory.KEPServerEX.DeviceList`
- Duplicate Mashup: `PTC.Factory.KEPServerEX.C_DeviceList_[ReleaseVersion]`
- Image:



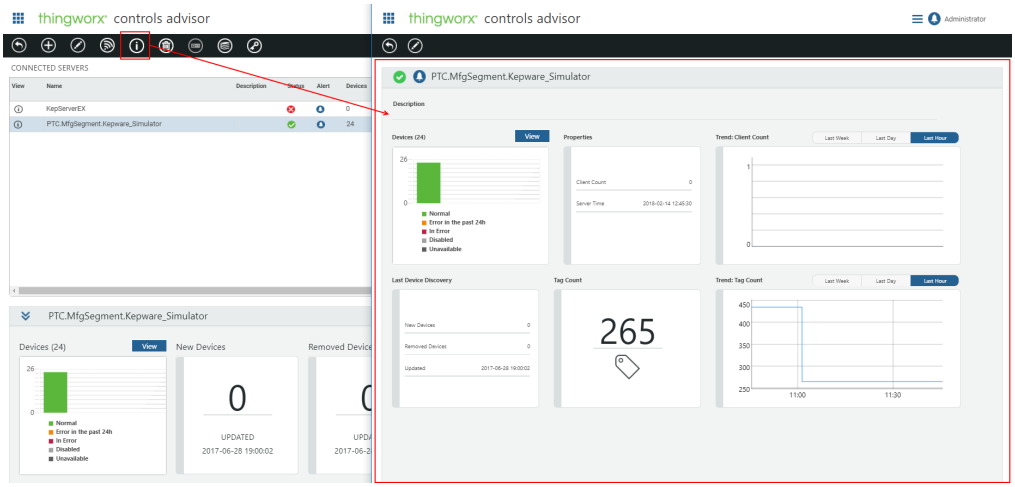
Production Historical Data Mashup

- Launch Point Key: `ProductionHistoricalDataMashup`
- Default Mashup: `PTC.Factory.ProductionHistoricalData`
- Duplicate Mashup: `PTC.Factory.C_ProductionHistoricalData_[ReleaseVersion]`
- Image:



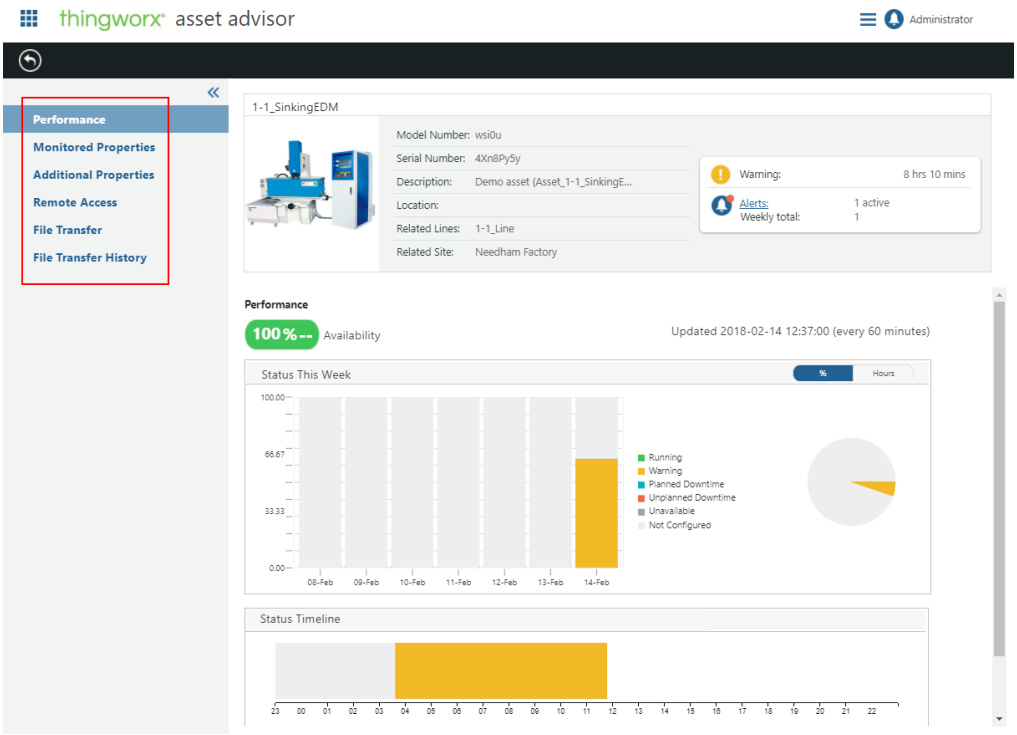
Server Detail Page Mashup

- Launch Point Key: `ServerDetailPageMashup`
- Default Mashup: `PTC.Factory.KEPServerEX.DetailPage`
- Duplicate Mashup: `PTC.Factory.KEPServerEX.C_DetailPage_[ReleaseVersion]`
- Image:



Asset Detail Action Menu

- Launch Point Key: `AssetDetailMenu`
- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.ActionMenu`
- Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.C_ActionMenu`
- Image:



3

Changing Labels in the Application

Modification of the label names must be done in the localization tables.

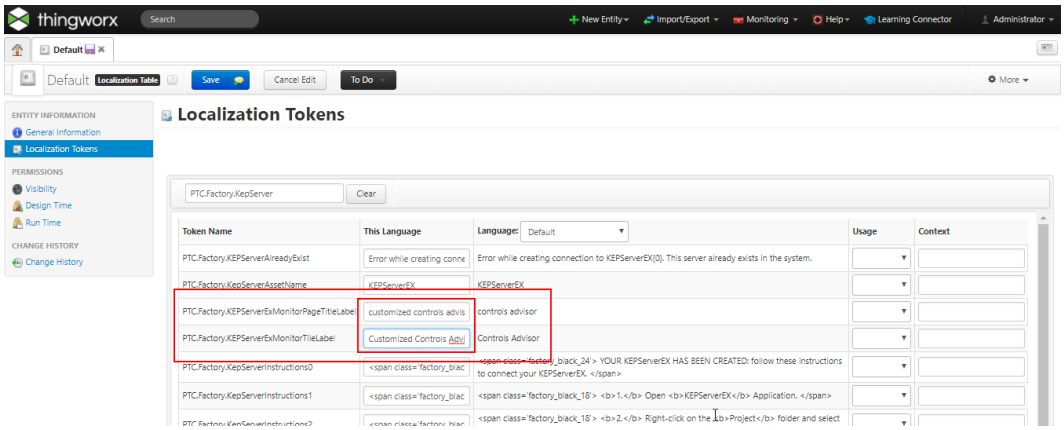
Note

The localization tables are overwritten when new upgrades are installed. To keep your localization table modifications, export the customized localization table before performing an upgrade, and import it back after the upgrade is complete.

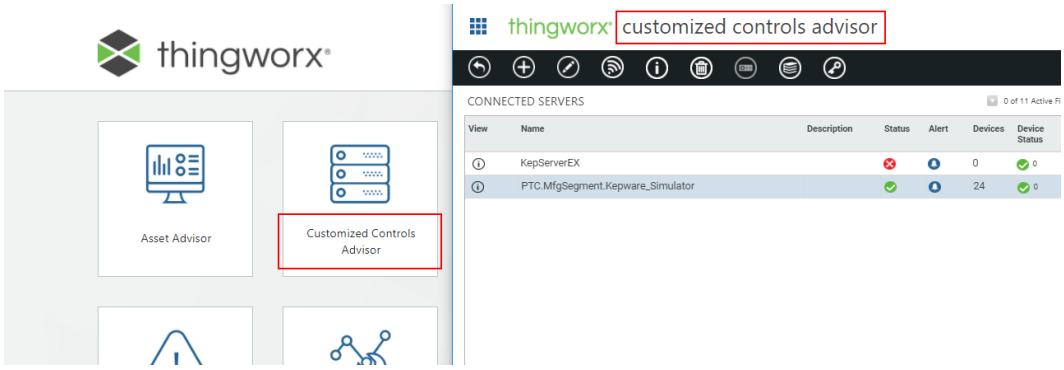
1. In ThingWorx Composer, open **System** ▸ **Localization Tables**.
2. Choose the localization table corresponding to the current language.
3. Modify the applicable tokens.

The following graphics show how to change the **Controls Advisor** label that is shown in the tile and window title.

Changed localization tokens:



Updated application display:

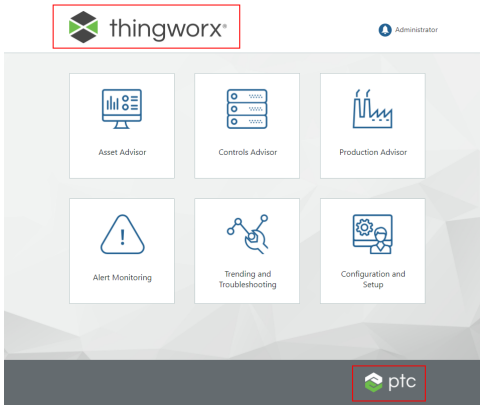


4

Changing the Logos in the Application Console

To change the PTC logos in the application console: 35
To keep the PTC logos and add an additional logo in the master console: 35

A common requirement for PTC customers and partners is the ability to change the logos, or add additional logos in the application console.



Note

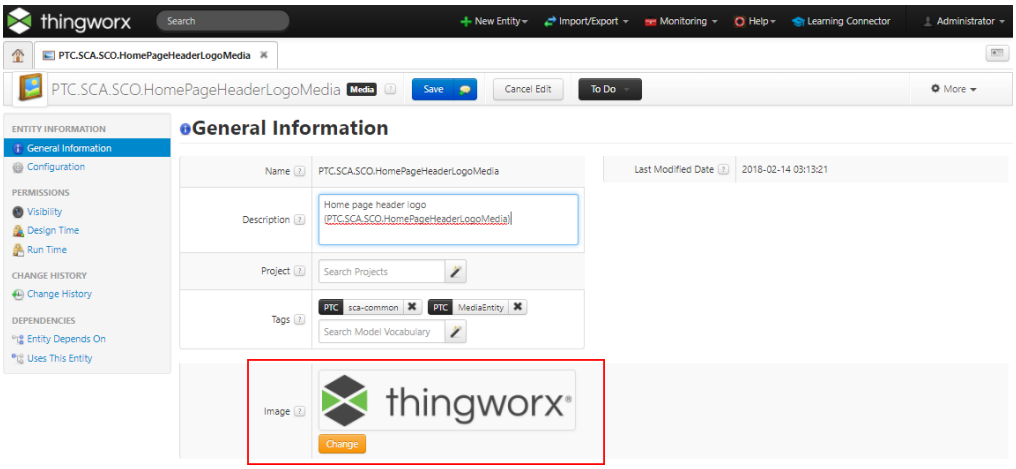
Changes to the media entries for the logos are not retained during an upgrade, and will need to be re-implemented after the upgrade is complete.

To change the PTC logos in the application console:

1. Edit the media entities `PTC.SCA.SCO.HomePageHeaderLogoMedia` and `PTC.SCA.SCO.HomePageFooterLogoMedia` in ThingWorx Composer.

Tip

- The size of the logo is fixed.
- You may need to restart the server to see the changes.



To keep the PTC logos and add an additional logo in the master console:

1. Use image editing software to create your new logo image.
2. In ThingWorx Composer, add the image to one or both of the media entities `PTC.SCA.SCO.HomePageHeaderLogoMedia` and `PTC.SCA.SCO.HomePageFooterLogoMedia`.

5

Customizing the Schedulers

To configure the launch schedule for a scheduler: 37
To enable or disable the scheduler: 37
To set the age of the data to keep after each purge: 38

The following editable schedulers are provided in the extension:

- **PTC.Factory.StatusEvaluationScheduler**—Used for calculating the status of a piece of equipment. It runs every minute.
- **PTC.Factory.MidnightDeviceDiscoveryScheduler**—Used to automatically discover devices for each connected server. It runs daily at midnight. (You can also discover devices for a server as needed from the Controls Advisor.)
- **PTC.Factory.MidnightPurgeWeekOldHistoryScheduler**—Used to purge all historical sensor data which is older than one week. It runs daily at midnight. The age of the data to be kept after each purge can be configured. If all historical data needs to be preserved, disable this scheduler.
- **PTC.SCA.Mfg.KPIsCalculationScheduler**—Used for calculating KPIs. It runs every minute. The actual calculation is performed based on the **KPI Calculation Period** setting on the individual piece of equipment.

For each scheduler, you can configure the launch schedule, as well as enable or disable the scheduler. For **PTC.Factory.MidnightPurgeWeekOldHistoryScheduler**, you can also set the age of data to keep after each purge.

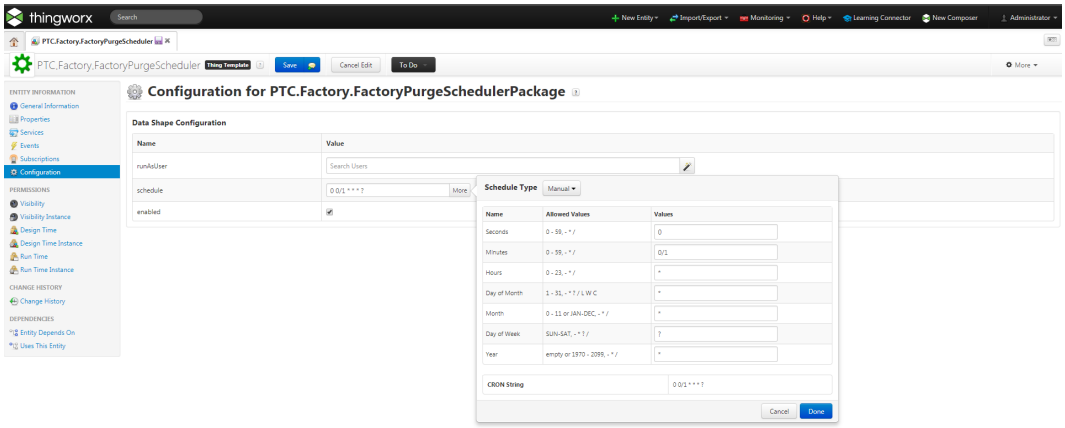
Note

Customizations made to the schedulers are not retained during an upgrade, and will need to be re-implemented after the upgrade is complete.

To configure the launch schedule for a scheduler:

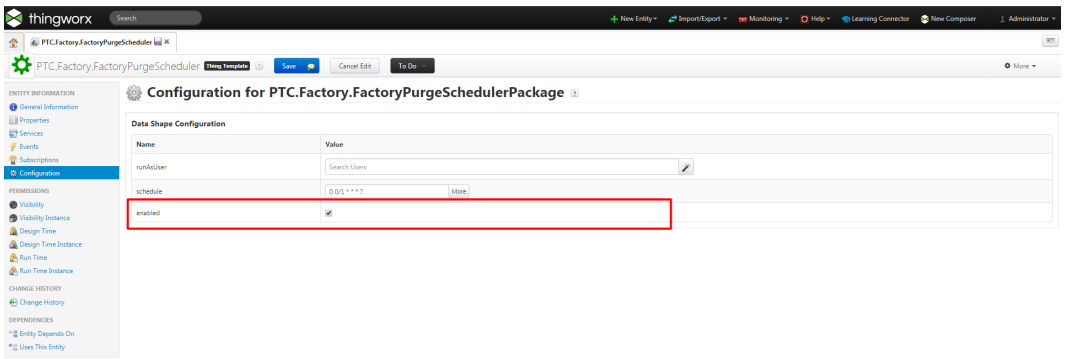
1. Open the scheduler in ThingWorx Composer.
2. Click **Edit** and select **Configuration**.
3. Modify the **schedule** field to set the launch schedule for a scheduler, then click **Done**.

   The **schedule** property uses CRON format. For more information on CRON, see https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm.
4. Click **Save** to save the configuration.



To enable or disable the scheduler:

1. Open the scheduler in ThingWorx Composer.
2. Click **Edit** and select **Configuration**.
3. Select the **enabled** checkbox to enable the scheduler. Clear the **enabled** checkbox to disable the scheduler.

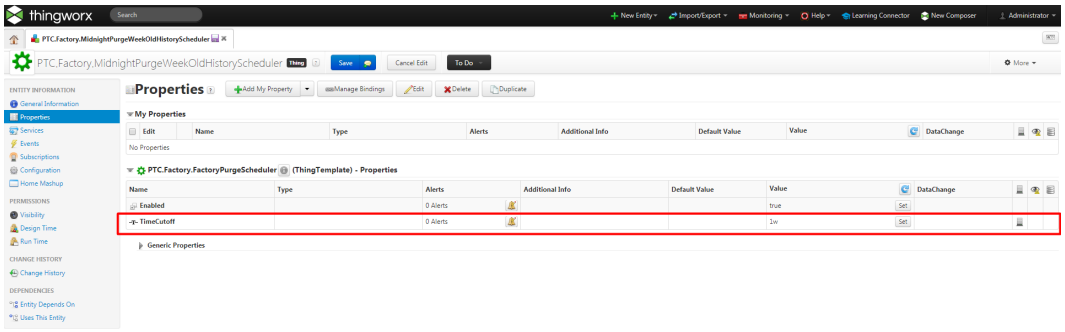


To set the age of the data to keep after each purge:

1. Open `PTC.Factory.MidnightPurgeWeekOldHistoryScheduler` in ThingWorx Composer.
2. Click **Edit** and select **Properties**.
3. Change the **TimeCutoff** property to set the age of the data to be kept after each purge by clicking **Set** in the **Value** column.

   The format required is a combination of numbers and letters: w (weeks), d (days), h (hours), m (minutes), and s (seconds). For example, to purge all data older than 3 and a half days, enter `84h` or `3d 12h`.

   Case, spaces, and order do not affect the value.



6

Customized KPI Evaluation

Instead of using the default KPI evaluation formulas provided out-of-the-box, you can choose your customized formulas to calculate the Availability, Quality, Performance, OEE, Status, and Overall KPIs.

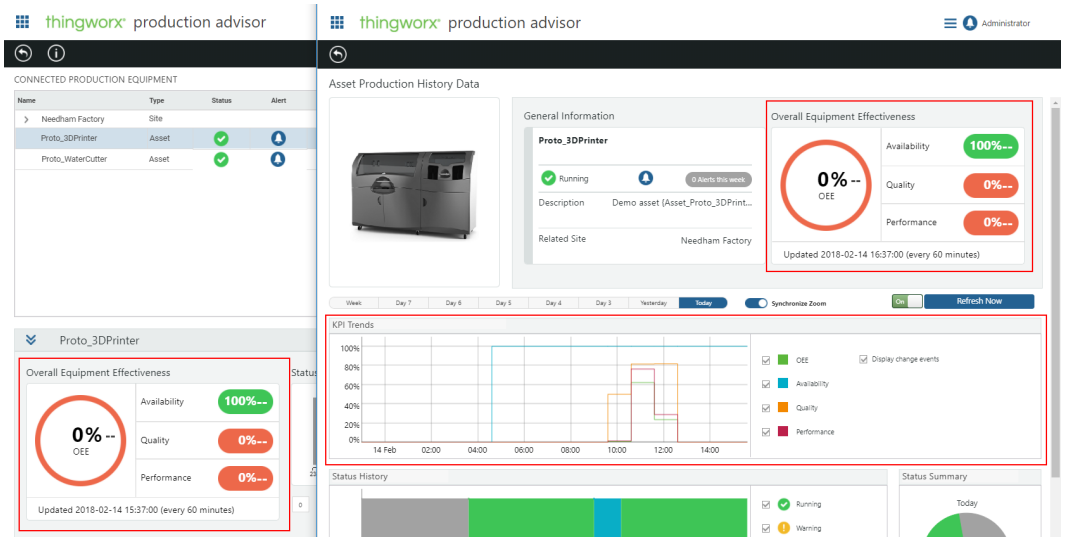
Note

Customizations made to the KPI formulas are not retained during an upgrade, and will need to be re-implemented after the upgrade is complete.

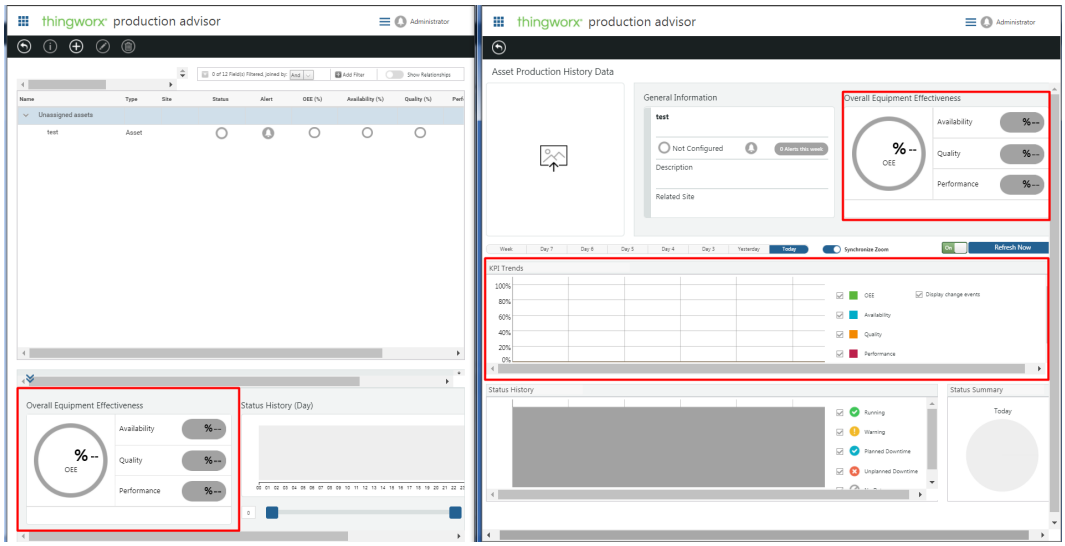
Open the `PTC.Factory.StatusExpressionResourceProvider` thing in ThingWorx Composer, go to the **Services** tag and you will find the following services under `PTC.Factory.StatusExpressionResourceProviderTemplate (ThingTemplate) –Services`.

- **CustomizedAvailabilityCalculation**
- **CustomizedOEECalculation**
- **CustomizedPerformanceCalculation**
- **CustomizedQualityCalculation**
- **CustomizedStatusEvaluation**
- **CustomizedKPIsCalculation**

The input **thingId** is the name of the thing representing the asset or line whose status or KPIs are evaluated. Inputs **quality**, **performance**, or **availability** are the default KPI values calculated out-of-the-box. For **CustomizedStatusEvaluation** and **CustomizedKPIsCalculation**, the input item is a one-row infotable containing the default KPIs. Click the **Allow Override** action button to override these services. You can then implement the customized KPI evaluation formulas or expressions by writing scripts.



The status of the KPIs is displayed in the **Production Advisor** and **Asset Production History Data** page in the application, as shown in the following figure.



7

Creating Custom Roles

You can define your own roles and assign permissions to those roles, in addition to, or instead of the roles provided with the ThingWorx Manufacturing and Service Apps. This enables you to tailor the application to your business processes. Custom roles are set up using ThingWorx Composer. Once created, these custom roles appear in the role assignment section of the **Users** tab of the **Configuration and Setup** page.

To create a custom role:

1. In ThingWorx Composer, click on **User Groups** under **Security**.
2. Click on the **New** button to create a new user group.
3. Enter the name of the role in the **Name** field
4. Select 'PTC.KinexManufacturing' for **Project**.
5. Select the following tags:
   - `PTC:factory-mv`
   - `Role:Factory-UserGroup`
6. Press **Save** to create the user group.
7. Select **FactoryUsers** from the list of **User Groups** and click on the **Edit** button.
8. Click on the **Edit Members** button.
9. Select the new user group from list on the left and move it to the list on the right.
10. Press the **Save** button to save the changes and close the dialog box.
11. Press the **Save** button on the screen for the FactoryUsers user group.

To assign access rights to a custom role:

1. In ThingWorx Composer, click on **User Groups** under **Security**.
2. Click on one of the following user groups to apply the same access rights to your custom role.
   - Controls Engineer
   - Maintenance Engineer
   - Maintenance Manager
   - Production Manager
3. Click on the **Edit Members** button.
4. Select your custom role from list on the left, and move it to the list on the right.
5. Click on **Save** to save the change and close the popup window.
6. Repeat these steps if you would like to apply the rights of another group to your custom role.

> **Note**
>
> If you would like your new custom role to replace an existing role or roles, after completing the previous steps, remove the `Role:Factory-UserGroup` tag from roles that you don't want to appear in the application.

8

Adding Custom Subtypes

Creating a New Thing Template to Use as a Subtype 44
Updating an Existing Thing Template to Use as a Subtype 45
Define the List of Types Displayed When Creating New Equipment 48
Defining a Display Alias for your SubType 48
Defining Launch Points for Custom Subtypes 50

You can add custom subtypes of the default asset type (for ThingWorx Service Apps) or the default site, line, and asset types (for ThingWorx Manufacturing Apps). This allows you to create custom types with pre-defined properties to suit your business needs. Once added, custom subtypes are displayed throughout the ThingWorx Manufacturing and Service Apps in the same manner as the default types are displayed.

Custom subtypes in the apps are achieved in ThingWorx Composer by using templates which derive from the appropriate base templates for your application. You can create a new thing template or update an existing template for this purpose.

The general steps for adding a custom subtype are the following:

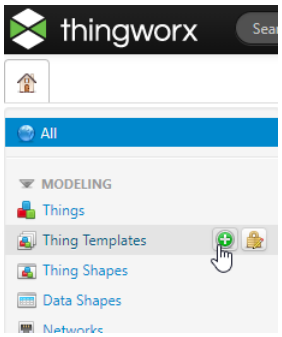
1. Create a new thing template or update an existing thing template.
2. Define the localizable alias for your custom subtype.
3. (Optional) Define which types can be selected when creating new equipment.

These steps are described in more detail in the following sections.

Creating a New Thing Template to Use as a Subtype

To create a new thing template for a subtype:

1. In ThingWorx Composer, click **Thing Templates**, and click the plus sign icon to add a new thing template.



2. Provide a name for the thing template.
3. Select the appropriate **Base Thing Template** for your apps and the type for which you are creating a custom subtype:

   ThingWorx Manufacturing Apps

   - Asset:
     - **Base Thing Template**: `PTC.ISA95.PhysicalAssetThingTemplate`
   - Line:
     - **Base Thing Template**: `PTC.ISA95.ProductionLineThingTemplate`
   - Site:
     - **Base Thing Template**: `PTC.ISA95.SiteThingTemplate`

   ThingWorx Service Apps

   - Asset:
     - **Base Thing Template**: `PTC.SCA.SCO.CSLMAssetThingTemplate`

> **Note**
>
> Ensure that all property names on your subtype thing templates are unique, and do not duplicate any property names on the base thing template.

4. Under **Permissions** ▸ **Visibility**, click **Add Org/Org Units**.
5. Search for and select `PTC.Factory.MachineVisibility`.
6. Click **Add Entire Organization**, then click **Done**.
7. Save the new thing template.

Updating an Existing Thing Template to Use as a Subtype

To update an existing thing template to use as a subtype:

1. In ThingWorx Composer, click **Thing Templates**.
2. Search for your existing thing template.
3. Edit your existing thing template as follows for your apps and the type for which you want a custom subtype.

   ThingWorx Manufacturing Apps

   - Asset:
     - Verify that your base template extends `RemoteThingWithTunnelsAndFileTransfer`.
     - Implement the following thing shapes in the **Implemented Shapes** field:
       - `PTC.Factory.PhysicalAssetThingShape`
       - `PTC.SCA.SCO.SiteThingShape`
       - `PTC.SCA.Mfg.AssetFilteringThingShape`
       - `PTC.SCA.SCO.KPIsThingShape`
       - `PTC.ISA95.EquipmentAssetMappingThingShape`
       - `PTC.ISA95.GeneralPhysicalAssetThingShape`
       - `PTC.SCA.SCO.StatusThingShape`
       - `PTC.ISA95.IdentifierThingShape`
       - `PTC.SCA.SCO.FileTransferHistoryHandlerThing-Shape`
       - `PTC.Factory.ShiftThingShape`
       - `PTC.SCA.SCO.MonitoredPropertiesThingShape`
       - `PTC.SCA.SCO.AssetIdentifierThingShape`
       - `PTC.SCA.SCO.RemoteTunnelingThingShape`

        - PTC.SCA.SCO.AnomalyThingShape
- Line:
    - Implement the following thing shapes in the **Implemented Shapes** field:
        - PTC.SCA.SCO.SiteThingShape
        - PTC.SCA.SCO.StatusThingShape
        - PTC.Factory.ShiftThingShape
        - PTC.SCA.Mfg.AssetFilteringThingShape
        - PTC.SCA.SCO.KPIsThingShape
        - PTC.ISA95.GeneralEquipmentThingShape
        - PTC.ISA95.IdentifierThingShape
        - PTC.ISA95.EquipmentAssetMappingThingShape
        - PTC.SCA.SCO.IdentifierThingShape
        - PTC.ISA95.DisplayNameThingShape
        - PTC.Factory.ProductionLineResourceThingShape
- Site:
    - Implement the following thing shapes in the **Implemented Shapes** field:
        - PTC.Factory.SiteResourceThingShape
        - PTC.ISA95.GeneralEquipmentThingShape
        - PTC.ISA95.IdentifierThingShape
        - PTC.ISA95.EquipmentAssetMappingThingShape
        - PTC.SCA.SCO.IdentifierThingShape
        - PTC.ISA95.DisplayNameThingShape

ThingWorx Service Apps

- Asset:
    - Verify that your base template extends RemoteThingWithTunnelsAndFileTransfer.
    - Implement the following thing shapes in the **Implemented Shapes** field:
        - PTC.SCA.CSLM.GeneralAssetThingShape
        - PTC.SCA.SCO.StatusThingShape
        - PTC.ISA95.IdentifierThingShape
        - PTC.SCA.SCO.FileTransferHistoryHandlerThing-Shape
        - PTC.Factory.ShiftThingShape

- `PTC.SCA.SCO.MonitoredPropertiesThingShape`
- `PTC.SCA.SCO.AssetIdentifierThingShape`
- `PTC.SCA.SCO.RemoteTunnelingThingShape`
- `PTC.SCA.SCO.AnomalyThingShape`

4. Under **Permissions ▸ Visibility**, click **Add Org/Org Units**.
5. Search for and select `PTC.Factory.MachineVisibility`.
6. Click **Add Entire Organization**, then click **Done**.
7. Repeat steps 4 through 6 for **Permissions ▸ Visibility Instance**.
8. Under **Permissions ▸ Run Time Instance**, search for the following user groups, and set the following permissions for them:

Group or User	Property Read	Property Write	Event Execute	Event Sub-scribe	Service Execute
Controls Engineer	Allow	Allow	Allow	Allow	Allow
Mainte-nance Engineer	Allow	Allow	Allow	Allow	Allow
Production Engineer	Allow	Use Inherited	Allow	Allow	Allow
Mainte-nance Manager	Allow	Allow	Allow	Allow	Allow

9. Under **Permissions ▸ Design Time Instance**, search for the following user groups, and set the following permissions for them:

Group or User	Read	Update	Delete
Controls Engineer	Allow	Allow	Allow
Maintenance Engineer	Allow	Allow	Allow
Maintenance Manager	Allow	Allow	Allow

10. Save the updated thing template.

Define the List of Types Displayed When Creating New Equipment

The thing templates added or updated for use as subtypes as described in the previous sections automatically display in the **Type** drop-down list when creating new equipment, along with the default types. You can customize this list to show only certain types and subtypes by excluding those types that you do not want to be selectable.

1. In ThingWorx Composer, open `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`.
2. Click **Configuration**, and scroll down to the **ResourceCreationSettings** configuration table.
3. Add the template which you want omitted from the **Type** list on the **Create Equipment** window to the **ExcludedThingTemplate** list.

Defining a Display Alias for your SubType

Defining a localizable alias for your subtype allows for localized display names to be provided.

To define a localizable alias for your new subtype thing template, complete the following steps:

1. Create a localization file with the following structure:

```
<?xml version="1.0" encoding="utf-8"?>
<Entities>
<LocalizationTables>
<LocalizationTable name="Default" description="Default localization
table" homeMashup="" tags="PTC:factory-mv">
<avatar/>
<DesignTimePermissions>
<Create/>
<Read/>
<Update/>
<Delete/>
<Metadata/>
</DesignTimePermissions>
<RunTimePermissions/>
<VisibilityPermissions>
<Visibility>
<Principal isPermitted="true" name="PTC.Factory.MachineVisibility"
type="Organization"/>
</Visibility>
</VisibilityPermissions>
<ConfigurationTables>
```

```
<ConfigurationTable isMultiRow="true" name="LocalizationTokens"
description="Localization Tokens" ordinal="0">
<DataShape>
<FieldDefinitions>
<FieldDefinition name="name" description="Token name"
aspect.friendlyName="Token name" baseType="STRING" ordinal="0"/>
<FieldDefinition name="value" description="Token value"
aspect.friendlyName="Token value" baseType="STRING" ordinal="1"/>
</FieldDefinitions>
</DataShape>
<Rows>
<Row>
<name><![CDATA[Customer1.Test.displayName]]></name>
<value><![CDATA[Hello World]]></value>
</Row>
<Row>
<name><![CDATA[Customer1.Test2.displayName]]></name>
<value><![CDATA[Hello World]]></value>
</Row>
</Rows>
</ConfigurationTable>
</ConfigurationTables>
<ConfigurationChanges/>
</LocalizationTable>
</LocalizationTables>
</Entities>
```

2. In the `LocalizationTable` element, change the `name` attribute value to the appropriate language short name:

Language	Short Name
Default Language	`default`
Japanese	`ja`
Chinese	`zh-CN`
French	`fr`
Italian	`it`
Korean	`ko`
German	`de`

3. To add a new alias, locate the `Rows` element.
4. Add a new row for each alias you want to add. For each row, include the following elements:

```
<name><![CDATA[thing template name.displayName]]></name>
<value><![CDATA[the alias for the subtype]]>
```

5. Save the localization file.

6. Import the localization file into ThingWorx Composer.
    a. Select **Import/Export** ▸ **From File**.
    b. Locate the localization file and select it.
    c. Click **Import**. A success message displays when the import completes.
7. Repeat steps 1 through 6 for each language in which you want a localizable alias for this subtype thing template.

Defining Launch Points for Custom Subtypes

Certain launch points can be defined to launch custom mashups or a custom menu for individual custom subtypes.

The following launch points can point to custom mashups based on the subtype of the piece of equipment:

- `PlantStatusFooterMashup`
- `AssetDetailContainerMashup`
- `ProductionHistoricalDataMashup`

The following launch point can point to a custom menu based on the subtype of an asset:

- `AssetDetailMenu`

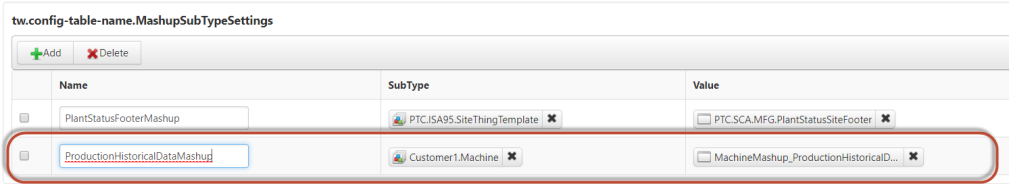
Note

To create custom mashups in addition to the provided duplicates, you must have a Developer Edition, Premium, or Enterprise license.

To define a custom mashup or menu to display for specific subtypes:

1. In ThingWorx Composer, open `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`, and click **Configuration**.
2. For a custom mashup, add an entry to the **MashupSubTypeSettings** table with the following information:
    - **Name**—the launch point key as listed in the **LaunchPointEntitySettings** table.
    - **SubType**—the name of the thing template for the subtype.
    - **Value**—the name of the custom mashup.

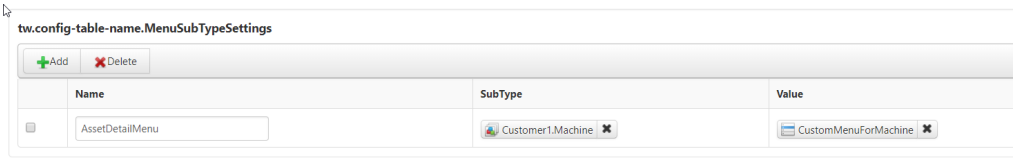
For example, to define the `MachineMashup_ProductionHistoricalDataMashup` custom mashup to launch for the `ProductionHistoricalDataMashup` launch point for `Customer1.Machine` asset subtypes, add an entry to the **MashupSubTypeSettings** table, as shown in the following figure:



For a custom menu, add an entry to the **MenuSubTypeSettings** table with the following information:

- **Name**—`AssetDetailMenu`
- **SubType**—the name of the thing template for the asset subtype.
- **Value**—the name of the custom menu.

For example, to define the `CustomerMenuForMachine` custom menu to launch for `AssetDetailMenu` launch point for `Customer1.Machine` asset subtypes, add an entry to the **MenuSubTypeSettings** table, as shown in the following figure:



9

Customizing the Tag Picker Common Component

Disable Preservation of the Last Selection.....53
Browse Data from Custom Connectors 53
Using the Tag Picker Common Component in a Mashup.....55

The tag picker common component can be customized in the following ways:

- Disable the preservation of the last selection within a session.
- Browse data from new custom connectors.

Disable Preservation of the Last Selection

By default, the last selection made by a user in the tag picker is preserved within a single session.

To disable this selection preservation:

1. In ThingWorx Composer, open `PTC.Factory.Administration.TagConfigurationUtils`.
2. In **Properties**, change the value of the **isEquipmentSelectionPreserved** property to false.
3. Click **Save**.

Browse Data from Custom Connectors

You can customize the tag picker common component to browse data from custom connectors.

1. In ThingWorx Composer, create a new thing template with the following settings, to add the new connector equipment type to the **Equipment Type** drop-down list:
   - **Name**—The name for the thing template, for example `My_Connector_ResourceProvider_Thing_Template`.
   - **Base Thing Template**—**GenericThing**
   - **Implemented Shapes**—`PTC.SCA.SCO.RemoteConnectionResourceProvider-ThingShape`
   - Click **Services**, and override the services in the new thing template that pertain to the `PTC.SCA.SCO.RemoteConnectionResourceProviderThingShape` thing shape. For each service, click  to override and edit each service by adding appropriate scripts for your connector.
     - **BindServerTags**—This service is used to bind tags defined in a remote server to properties on the remote server thing. A new property is created if a tag has never been bound. This service is not needed if no remote bindings are needed.
     - **GetConnectedServers**—(optional) This service is used to retrieve all the connected servers, resources, or equipment for a given thing template. You can optionally provide your own script for this, or use the service as implemented.
     - **GetServerTags**–This service browses tags for a given path and type filter.

- **GetServerTreeStructure**—This service retrieves the next level of a tree structure for a given node of the connected server.
- **RemovePropertyBinding**—(optional) This service removes the binding between the server tag and the target thing. It also removes the property itself from the target thing. You can optionally provide your own script for this service, or use the service as implemented.

For example scripts, refer to the services implemented on `PTC.SCA.SCO.NITestStandResourceProviderThingTemplate` and `PTC.Factory.KepServerResourceProviderThingTemplate`.

2. Create a resource provider thing implementing the thing template created in step 1:
   - **Name**—For example, `My_Connector_ResourceProviderThing`.
   - **Base Thing Template**—Specify the new thing template created in step 1. In this example, `My_Connector_ResourceProvider_Thing_Template`.
3. Create a thing template representing the remote thing that you are attempting to expose:
   - **Name**—For example, `My_RemoteThing_ThingTemplate`.
   - **Base Thing Template**—**RemoteThing**
4. Create a remote thing to represent the individual piece of equipment:
   - **Name**—For example, `My_NewEquipment`.
   - **Base Thing Template**—Specify the thing template created in step 3. In this example, `My_RemoteThing_ThingTemplate`.
5. Add your new equipment type to the launch point configuration thing.
   a. Open `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`.
   b. Click **Configuration**.
   c. Add a new row to the **RemoteConnectionSettings** table with the following settings:
      - **ConnectionType**—Enter the name that you want displayed for the connection type in the **Equipment Type** drop-down list on the tag picker. For example, `New Equipment Type`.
      - **ResourceProviderName**—Enter the resource provider thing created in step 2. In this example, `My_Connector_ResourceProviderThing`.

- **RemoteConnectionThingTemplateName**—Enter the name of the thing template representing the remote thing created in step 3. In this example, `My_RemoteThing_ThingTemplate`.
- **Enabled**—Select this checkbox to make the connection type visible in the tag picker.

  As a best practice, disable an equipment type by clearing the **Enabled** checkbox for the table row, rather than deleting the row.

Note

The newly created equipment type does not show in the **Equipment Type** drop-down list on the tag picker unless there is a connected remote thing representing that equipment type.

The tags and properties in the tag picker can be displayed as a table column or as a tree view. For more information, see the documentation related to the Grid Advanced extension, available from the ThingWorx Marketplace, at the following URL: https://marketplace.thingworx.com/tools/gridadvanced.

Using the Tag Picker Common Component in a Mashup

To use the tag picker common component in a mashup, add a **Navigation** widget with the following settings:

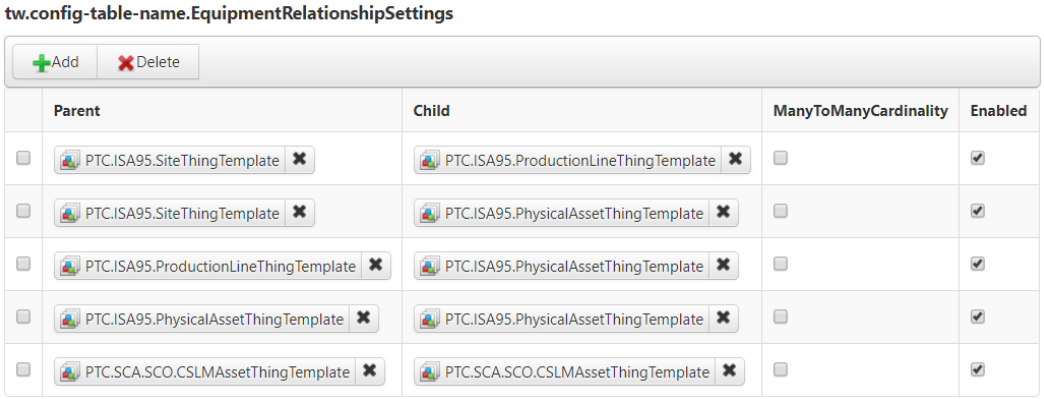
Property	Value
MashupName	Search for and select `PTC.Factory.CommonTagPicker`.
TargetWindow	Select **Modal Popup**.
addTitle	Enter a localization token representing the title for the tag picker window. If this property is not specified, the tag picker title displays as **???**.
sourceName	Leave this field blank. (This is the name of the thing where the tag is coming from. This value is filled in when the user selects a value in the **Equipment** drop-down list.)
bindOnOK	Select this checkbox for the tags and properties selected in the tag picker to bind to the target thing when the user clicks **OK**.
Determine what displays for the **Equipment Type** field on the tag picker:	
providerClasses	If this field is left empty, only remote connections display, such as KEPServerEX. To additionally display other

Property	Value
	equipment types, enter the provider thing name for those equipment types. For example, to also display devices, lines, and assets, enter: `PTC.Factory.DeviceResourceProvider;PTC.Factory.ProductionLineResourceProvider;PTC.SCA.SCO.AssetResourceProvider`.
resourceType	Provides an additional filter for the **Equipment Type** drop-down list. If this field left empty, all available equipment types display for the **Equipment Type** drop-down list. If you want to limit display to a single type, enter the equipment type name, for example, `Asset`. This value is case-insensitive.
Determine what displays for the **Equipment** field on the tag picker:	
filterName	If left blank, all available equipment for the selected **Equipment Type** displays in the **Equipment** list. If you want to limit **Equipment** to only certain equipment, enter a regular expression. This value is case-insensitive. This property filters on the **Name** of the thing as shown in ThingWorx Composer, rather than filtering on its display name.
Determine whether users can select a single tag or property on the tag picker, or can select multiple tags or properties:	
isMultiSelect	Select to enable multi-selection on the tag picker, and to make the **Apply** button visible. **Note** To limit the number of selectable properties, create an **Expression** widget in the mashup itself.
informationMessage	Optionally enter a message to display next to the **Equipment** drop-down list. For example, `Select up to 5`.
Determine the data types of tags or properties to display on the tag picker:	
typeFilter	A semi-colon separated list of the types (ThingWorx base types) of tags or properties that can be selected. If left empty, then all types of tags or properties display. For example, to show only string and boolean properties, enter `STRING;BOOLEAN`.

10

Customizing Equipment Structure Relationship Rules

When adding related child equipment from the **Equipment Structure** page for a piece of equipment, the equipment available to be added as a related child is determined by the defined equipment relationship rules. These rules are defined in the `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`, under **Configuration**, in the **EquipmentRelationshipSettings** configuration table.



Each rule in the table defines a thing template that can be added as a child of another thing template, whether the relationship is one-to-one (the default) or can be many-to-many, and if the rule is enabled in the system.

Note

Rules for `PTC.SCA.SCO.CSLMAssetThingTemplate` apply only for ThingWorx Service Apps.

Rules for `PTC.ISA95.PhysicalAssetThingTemplate` apply only for ThingWorx Manufacturing Apps.

The initial rules shown in the preceding screenshot have the following impact:

- Sites and site subtypes (ThingWorx Manufacturing Apps):
  - Can only be created at the first level in the equipment structure hierarchy.
  - Can have multiple lines and assets as children.
- Lines and line subtypes (ThingWorx Manufacturing Apps):
  - Can be created at the first level in the equipment structure hierarchy.
  - Can have only one parent site or site subtype.
  - Can have multiple assets or asset subtypes as children.
- Assets and asset subtypes (ThingWorx Manufacturing Apps and ThingWorx Service Apps)
  - Can be created at the first level in the equipment structure hierarchy.
  - Can have only a single parent, either a line or line subtype, or a site or site subtype. (ThingWorx Manufacturing Apps only)
  - Can have multiple assets or asset subtypes as children.

When creating a new piece of equipment from **Configuration and Setup ▸ Equipment**, these rules determine what types of equipment can be created based on the row selected in the table (if any). If no row is selected, then any type can be created. If a row in the table is selected, only the types allowed as children of the selected piece of equipment can be created.

These rules also apply when adding related child equipment from the **Equipment Structure** page when configuring a piece of equipment. Only equipment of types allowed as children of the current piece of equipment can be added.

Rules can be disabled by clearing the **Enabled** checkbox, and re-enabled by selecting the checkbox. Rules can also be added and deleted.

Best Practice

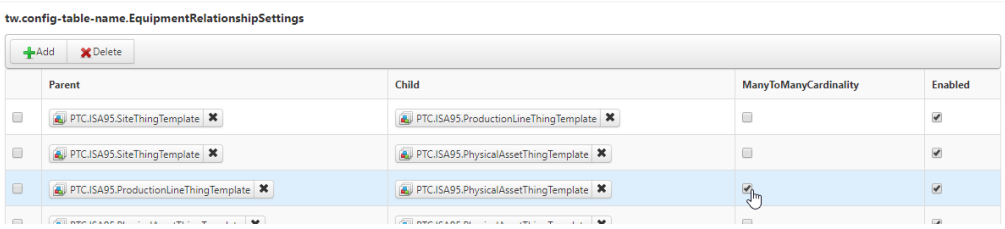
As a best practice, if you do not want the default rules to be enforced, disable them rather than deleting them.

When changing the initial relationship rules, keep the following in mind:

- To allow any type to be related as a child of any other type, create explicit rules for every relationship, and select the **ManyToManyCardinality** checkbox.
- To allow a child type to have multiple parents of a given type, select the **ManyToManyCardinality** checkbox for that relationship.
- To allow a child type to have multiple parents of any type, select the **ManyToManyCardinality** checkbox for all of those relationships.
- If a given child type is involved in multiple relationship rules, and one of those rules has the **ManyToManyCardinality** checkbox cleared, then that relationship rule prevents this child type from having multiple parents.

  For example, if the **ManyToManyCardinality** checkbox is selected for the **Parent**=`PTC.ISA95.ProductionLineThingTemplate` to **Child**= `PTC.ISA95.PhysicalAssetThingTemplate` rule, the result is as follows:

  - Users are able to relate a given asset to multiple lines, provided that this asset is not already related to a site or another asset.
  - If the asset is related to a site or another asset, users are not able to relate that asset to any other type of equipment.



- If there are no rules specified or enabled for a particular type, no related child equipment can be added for that type.
- Custom subtypes can have individually defined relationship rules.
  - If there is no explicit rule for a subtype, it automatically inherits the relationship rules of the type on which it is based.

- If the base thing templates for site, line, and asset are not defined as parents in any rule, or those rules are all disabled, individual rules for the custom subtype thing templates must be explicitly defined.

Adding a New Relationship Rule

To add a new relationship rule:

1. On the **EquipmentRelationshipSettings** configuration table, click **Add**.
2. Specify the thing templates of the parent and child equipment types for this relationship rule.
3. Leave the **ManyToManyCardinality** checkbox clear for the default one-to-one relationship between these types of equipment. Select the checkbox to allow a many-to-many relationship between these types of equipment.
4. Leave the **Enabled** checkbox selected for the rule to be enabled. Clear the checkbox for the rule to be disabled.
5. Click **Save**.

Deleting Rules

To delete relationship rules:

1. Select one or more relationship rules from the **EquipmentRelationshipSettings** configuration table.
2. Click **Delete**.
3. Click **Save**.

11

Customizing Tab Pages in the Configuration and Setup Tile

Changing the Main Mashup....62
Adding Tab Pages in the Configuration and Setup Main Mashup....62
Granting Access Control to the Tab Page....64
Modify the Existing Tab Pages....65

The **Configuration and Setup** tile includes tab pages used for configuring equipment, alerts, notification delivery, and users.

You can customize the tabs in the **Configuration and Setup** tile by adding and removing tab pages, or by modifying the existing tab pages.

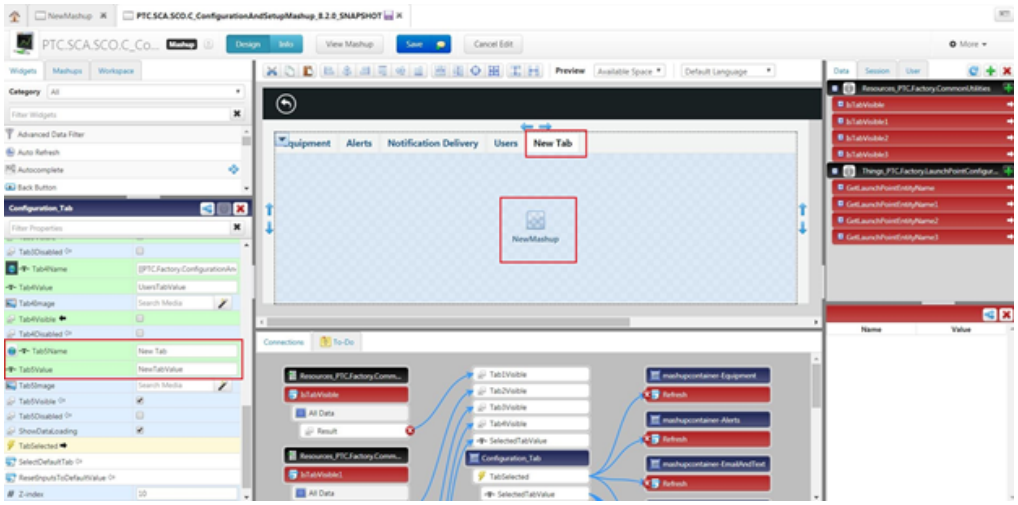
Changing the Main Mashup

To add or remove tab pages from the **Configuration and Setup** tile, the main mashup for the tile must be updated. The duplicate mashup `PTC.SCA.SCO.C_ConfigurationAndSetupMashup_[ReleaseVersion]` is provided for this purpose. For more information on customizing mashups, see the example in Changing the Tiles in the Main Application Console on page 18.

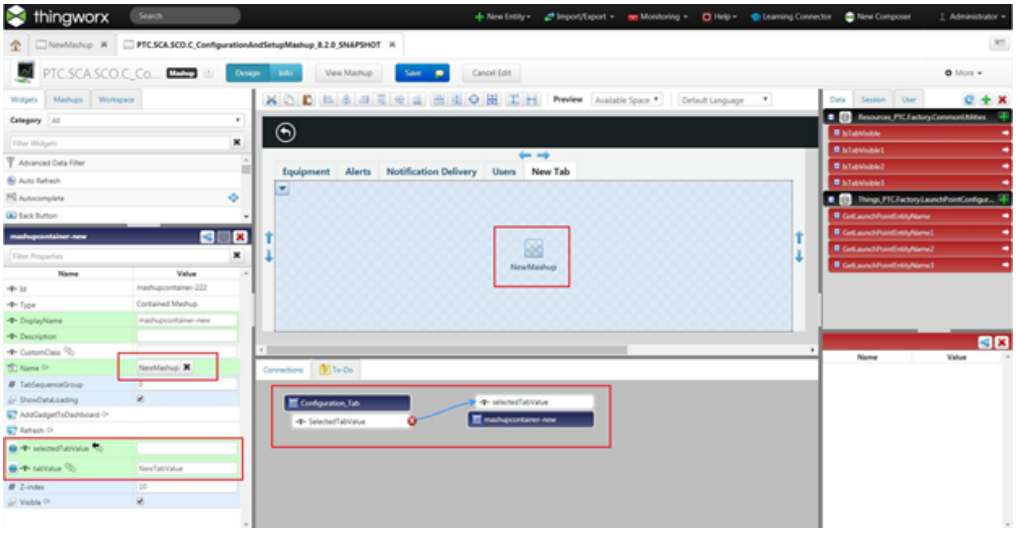
Adding Tab Pages in the Configuration and Setup Main Mashup

To add a new tab page to the **Configuration and Setup** tile, complete the following steps:

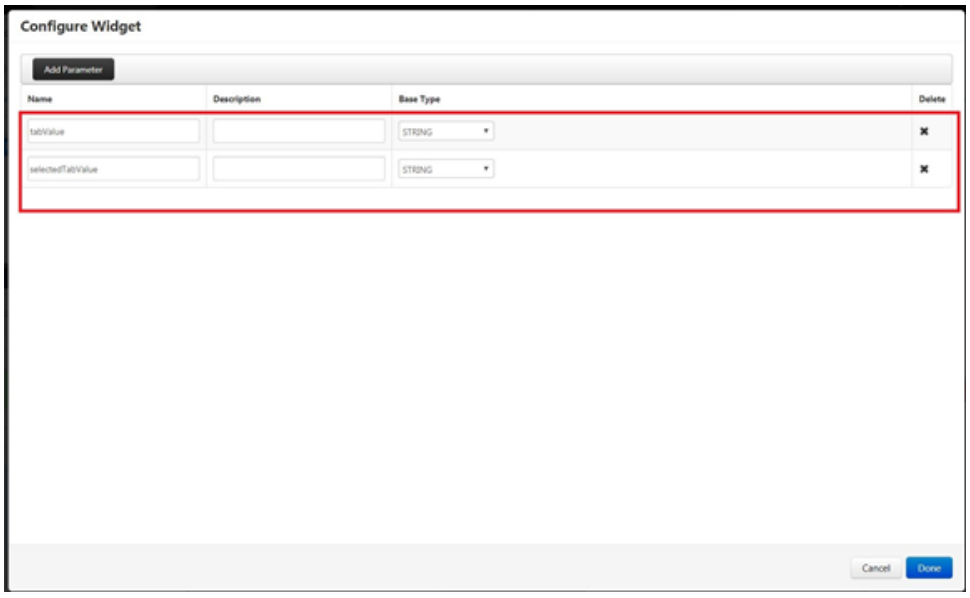
1. Create a new mashup for the new tab page.
2. From `PTC.SCA.SCO.C_ConfigurationAndSetupMashup_[ReleaseVersion]`, select the `Tabs` widget and add a new tab page by increasing the value of `NumberOfTabs`.
3. Add a mashup container inside the new tab page. Bind the name of the new mashup container to the new mashup created in step 1.



4. From the main mashup:
    - Bind the tab's `SelectedTabValue` to the `selectedTabValue` parameter of the container mashup.
    - Set the `tabValue` parameter of the container mashup with the tab value of the new tab page.



5. From the new mashup created in step 1, complete the following steps:
    a. Add two parameters with the `STRING` type, named `tabValue` and `selectedTabValue`.



b. Add a validator called `validator-loadContent` with two `STRING` type input parameters (`tabValue` and `selectedTabValue`) and bind them from the mashup parameters created in step 5a. Set the validator to be triggered by the `Load` event and `RefreshRequested` event of the mashup.

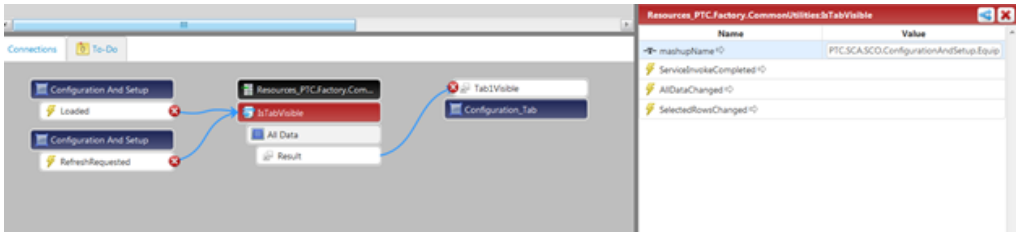
   The validator will verify if the current tab page is the selected one. If yes, it will trigger all the behaviors for loading and refreshing the mashup.

Granting Access Control to the Tab Page

Complete the following steps to have access control for your added tab page:

1. Create a contained mashup, and then grant visibility to the desired organization. See "Organizations" in the ThingWorx help for more information on how to grant visibility to a mashup. Here is an example of a contained mashup called `PTC.SCA.SCO.ConfigurationAndSetup.EquipmentConfigurationMashup` with some visibility.

2. Once your mashup has been created, call a service called `IsTabVisible` from the `PTC.Factory.CommonUtilities` resource, and provide your contained mashup's name as an input parameter. This service needs to be triggered by the `Loaded` and `RefreshRequested` events. Bind the output of the service to the configuration tab property called `TabXVisible`.



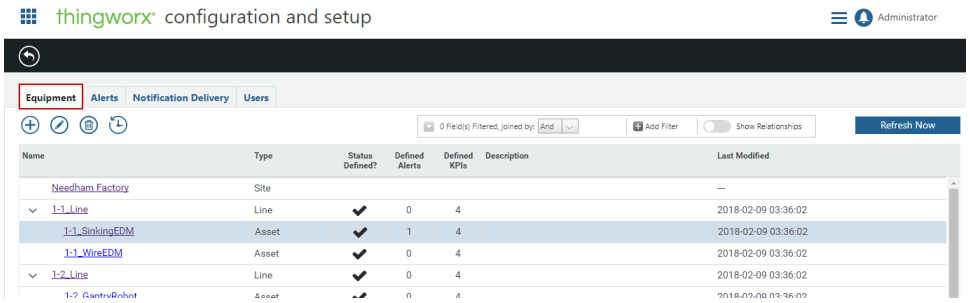
3. Call the `getTabMashupName` service from the `PTC.Factory.CommonUtilities` resource and provide your contained mashup name as an input parameter. This service needs to be triggered by the `Loaded` and `RefreshRequested` events. Bind the output of the service to the contained mashup name parameter.

4. Also bind your contained mashup name parameter to the default `PTC.SCA.SCO.ConfigurationAndSetup.DummyTabMashup` mashup.

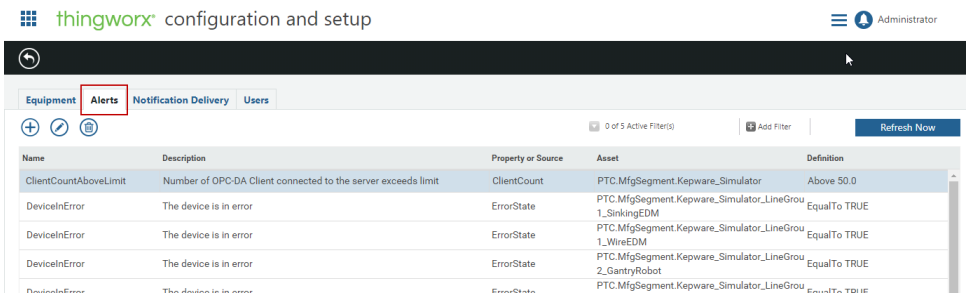
Modify the Existing Tab Pages

The existing tab pages for the **Configuration and Setup** tile can be modified in a duplicate mashup or replaced by a new mashup. From the configuration table of `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`, change the launch point to the modified duplicate mashup or the new mashup.

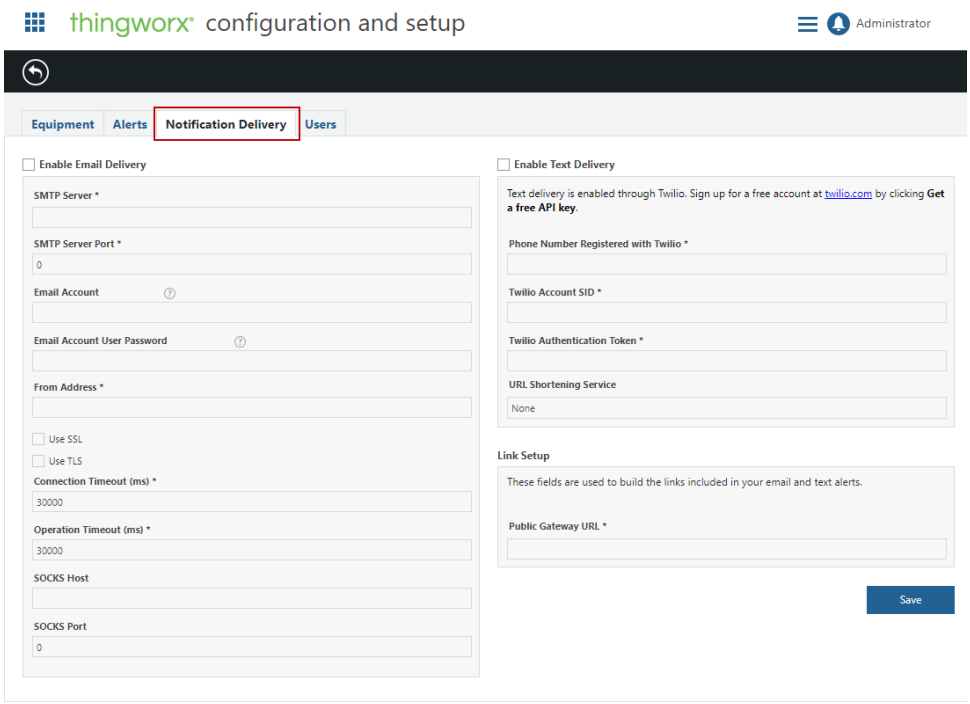
- **Equipment** tab:
  - Launch Point Key: `EquipmentConfigurationMashup`
  - Default Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.EquipmentConfigurationMashup`
  - Duplicate Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.C_EquipmentConfigurationMashup_[ReleaseVersion]`
  - Image:



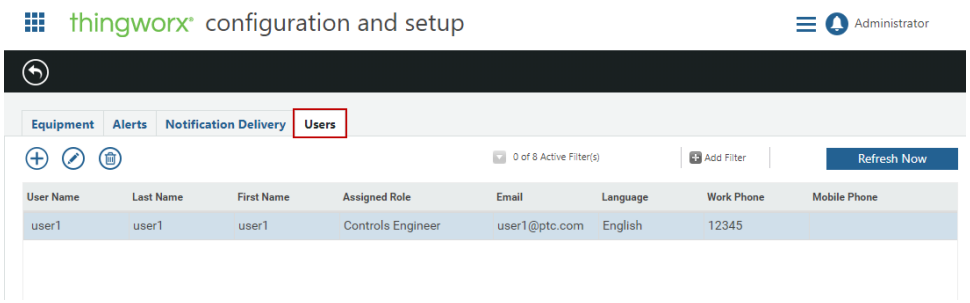
- **Alerts** tab:
  - Launch Point Key: `AlertsConfigurationMashup`
  - Default Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.AlertsConfigurationMashup`
  - Duplicate Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.C_AlertsConfigurationMashup_[ReleaseVersion]`
  - Image:



- **Notification Delivery** tab:
  - Launch Point Key: `EmailAndTextConfigurationMashup`
  - Default Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.EmailAndTextConfigurationMashup`
  - Duplicate Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.C_EmailAndTextConfigurationMashup_[ReleaseVersion]`
  - Image:



- **Users** tab:
  - Launch Point Key: `UsersConfigurationMashup`
  - Default Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.UserManagerMashup`
  - Duplicate Mashup: `PTC.SCA.SCO.ConfigurationAndSetup.C_UserManagerMashup_[ReleaseVersion]`
  - Image:



12

Adding a Custom Notification Handler

Two notification delivery methods are provided with the ThingWorx Manufacturing and Service Apps: email and text (SMS) delivery. These delivery methods are enabled and configured in **Configuration and Setup ▸ Notification Delivery**. By default, individual users must have their **Notification Preference** configured in **Configuration and Setup ▸ Users** to be added as a recipient of an alert notification.

To use a different delivery method for notifications, you can create a new notification handler, and set it as the **AdditionalAlertNotificationHandler** on the **Configuration** page for the `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`. Any user can then be added as a recipient on an alert.

To create a new notification handler:

1. Create a new data shape, for example `MyNotificationHandlerContent`, and click **Save**. This data shape is where, if needed, you can define content for your notification. For examples, see `SMSContent` and `EmailContent`.
2. Create a new thing template, for example `MyNotificationHandlerTemplate`, with a **Base Thing Template** of `NotificationHandler`.

3. On the `MyNotificationHandlerTemplate`, click **Services**, then click **Add My Service** to add a new service with a **Service Name** of `Notify`. On the **Inputs/Outputs** tab for the service, add the following input parameters:

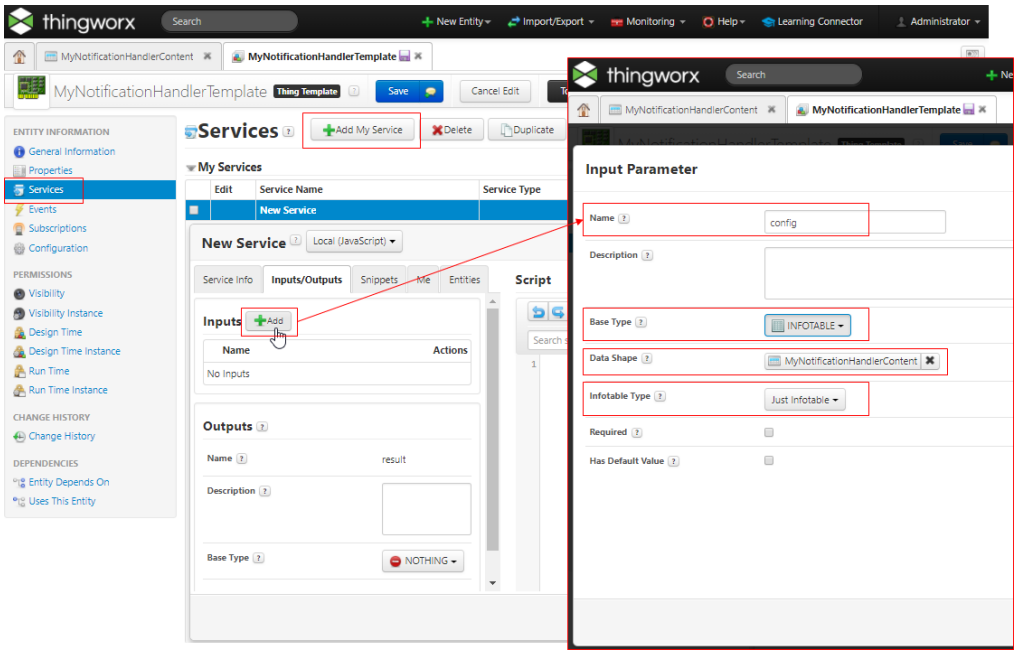
Input Parameter Properties			
Name	**Base Type**	**Data Shape**	**Infotable Type**
`definition`	**NOTIFICATIONDE-FINITIONNAME**	—	—
`event`	**INFOTABLE**	**Event**	**Just Infotable**
`config`	**INFOTABLE**	Select the data shape created in step 1, for example **MyNotification-HandlerContent**	**Just Infotable**

   a. Click **Add** next to **Inputs**.

   b. On the **Input Parameter** window, enter the properties for an input parameter, as shown in the table.

   c. Click **Done**.

   d. Repeat steps 3a through 3c for each input parameter.



   Click **Done** to create the new service.

4. Click **Save**.

5. Create a new thing, for example `MyNotificationHandler`, with the **Thing Template** set to the thing template created in step 2. Click **Save**.

6. Create a localization token for your notification handler thing, `notificationHandlers.MyNotificationHandler`. For more information, see "Localization Tables" in the ThingWorx Help Center.
7. From the **More** menu at the top right of the thing, select **Export for Source Control**.
8. Open the exported file in a text editor.
9. Locate the `ConfigurationTable` tags. Inside the `ConfigurationTable` tags, find the empty `Rows` tags.
10. Add the following content inside the `Rows` tags:

```
<Row>
     <configuration><![CDATA[<DataShape>]]></configuration>
     <handlerID><![CDATA[<UUID>]]></handlerID>
     <localizedName><![CDATA[[[<TokenName>]]]]></localizedName>
     <serviceName><![CDATA[<Service>]]></serviceName>
</Row>
```

Replace the variables as follows:

- *<DataShape>*—Name of the data shape created in step 1.
- *<UUID>*—A universally unique identifier (UUID).
- *<TokenName>*—The localization token created in step 6.
- *<Service>*—Service created in step 3.

For example:

```
<Row>
     <configuration><![CDATA[MyNotificationHandlerContent]]></configuration>
     <handlerID><![CDATA[90ded6fc-7fd7-4141-80ee-34aea5e6fb71]]></handlerID>
     <localizedName><![CDATA[[[notificationHandlers.MyNotificationHandler]]]]></localizedName>
     <serviceName><![CDATA[Notify]]></serviceName>
</Row>
```

11. Save the file.
12. In ThingWorx Composer, select **Import/Export ▸ Import ▸ From File**.
13. Click **Choose File**, and navigate to the updated export file.
14. Ensure that **Entities** is selected, and click **Import**.
15. Click **Close**.
16. Open `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]`, and click **Configuration**.
17. For **AdditionalAlertNotificationHandler**, search for and select the thing created in step 7, for example `MyNotificationHandler`. Click **Save**.

Now, when creating an alert, any user can be selected as an alert recipient, not only those users who have their **Notification Preference** configured. For more information, see “Configuring Alerts” in the *ThingWorx Manufacturing Apps Setup and Configuration Guide* or *ThingWorx Service Apps Setup and Configuration Guide*.

13

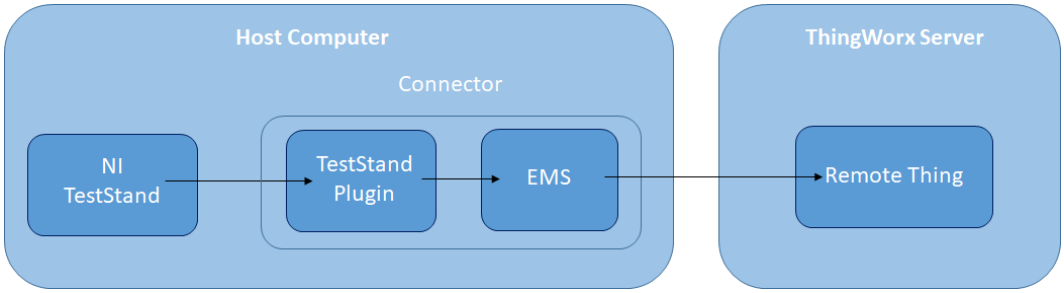
Implementing the NI TestStand Connector

Prerequisites....74
Download the NI TestStand Connector....74
Set the Environment Variable....74
ThingWorx Composer Configurations....75
Setup the Edge Micro Server (EMS) Environment....76
Install the NI TestStand Plugin....76
Adding Steps in a Test Stand Sequence....79

NI TestStand is a tool that allows you to create and run automated validation tests against any type of hardware and drivers. The NI TestStand Connector allows the results from those tests to display in ThingWorx.

The following components are involved in the NI TestStand Connector:

- NI TestStand plugin
- ThingWorx Edge MicroServer (EMS)
- ThingWorx



Prerequisites

Before implementing the NI TestStand Connector, you must have the following installed:

- Visual C++ Redistributable for Visual Studio 2015: https://www.microsoft.com/en-US/download/details.aspx?id=48145
- NI TestStand 2016

Download the NI TestStand Connector

1. Download the "National Instruments TestStand Connector" from the ThingWorx Marketplace: https://marketplace.thingworx.com/.
2. Unzip the downloaded bundle to a known location on your local system. In the following sections, this location is referred to as *<NI_TestStand_Connector>*.

The NI TestStand Connector download bundle contains both the Edge MicroServer (EMS) and the NI TestStand plugin.

Set the Environment Variable

Create an environment variable named `PTC_TWX_TESTSTAND_PLUGIN_PATH`. For the value of this environment variable, specify the full path to the *<NI_TestStand_Connector>*`\TestStandPlugin\config.ini` file.

ThingWorx Composer Configurations

Complete the following steps in ThingWorx Composer.

1. Create a new remote thing with the following settings. This remote thing will receive information from the NI TestStand Connector.
   - **Name**—For example, `TestStandRemoteThing`
   - **Thing Template**—`RemoteThing`
   - Create the following properties with a **Base Type** of **String**:
     - **TestName**
     - **TestResult**
     - **Utilization**

   > **Note**
   > If you have ThingWorx Manufacturing Apps or ThingWorx Service Apps installed, you can select `PTC.SCA.SCO.NITestStandThingTemplate` as the **Thing Template**, which includes these properties

2. Create a new application key from **Security** ▸ **Application Keys**.
   - **Name**—This value is needed when configuring the EMS.
   - **User Name Reference**—Select a user who will use EMS. For example, **Administrator**.
   - **Expiration Date**—Set an appropriate expiration date for the application key, based on your company policies. If left blank, it defaults to one day.
3. If you want to log historical values for your remote thing (`TestStandRemoteThing`), complete the following steps:
   a. Create a value stream from **Data Storage** ▸ **Value Streams**, with the following settings:
      - On the **Choose Template** window, choose the **ValueStream**.
      - **Name**—For example, `ValueStream_TestStand`.
   b. Edit the remote thing created in step 1 to set **General Information** ▸ **Value Stream** to the name of the value stream created in step 3.

Setup the Edge Micro Server (EMS) Environment

The Edge Micro Server (EMS) is available from `<NI_TestStand_Connector>\microserver`.

1. Configure the `config.json` file and run the `wsems.exe` file, following the instructions in the *WebSocket-based Edge MicroServer Developer's Guide*, available at the following location: `<NI_TestStand_Connector>\microserver\doc\ThingWorx_WebSocket_based_Edge_MicroServer_Developers_Guide_v5.4.0.pdf`
2. Update the `<NI_TestStand_Connector>\TestStandPlugin\config.ini` file to match the setting you configured in `config.json`:
   - `protocol`—`http`
   - `host`—The host of your EMS.
   - `thingname`—Name of the remote thing receiving the information from TestStand Connector. This is the remote thing created in ThingWorx Composer Configurations on page 75.
   - `port`—The port for the EMS connection.
   - `username`—If authentication was configured for EMS, specify the user name. Otherwise, leave blank.
   - `password`—If authentication was configured for EMS, specify the password. Otherwise, leave blank.

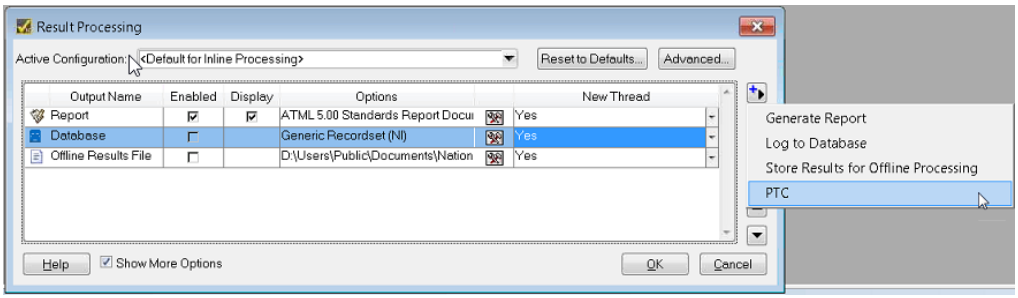
Install the NI TestStand Plugin

The following sections provide information on installing the NI TestStand plugin into NI TestStand. A working knowledge of NI TestStand is assumed.

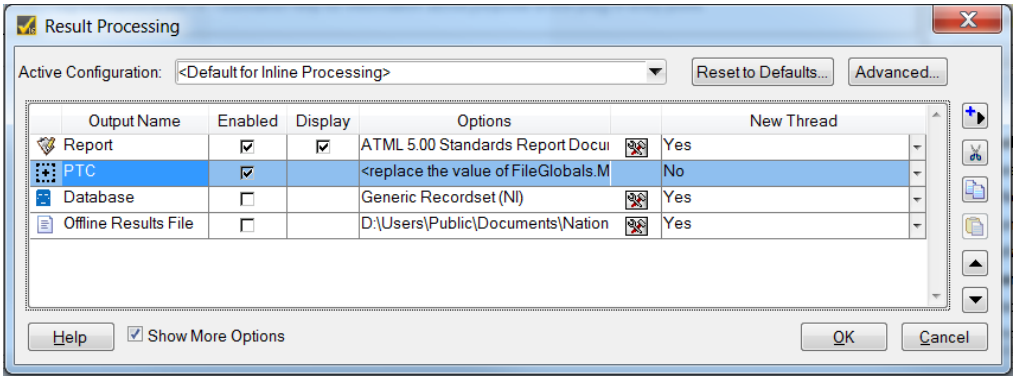
Add PTC as a Result Processing Option

1. Copy the `PTC.seq` file from `<NI_TestStand_Connector>\TestStandPlugin` to `<Public Installation Directory>\Documents\National Instruments\TestStand 2016 (32-bit)\Components\Models\ModelPlugins`. For example: `C:\Users\<username>\Public\Documents\National Instruments\TestStand 2016 (32-bit)\Components\Models\ModelPlugins`.
2. In the NI TestStand Sequence Editor, go to **Configure ▸ Results Processing**.

3. From the insert new list, select **PTC**.



PTC is inserted as a sequence.



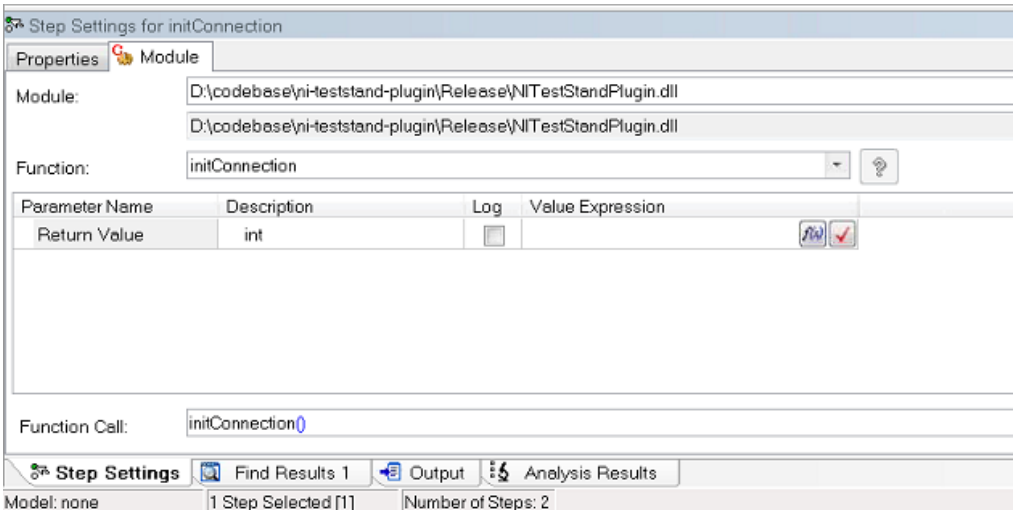
4. Ensure that the **Enabled** checkbox for the PTC sequence is selected, and click **OK**.

Configure the Sequence Model

1. In NI TestStand, select **File** ▸ **Open File**, and navigate to the `PTC.seq` sequence model file.
2. In the following sequences, configure the **Step Settings** so that the **Module** points to *<NI_TestStand_Connector>*`\TestStandPlugin\NITestStandPlugin.dll`. The following table identifies the sequences and steps to be updated, and what each step does.

Sequence	Step	Usage
Model Plugin – Initialize	**initConnection**	Reads the `config.ini` and sets the parameters from the file, such as `host`, `port`, and so on.
Model Plugin – Begin	**GetUtilization**	Sends a PUT query to ThingWorx for the **Utilization** property, and sets it to true.

Sequence	Step	Usage
Model Plugin – UUT Done	**sendThingworx**	Updates the **TestResult** and **TestName** properties in ThingWorx from the current running unit under test (UUT).
Model Plugin – End	**GetUtilization**	Sends a PUT query to ThingWorx for the **Utilization** property, and sets it to false.



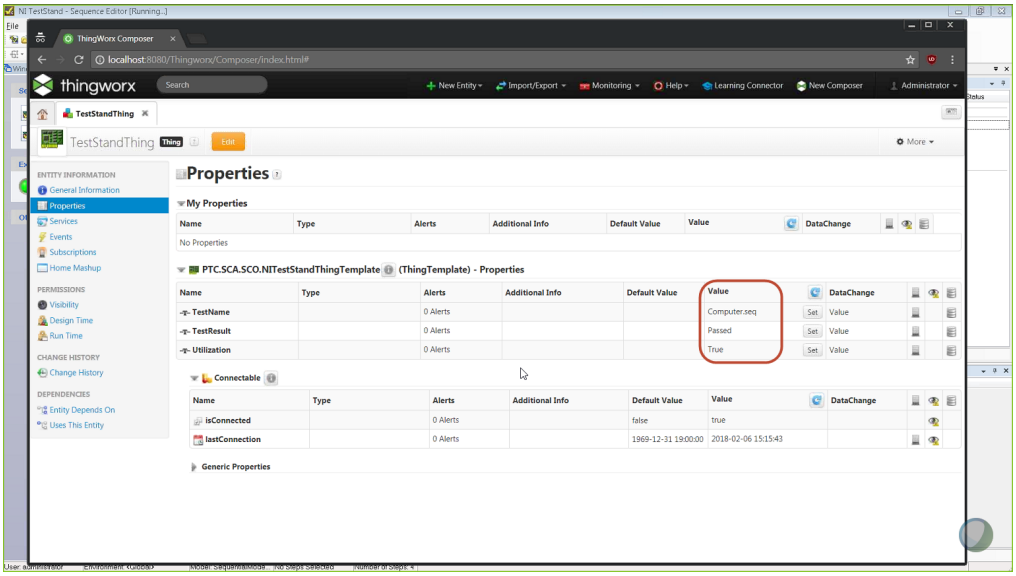
3. Save your updated `PTC.seq`.

Test the NI TestStand Connector

Test your NI TestStand Connector connector configuration.

1. In NI TestStand, open the `Computer.seq` tutorial provided with NI TestStand.
2. Run the test by selecting **Execute** ▸ **Test UUTs**.
3. In the **Enter UUT Serial Number** field, enter a serial number, or leave it blank. Click **OK**.

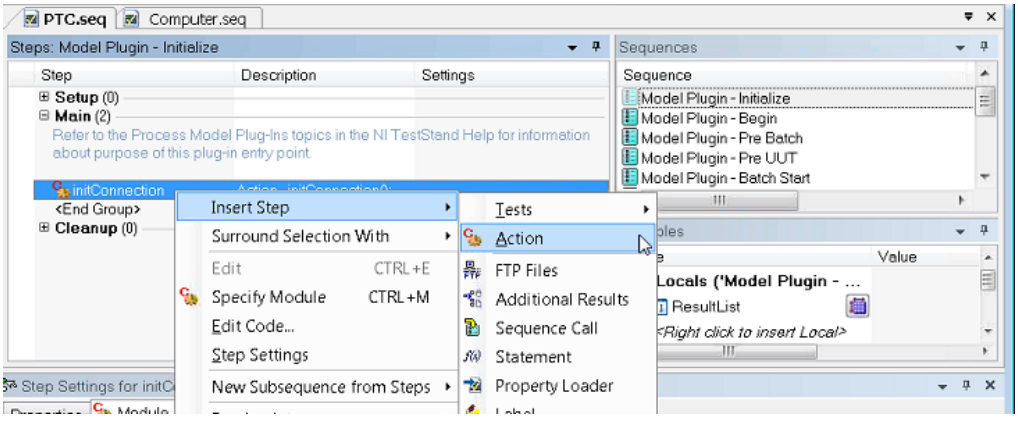
4. On the **Test Simulator** window, select a component to fail, or leave all the checkboxes clear for the test to pass. Click **OK**.
5. When the test completes, view the `TestStandRemoteThing` in ThingWorx Composer. The values for the **TestName**, **TestResult**, and **Utilization** properties display the appropriate values.



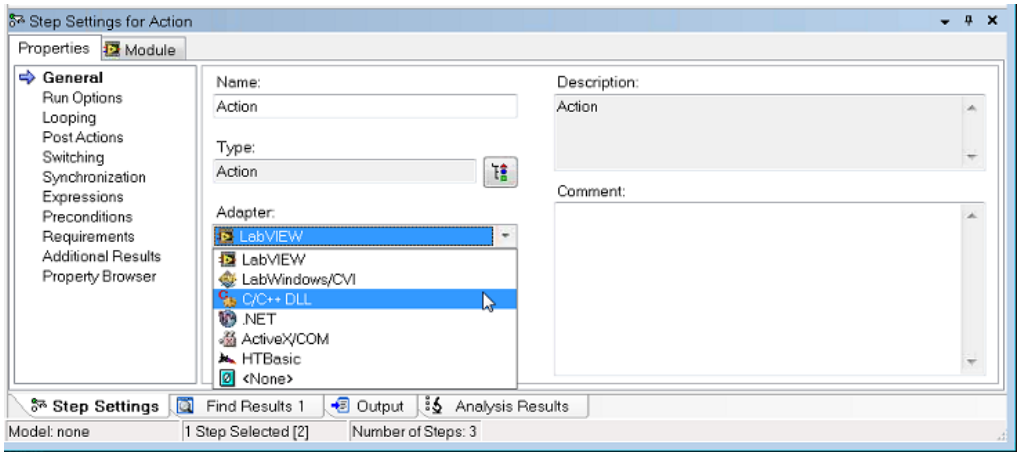
Adding Steps in a Test Stand Sequence

You can add steps in an NI TestStand sequence to send values to ThingWorx, get values from ThingWorx, and execute services in ThingWorx.

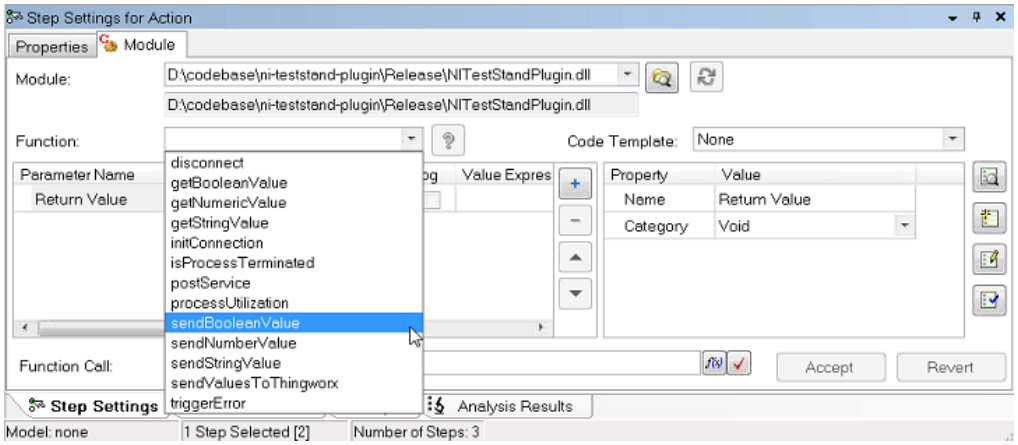
1. In ThingWorx Composer, create any properties on your `TestStandRemoteThing` that you want to use in the sequence steps.
2. In NI TestStand, select the sequence, right-click on the step, and select **Insert Step ▸ Action**.



3. In the **Step Settings for Action** pane, on the **Properties** tab, select **C/C++ DLL** from the **Adapter** list.



4. On the **Module** tab, specify the location of your `NITestStandPlugin.dll` file, as you did in Configure the Sequence Model on page 77.
5. Select the appropriate function from the **Function** drop-down list.



6. Enter the value expression for each parameter as needed.
7. Click **Accept**.
8. Save the updated sequence.

The following sections detail the provided functions, and their required parameters.

Note

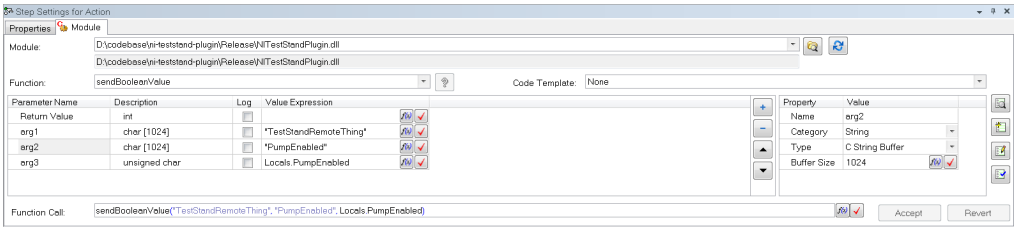
The **Locals** variables are one of multiple ways to store or use values in NI TestStand. Use the method that works for your use case, being sure to match the needed data type.

To send values to ThingWorx

Three functions are provided with the NI TestStand plugin to allow sending property values to ThingWorx.

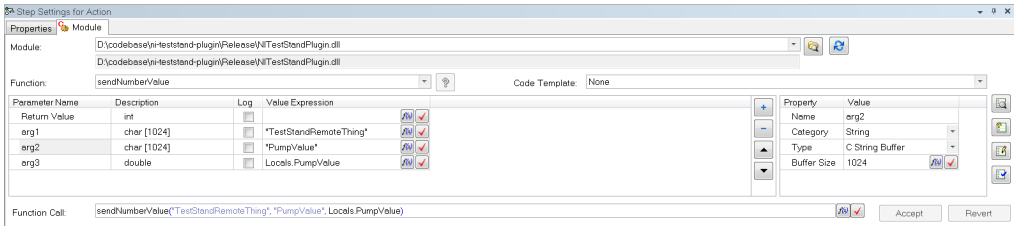
- `sendBooleanValue`—

Parameter	Data Type	Value	Example
`arg1`	String	The name of the thing to which you are sending a property value.	`"TestStandRemoteThing"`
`arg2`	String	The name of the property to which you are sending the value.	`"PumpEnabled"`
`arg3`	Boolean	Boolean value of sent fromNI TestStand.	`Locals.PumpEnabled`



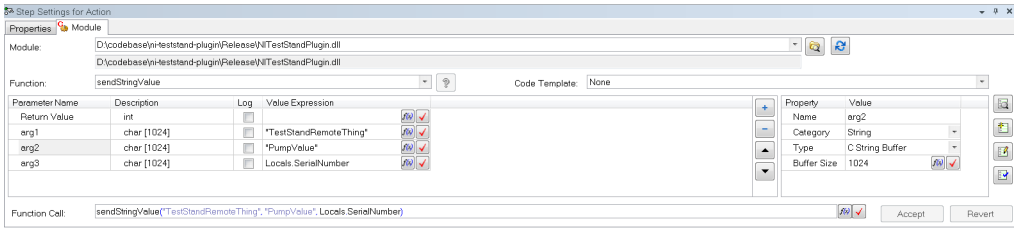
- `sendNumberValue`—

Parameter	Data Type	Value	Example
`arg1`	String	The name of the thing to which you are sending a property value.	`"TestStandThing"`
`arg2`	String	The name of the property to which you are sending the value.	`"PumpValue"`
`arg3`	Double	Numeric value set from NI TestStand.	`Locals.PumpValue`



- `sendStringValue`—

Parameter	Data Type	Value	Example
`arg1`	String	The name of the thing to which you are sending a property value.	`"TestStandRemoteThing"`
`arg2`	String	The name of the property to which you are sending the value.	`"SerialNumber"`
`arg3`	String	String value sent from NI TestStand.	`Locals.SerialNumber`

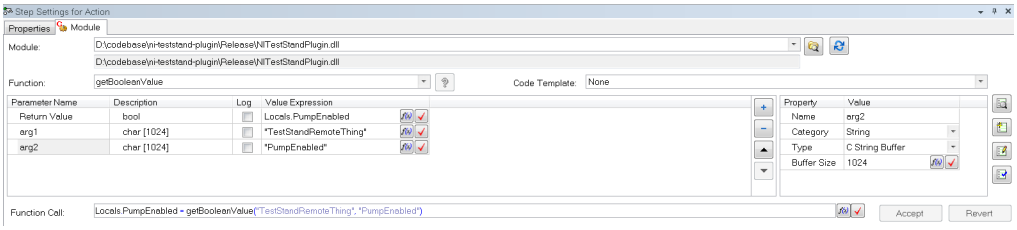


To get values from ThingWorx

The following functions retrieve property values from ThingWorx, and stores them in **Locals** variables in NI TestStand:

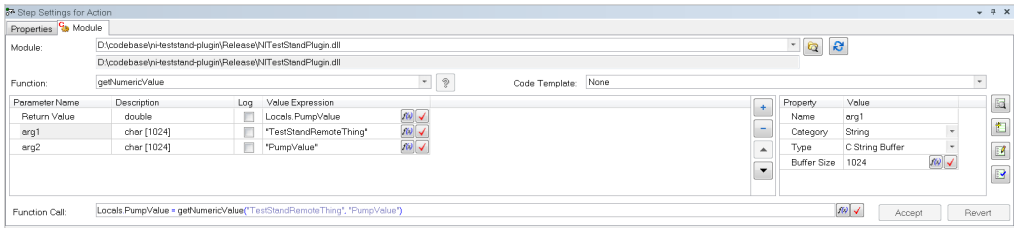
- `getBooleanValue`—

Parameter	Data Type	Value	Example
`Return Value`	Boolean	A **Locals** variable to which to bind the return value.	`Locals.PumpEnabled`
`arg1`	String	The name of the thing from which you are retrieving a property value.	`"TestStandRemoteThing"`
`arg2`	String	The name of the property from which you are retrieving the value.	`"PumpEnabled"`



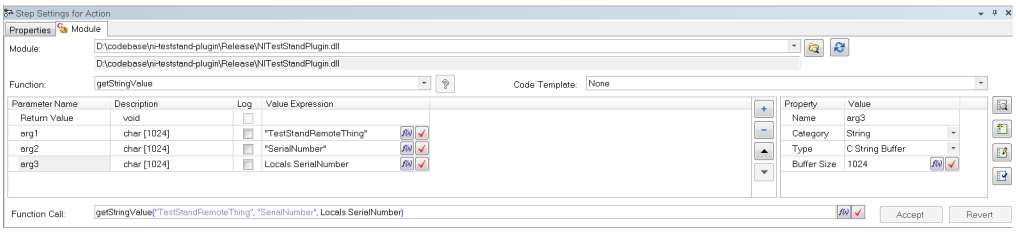
- `getNumericValue`—

Parameter	Data Type	Value	Example
`Return Value`	Double	A **Locals** variable to which to bind the return value.	`Locals.PumpValue`
`arg1`	String	The name of the thing from which you are retrieving a property value.	`"TestStandRemoteThing"`
`arg2`	String	The name of the property from which you are retrieving the value.	`"PumpValue"`



- `getStringValue`—

Parameter	Data Type	Value	Example
`arg1`	String	The name of the thing from which you are retrieving a property value.	`"TestStandRemoteThing"`
`arg2`	String	The name of the property from which you are retrieving the value.	`"SerialNumber"`
`arg3`	String	As strings cannot be returned to NI TestStand, this value is the location where the value is stored.	`"Locals.SerialNumber"`

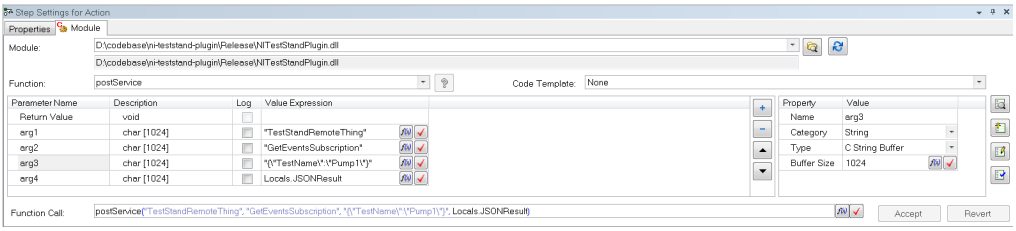


To execute services in ThingWorx

Executing services on specific things in ThingWorx is possible using the `postService` function. This is similar to clicking **Test** on a service in ThingWorx Composer.

- `postService`—

Parameter	Data Type	Value	Example
`arg1`	String	The name of the thing on which you are executing a service.	`"TestStandRemoteThing"`
`arg2`	String	The name of the service to execute.	`"GetEventSubscriptions"`
`arg3`	String	The payload being sent, such as a JSON payload matching the ThingWorx service input parameters, if needed.	`"{\"TestName\":\"Pump1\"}"`
`arg4`	String	The output, such as a local variable in NI TestStand where the result is stored.	`Locals.JSONResult`



14

Customizing Asset Advisor

Customizable Asset Advisor Mashups 88
Anomaly Detection and Asset Advisor 92
Example: Displaying Anomaly Count in Asset Summary Mashup 98

This chapter discusses the Asset Advisor mashups available for customization, as well as the impact of anomaly detection on Asset Advisor. An example customization displaying the anomaly count in the asset summary mashup is also provided.

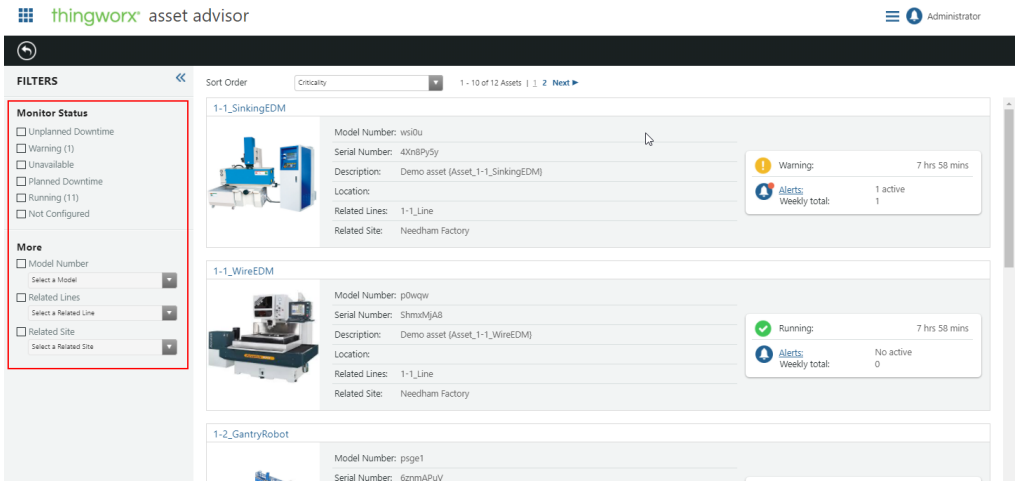
Customizable Asset Advisor Mashups

The following duplicate mashups are available for customizing Asset Advisor:

- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetList.FilterMashup`

  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetList.C_FilterMashup_[ReleaseVersion]`

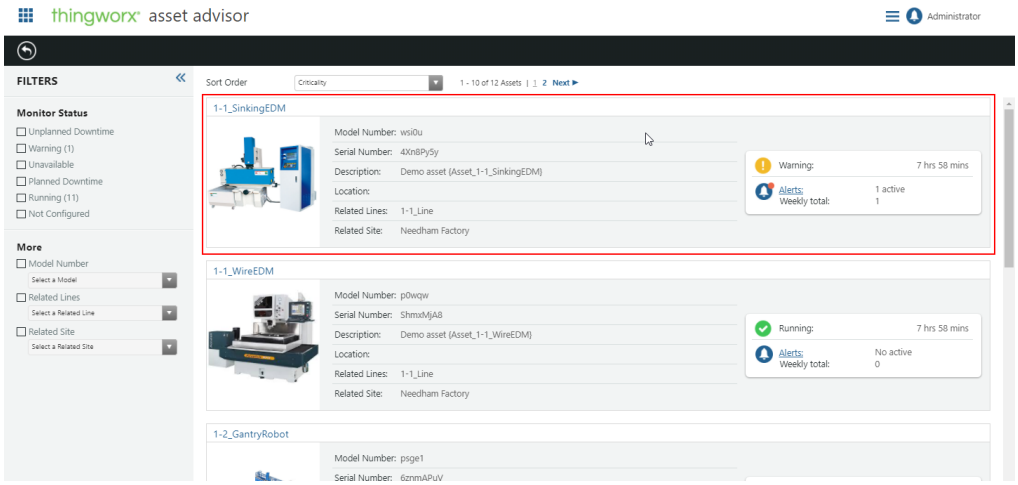
  Image:



- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetSummaryMashup`

  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.C_AssetSummaryMashup_[ReleaseVersion]`

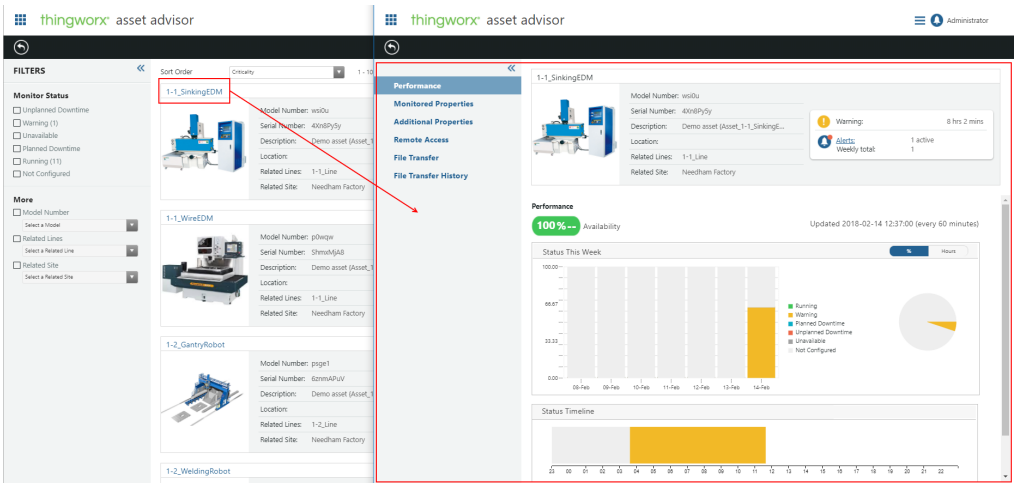
  Image:



- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetailContainerMashup`

  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.C_AssetDetailContainerMashup_[ReleaseVersion]`

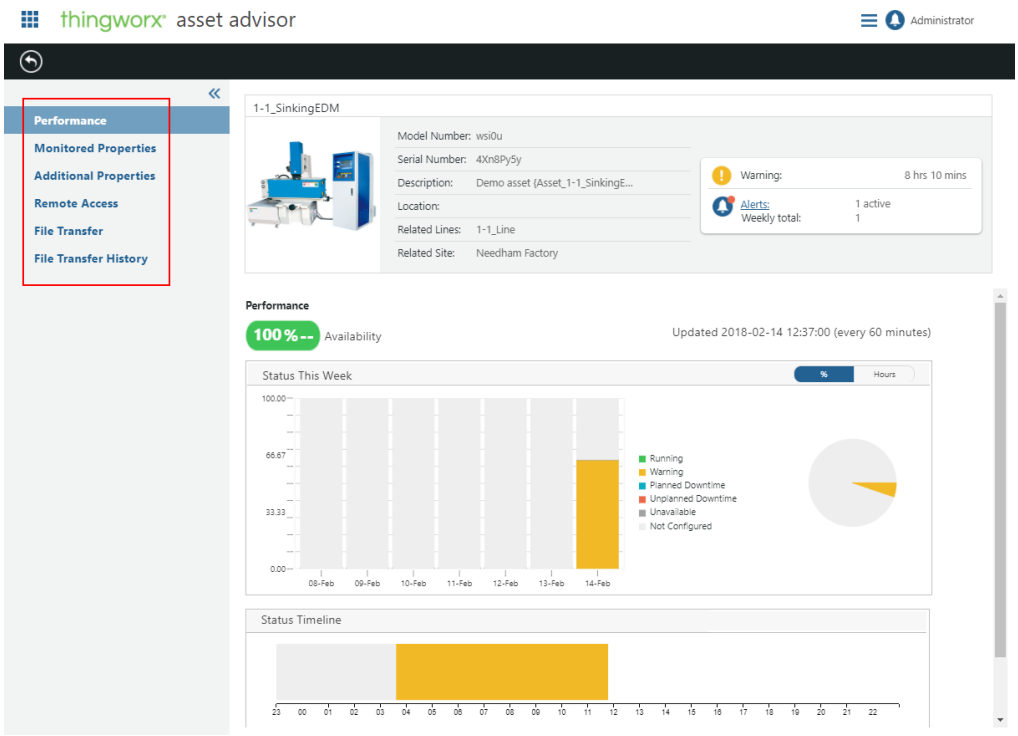
  Image:



- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.ActionMenu`

  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.C_ActionMenu`

  Image:



- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertiesContainerMashup`

  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.C_MonitoredPropertiesContainerMashup_[ReleaseVersion]`

  Image:

- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertiesWithChart`

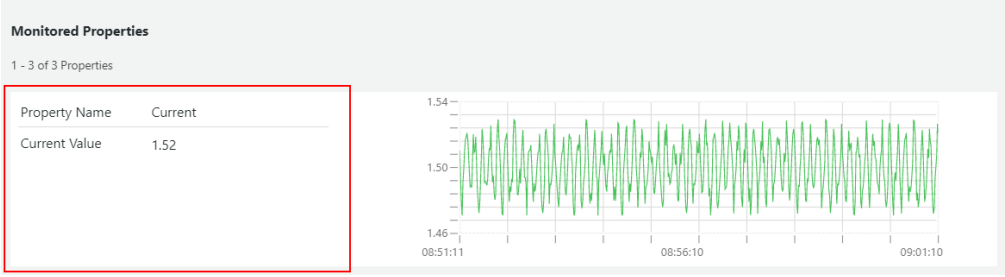
  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.C_MonitoredPropertiesWithChart_[ReleaseVersion]`

  Image:

- Default Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertyWithChart`

  Duplicate Mashup: `PTC.SCA.SCO.AssetMonitor.AssetDetail.C_MonitoredPropertyWithChart_[ReleaseVersion]`

  Image:



Anomaly Detection and Asset Advisor

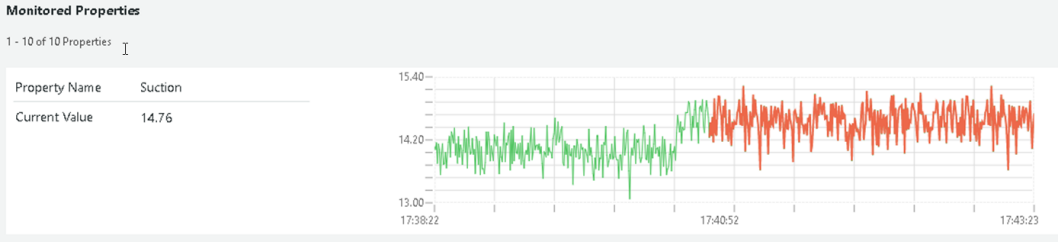
Anomaly detection can be enabled and configured through the ThingWorx platform. For more information, see "Anomaly Detection" under "ThingWorx Model Definition and Composer" in the ThingWorx Help Center, available at the following URL: https://support.ptc.com/help/thingworx_hc/thingworx_8_hc/.

After configuring anomaly detection in the platform, enable the anomaly status update frequency in Asset Advisor. Asset Advisor calculates and records the anomaly status at regular intervals. The update frequency defaults to 30 seconds. It can be made shorter (to improve responsiveness to sensor status changes) or longer (to improve performance or reduce system requirements).

To enable the anomaly status update frequency:

1. In ThingWorx Composer, open the `PTC.SCA.SCO.AnomalyStatusEvaluationScheduler` thing for editing.
2. Click on the **Configuration** tab.
3. Select the **enabled** checkbox.
4. In the **schedule** field, set a new update rate or accept the default value.
5. Click **Save**.
6. Click on the **Services** tab.
7. Click **Test** next to **EnableScheduler**, and click **Execute Service**.

When anomaly detection is enabled, anomalous data displays in orange on the **Monitored Properties** page in Asset Advisor.



The following sections provide additional information on understanding and configuring anomaly detection.

Anomaly Detection Configuration Guidelines

This section provides general guidelines for configuring anomaly detection. There are three main configuration parameters: training time, certainty, and outbound anomaly rate. Training time is used in anomaly detection training to build a baseline. Certainty is used to classify whether an observed data stream is anomalous when compared to the baseline. Outbound anomaly rate smooths brief state changes. These parameters are set on each sensor on an asset.

Minimum Data Collection Time (Training Time)

The first thing to consider in anomaly detection is the amount of data that needs to be collected to produce an accurate model of the system. The system works best on periodic data with cycles of fixed length. Ideally, the system should train on non-interrupted data streams which contain at least five cycles. All training must occur on normal, non-anomalous data.

Once training is complete, anomaly detection looks for anomalies in a time window equal to 20% of the training time. For example, if the training time was ten minutes, when an anomaly occurs, it may continue to be reported for up to two minutes after it subsides. This time window allows relatively subtle anomalous patterns to be detected.

Certainty

The certainty parameter defines a percentage threshold, a value between 50 and 100 (exclusive), used to identify whether a new sensor reading should be considered anomalous based on the comparison between the prediction from the baseline model and actual observations. Very high certainty values make it less likely to report a false anomaly, while lower certainty values lead to fewer undetected anomalies.

The choice of certainty is based on business impact. If missing an anomaly will cause critical asset failure, then the certainty should not be set too high. On the other hand, if the asset is very durable or the sensor data has much noise, and

frequent anomaly alerts cause too much disruption, set the certainty to a high value to reduce false anomalies. Customers need to adjust the certainty for each sensor until false anomalies are at a manageable level.

Because certainty is defined as a statistical threshold, choosing 99.9999 may produce noticeably fewer false positives than 99.9 (for example), despite the small absolute difference in those values.

Outbound Anomaly Rate

This parameter represents the duration over which to smooth anomaly detection results as well as the interval at which to test for anomalies. An anomaly alert is triggered only while the underlying machine learning algorithm has reported anomalies for more than 50% of data points during the most recent interval. That is, if the parameter is set to 1 minute, then every minute, ThingWorx evaluates whether more than 30 seconds of the previous minute's data was anomalous, and sets the alert status accordingly. Increasing this value is the best way to avoid "churn" where brief anomalies appear and disappear.

Outbound anomaly rate must be at least as high as the data scan rate, but should typically be higher to reduce spurious alert activity. The disadvantages of a high value are that the longer test interval can delay reporting an anomaly state change, and that anomalies shorter than the outbound anomaly rate may not be reported at all.

Anomaly Detection Limitations

It is not recommended to apply anomaly detection for data streams with any of the following conditions.

- Sensors with multiple normal states. For example, an HVAC unit has a number of different states of operation during the course of a day. All of these states are "normal", but present very different behavior. For instance, the unit drains more power when actively trying to lower the temperature of the building.
- Sensors with chaotic, unpredictable patterns, such as temperature sensors.
- Sensors with periodic patterns, too fast to be recorded by KEPServerEX, making them appear chaotic.

Understanding Anomaly Detection and Troubleshooting

Anomaly detection in Asset Advisor does not correspond exactly to the judgment of a human eye, and may seem inconsistent at times. Here are some guidelines for interpreting and improving its results.

Expect some false positives

The anomaly detection system works by modeling sensor data and comparing the model's results on recent data against a validation set created during calibration. The system estimates the likelihood that the data sets came from different distributions. If this likelihood exceeds the certainty parameter, the recent data is considered anomalous. For more information on the certainty parameter, see Anomaly Detection Configuration Guidelines on page 93.

Because of the statistical nature of this methodology, sometimes a sensor can be declared as anomalous without the sensor being physically in an anomalous state, especially if certainty or outbound anomaly rate is low or the data is noisy. This can happen even for simulated example data. As a result, brief anomaly reports that are not repeated should not be a major concern for most users. There are some ways to reduce false positive reports, but eliminating them entirely is not always feasible.

Additionally, anomaly detection analyzes a sliding window of data for anomalies, with a length equal to 20% of the training time. A short anomaly (true or false) can continue to be reported for this length of time, even after the sensor data has returned to normal.

If too many false positives, try re-calibrating

False positives are often due to lasting, but harmless, changes in sensor data, such as those caused by environmental changes occurring after the system was trained. These can be fixed by re-calibrating, which is usually the easiest and best thing to try first.

Reducing brief false positives

In many applications, brief anomalies tend to represent statistical noise rather than real problems that require attention. Set the outbound anomaly rate to at least twice the duration of the longest such false positives.

Reducing false positives in un-patterned data

If a sensor's normal variation from moment to moment is mostly noise with no repeated pattern over time, the false positive anomaly reports can be especially common. We recommend setting the certainty parameter to the maximum possible

value in this case. This reduces sensitivity to true anomalies, but true anomalies on this kind of sensor usually result in very low or very high values, so the algorithm does not need to be particularly sensitive.

The training time should be long enough to guarantee that a full range of normal data values can be observed several times each. Additional training time is unlikely to help.

If there are still an unacceptable number of false positives at maximum certainty, consider configuring an alert when data falls outside an expected range, rather than using anomaly detection for that sensor.

Reducing false positives in cyclic data

Many types of sensors tend to generate a repeating pattern of data over time. While the anomaly detection in Asset Advisor excels at detecting subtle anomalies in such data, careful configuration is sometimes necessary if too many false positives are observed.

If sequences of false positives seem to appear and disappear periodically, we recommend changing the training time. Examine the data to see how long the data pattern takes to repeat itself. If the pattern is short (less than two minutes), then we recommend training for at least 20 times this cycle length. An even longer training time can help, especially if the data also seems noisy. For longer patterns, instead of increasing the training time, measure the cycle length more precisely and set the training time to an exact multiple of 5 times the cycle length.

After completing this training time, we recommend increasing the certainty until false positives reach an acceptable level.

Missed anomalies

A false negative occurs when sensor data appears anomalous, but Asset Advisor reports that it is normal. This usually means that the certainty has been set too high and should be reduced. (A small number of false negative data points within a correct anomalous report can be safely ignored.)

If Asset Advisor is taking too long to report a real anomaly, or if the missed anomalies are short, try reducing the outbound anomaly rate. Asset Advisor also may not react to anomalies of just a few data points. If true anomalies on a sensor are expected to be this brief, then configuring ThingWorx range alerts for the sensor data may be a useful supplement.

If brief false negative intervals appear within a correct anomalous report, increase the outbound anomaly rate to at least twice the duration of these intervals.

System limitations

Some kinds of data cannot be effectively handled by anomaly detection in Asset Advisor. Avoid configuring anomaly detection for systems with more than one normal state (such as a belt with multiple speeds), or for sensors whose values may have chaotic, non-repeating patterns, or ranges of values that are not seen during a training period (such as temperature).

The system is not very sensitive to changes in cycle period or frequency. If an anomaly manifests as a pattern with normal amplitude but with a faster or slower cycle, the Asset Advisor anomaly detection may not react to changes in the individual intervals between these patterns, although it may detect a change if the activity becomes much more or less frequent overall.

Also note that KEPServerEX generally does not handle incoming data faster than 20 Hz, which causes data patterns with a higher frequency (such as alternating current) to look un-patterned.

Example: Displaying Anomaly Count in Asset Summary Mashup

In this example, we will customize the asset summary mashup in Asset Advisor to display the current total number of anomalies on the asset.

Prerequisites:

1. Anomaly detection is enabled and configured in ThingWorx.
2. The **PTC.SCA.SCO.AnomalyStatusEvaluationScheduler** has been enabled.
3. At least one asset is configured with at least one tag-based numeric property.
4. At least one anomaly alert is configured.

Complete the following steps in ThingWorx New Composer:

Note

To access ThingWorx New Composer:

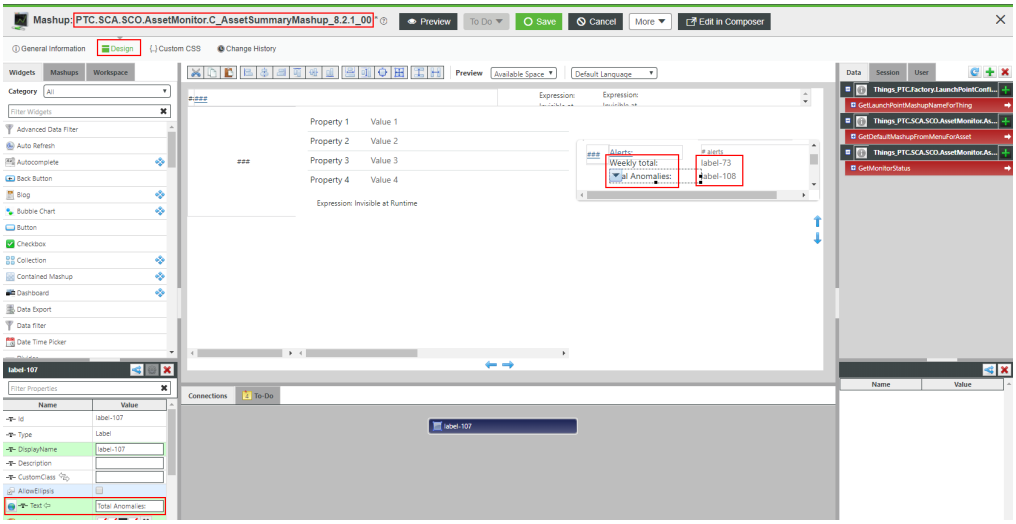
1. In the toolbar, select **Administrator** ▸ **Preferences**.
2. Select **Turn on New Composer Features**.
3. Click **Done**.
4. In the toolbar, click the **New Composer** link.

1. Open `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion]` for editing.
2. Under **Configuration**, set the **AssetListEntryMashup** value to `PTC.SCA.SCO.AssetMonitor.C_AssetSummaryMashup_[ReleaseVersion]`, which is the mashup to be customized, and click **Save**.
3. Open the `PTC.SCA.SCO.AssetMonitor.C_AssetSummaryMashup_[ReleaseVersion]` mashup for editing.

4. Under **Custom CSS**, add custom CSS similar to the following to increase the size of the asset summary to accommodate the custom text:

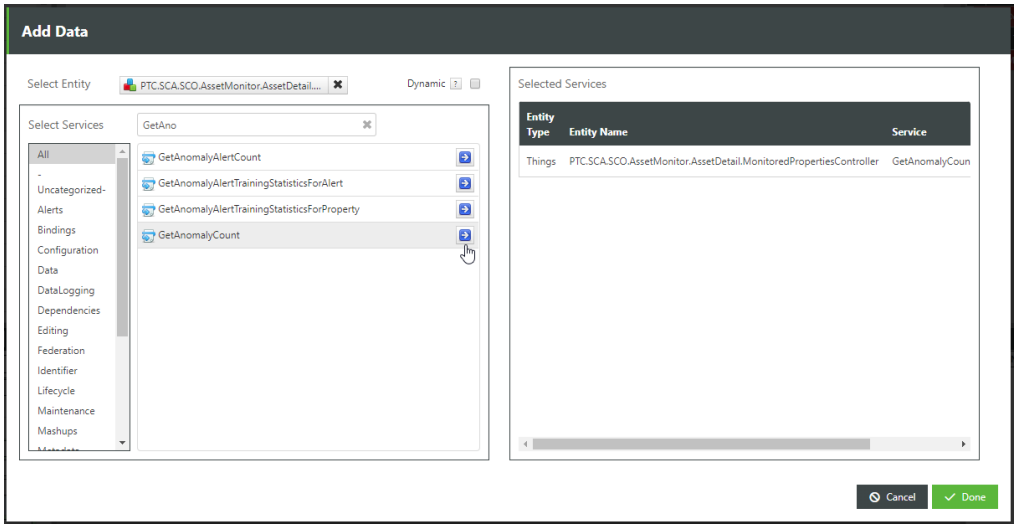
```
.asset-identity .status-panel .status-box {
height: 10.3125rem !important;
    margin-top: 0.3rem;
}
```

5. Click **Save**.
6. Under **Design**, copy and paste the **Weekly total** label and change the text of the duplicate to **Total Anomalies**. Also duplicate the value label to the right as shown.

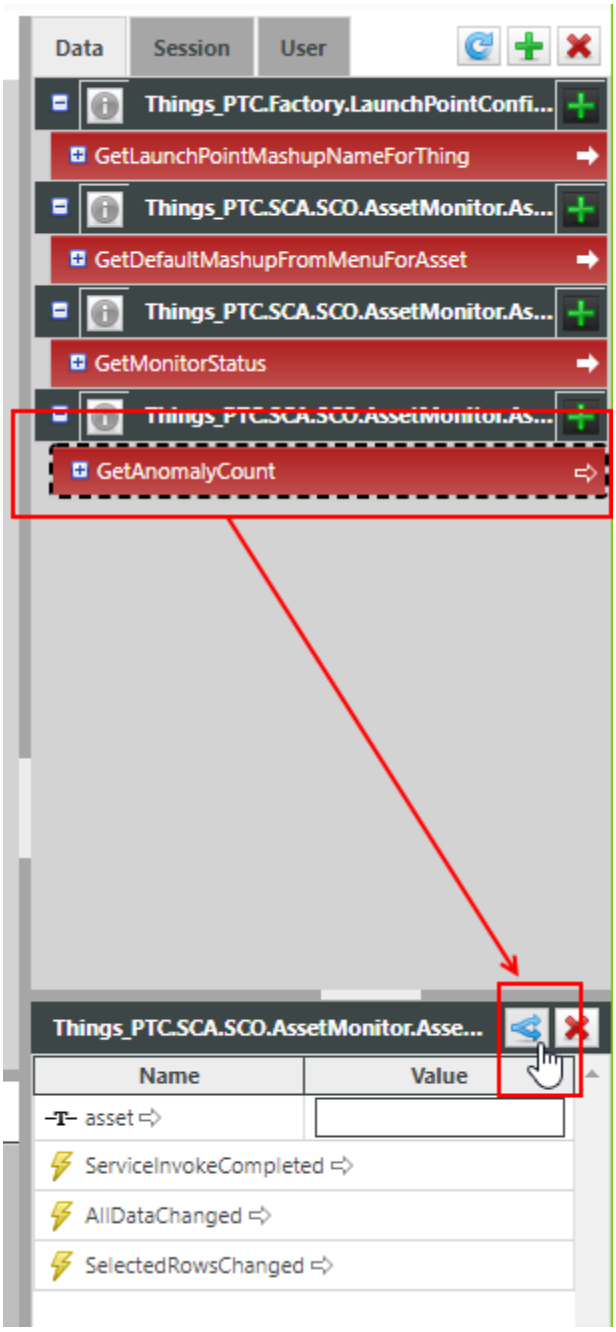


7. Click  on the **Data** tab on the top right to open the **Add Data** window.

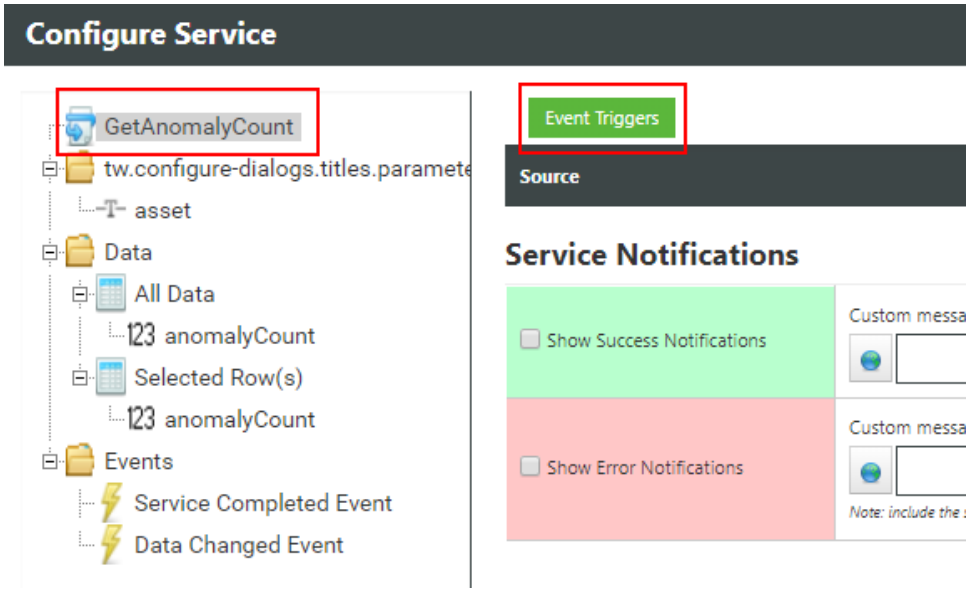
8. Search for `PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertiesController`, and add the `GetAnomalyCount` service. Click **Done**.



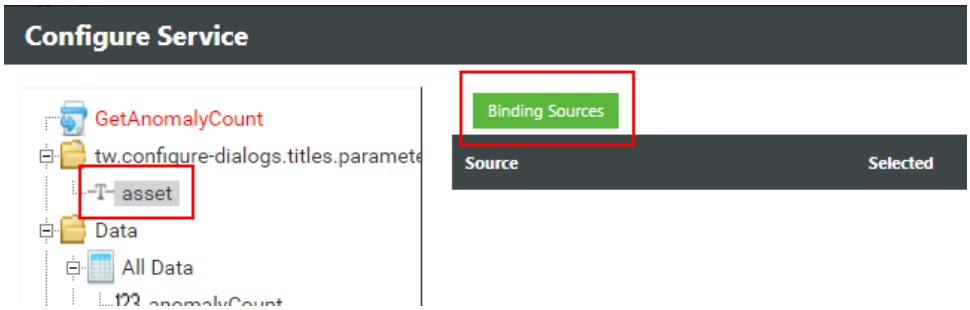
9. In the **Data** tab on the right pane, select **GetAnomalyCount**, then click  to launch the **Configure Service** window.



10. Define the event triggers.
    a. On the **Configure Service** window, with **GetAnomalyCount** selected, click **Event Triggers**.

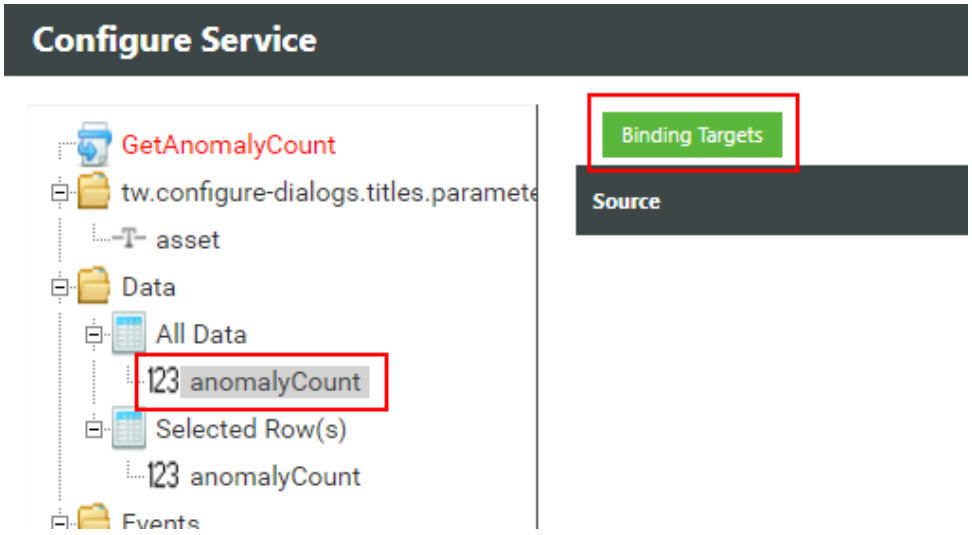


    b. On the **Add Data Binding** window, in the **WIDGETS** pane, under **Mashup**, select the following event triggers:
        - **RefreshRequested**
        - **AssetInfoChanged**
        - **AnomalyStatusAutoRefresh ▸ Refresh**
    c. Click **Done**.
11. Define the input.
    a. On the **Configure Service** window, select **asset**, and click **Binding Sources**.

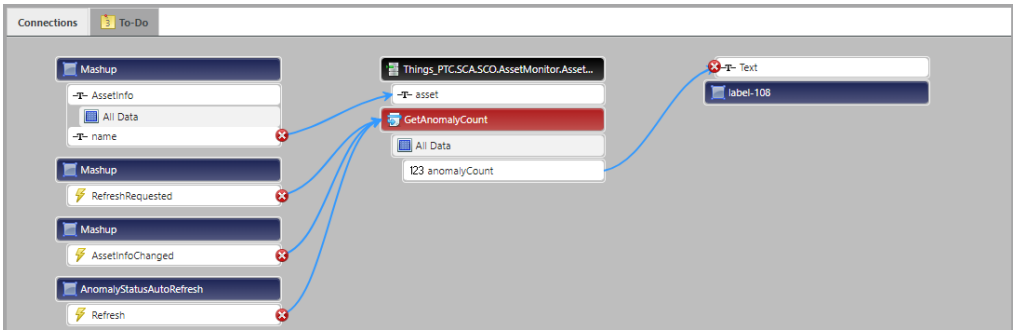


    b. On the **Add Data Binding** window, in the **WIDGETS** pane, under **Mashup ▸ AssetInfo ▸ All Data**, select **name**.
    c. Click **Done**.

12. Define the output location.
    a. On the **Configure Service** window, under **Data** ▸ **All Data**, select **anomalyCount** and click **Binding Targets**.

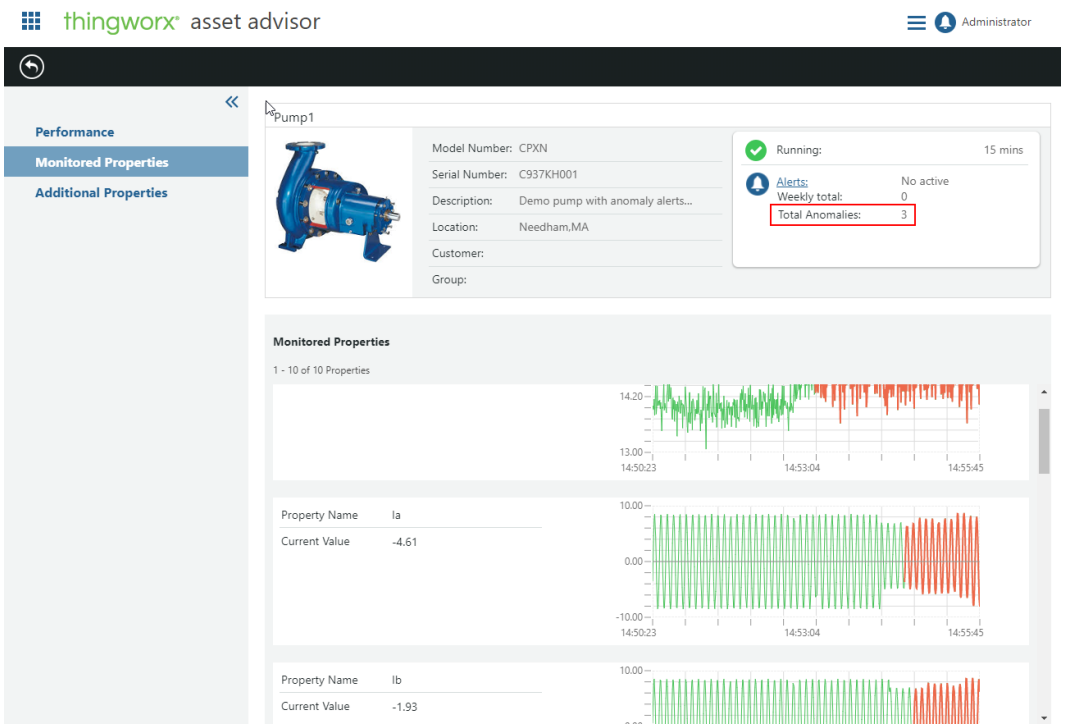


    b. On the **Add Data Binding** window, scroll down to the value label created in step 6(in this example, **label-108**), and select **Text**.
    c. Click **Done**.
13. Click **Done** to close the **Configure Service** window. The configured event triggers, input, and output display in the **Connections** pane.



14. Click **Save** to save the mashup.

The asset status panel in **Asset Advisor** now shows the current total of anomalies for the asset:



15

Deprecated Entities, Services, and Properties

The following entities, services, and properties are deprecated as of the 8.2 release. They will be removed from ThingWorx Manufacturing and Service Apps in release 9.

Deprecated Entities

- `PTC.ISA95.ExtendedPhysicalAssetThingShape`
- `PTC.ISA95.PropertyManagerThingShape`
- `PTC.ISA95.ExtendedEquipmentThingShape`
- `PTC.Factory.AlertMonitor.SelectTag`
- `PTC.ISA95.ProductionLineThingShape`
- `PTC.ISA95.SiteThingShape`
- `PTC.Factory.ManagePlantNetwork`
- `PTC.Factory.AddAssetToPlantNetwork`
- `PTC.Factory.OPCTagProperties`

Deprecated Java-based Entities

- `PTC.Factory.AssetPerformanceUtils`

Deprecated Services

- On the `PTC.SCA.SCO.ManageResourceUtils` entity:
  - `RemoveExtraProperty`
  - `GetPropertyForUI`
  - `SaveExtraProperty`

    - CreateInfoTableForClearHistoryPhysicalAsset
    - AddSingleAssetToLines
    - CreateAsset
    - CreateLine
    - CreateSite
    - GetListForDropDownMenu
    - GetManufacturingElements
    - SiteWithNameAlreadyExists
    - UpdateRelatedLinesAndSites
    - UpdateRelatedLinesAndSitesForAllAssetsAndLines
- On the PTC.Factory.PlantStatusUtils entity:
    - AddMultipleAssetsToLine
    - CheckPhysicalAssetInputField
    - CheckShiftInputFields
    - CreatePhysicalAsset
    - CreateResource
    - FindTreeRow
    - Get_AllPhysicalAssetsFromPlant
    - Get_AllPhysicalAssetsUnderLine
    - GetElementInfoForUsageReporting
    - GetExtraPhysicalAssetPropertiesAndValuesForBinding
    - GetGeneralInfoTooltip
    - GetPhysicalAssets
    - GetPhysicalAssetInfo
    - GetSiteInfoForUsageReporting
    - GetSiteDropDownList
    - GetRelatedManufacturingElements
    - GetUnrelatedManufacturingElements
    - Get_UnassignedAssetsFromPlant
    - RemoveLinesFromPlant
    - RemoveMultipleAssetsFromLine
    - RemoveSingleAssetFromLines
- On the PTC.Factory.KepServerResourceProviderThingTemplate entity:
    - getSubscribedTagData

- On the `PTC.Factory.Administration.TagConfigurationUtils.java` entity:
  - `getSubscribedTagData`
  - `browserTypeAndSourceFilteredItems`
  - `browseGroups`
  - `CloneOPCTagPropertiesInfoTable`
- On the `PTC.Factory.CommonTagPickerUtilities` entity:
  - `GetAllBindSource`

Deprecated Properties

- `displayId` in the `PTC.ISA95.IdentifierThingShape` entity.
- `LastCurrentServerTimeValueUpdate` in the `PTC.Factory.KepServerThingShape` entity.
  - This value is calculated by the `KepServerResourceProvider` entity.
  - This value can also be calculated by calling the `GetPropertyTime` service on the `CurrentServerTime` property.

Deprecated in Previous Release

The following service was deprecated in a previous release. It will be removed from ThingWorx Manufacturing and Service Apps in release 9.

- On the `PTC.Factory.KepServerThingShape` entity:
  - `GetServerStatus`