Contents

About This Guide .................................................................................................................. 6

Customizing the ThingWorx Apps ....................................................................................... 8
  Using Duplicate Files for Customization ................................................................. 9
  Comparing Your Customizations with the Default .............................................. 10
  Viewing User Interface Components In Use ...................................................... 11
  Using Tags and Descriptions to Find Entities ...................................................... 12
  Upgrade and Customizations ...................................................................................... 15
  Standard License Limitations ....................................................................................... 16

Customizing Information Content .................................................................................... 17

  Creating Custom Equipment Types and Structures ........................................ 18
  Defining Equipment Types ....................................................................................... 19
  Defining Equipment Relationships ........................................................................ 21
  Creating Custom Thing Templates for Equipment Types ................................ 22
  Creating a Configuration Menu for an Equipment Type .................................... 24
  Defining Property Display Templates for Equipment Types .......................... 25
  Configuring the Thing Name Format For Your Equipment Type .................. 27
  Disabling vs. Deleting an Equipment Type ............................................................ 28
  Defining Mashups and Menus for Equipment Types ............................................ 28

Customizing Equipment Status ....................................................................................... 30

  Customizing the Default Status Expressions for Custom Equipment Templates .................................................. 31
  Customizing Equipment Status Evaluation ............................................................ 32
  Detecting Bad Quality Data in Custom Code ......................................................... 33

Customizing Key Performance Indicator (KPI) Evaluation ................................ 36

  Customizing the KPI Evaluation Services .............................................................. 37
  Customizing the Help Text for the Performance Metrics Page ....................... 39

Excluding Property Base Types when Exporting Equipment ........................ 42

Customizing Equipment Import Services ................................................................... 43
  Overridable Import Services ................................................................................... 44
  Service Customization Examples ........................................................................... 45

Adding Properties and Fields on Assets, Lines, and Sites ................................ 59

Creating Custom Roles ................................................................................................. 62

Adding a Custom Notification Handler ....................................................................... 64

Customizing the Schedulers ......................................................................................... 67
  Editable Schedulers ................................................................................................... 68
To configure the launch schedule for a scheduler: ............................................. 68
To enable or disable the scheduler: ................................................................. 69
To set the age of the data to keep after each purge: .................................... 69

Implementing the NI TestStand Connector ..................................................... 71
Prerequisites ................................................................................................. 72
Download the NI TestStand Connector ......................................................... 72
Set the Environment Variable ....................................................................... 72

ThingWorx Composer Configurations ............................................................. 73
Setup the Edge Micro Server (EMS) Environment ........................................ 74
Install the NI TestStand Plugin ..................................................................... 74
Adding Steps in a Test Stand Sequence ......................................................... 78

Displaying Azure IoT Hub Data as Property Values on an Asset ................... 86
Prerequisites ................................................................................................. 87
Displaying Azure IoT Hub Data in ThingWorx Apps .................................... 87

Customizing Presentation .......................................................................... 91

Using the Launch Point Configuration Thing ................................................. 92
Example Customization Using the Launch Point Configuration Thing ............ 93
Changing the Tiles in the Main Application Console ..................................... 96
Changing the Logo and Text on the Welcome Page ...................................... 104
Changing or Adding New Logos in the Application Headers ........................ 106
Changing the TagCountAboveLimit Alert Default ...................................... 107
Other Customizable Launch Point Mashups ................................................... 107

Changing Labels in the Application ............................................................. 113
Changing the Logos in the Application Console ......................................... 116
To change the PTC logos in the application console: ................................... 117
To keep the PTC logos and add an additional logo in the master console: ..... 118

Customizing the Duration Display Format .................................................... 119

Customizing the Tag Picker Common Component ...................................... 121
Disable Preservation of the Last Selection .................................................. 122
Browse Data from Custom Connectors ....................................................... 122
Using the Tag Picker Common Component in a Mashup ............................. 124

Customizing Tab Pages in the Configuration and Setup Tile ....................... 126
Changing the Main Mashup ......................................................................... 127
Adding Tab Pages in the Configuration and Setup Main Mashup .................. 127
Granting Access Control to the Tab Page .................................................... 129
Modify the Existing Tab Pages .................................................................... 130

Customizing Controls Advisor .................................................................... 134
Customizing the Calculation of Remote Server Status ................................ 135
Including New Server Types or Custom Connectors .................................. 137

Customizing Asset Advisor ......................................................................... 142
Customizable Asset Advisor Mashups ......................................................... 143
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anomaly Detection and Asset Advisor</td>
<td>147</td>
</tr>
<tr>
<td>Example: Displaying Anomaly Count in Asset Summary Mashup</td>
<td>152</td>
</tr>
<tr>
<td>Dygraph Widget</td>
<td>159</td>
</tr>
<tr>
<td>Appendix A. Deprecated Entities, Services, and Properties</td>
<td>167</td>
</tr>
</tbody>
</table>
About This Guide

This guide describes how to customize ThingWorx Apps in ThingWorx Composer.

This guide assumes that prerequisite products are installed and configured. For more information, see the Product Requirements section in the ThingWorx Apps Setup and Configuration Guide located at the PTC Reference Documents website.

Note

Some commands and code examples in this guide have been reformatted for presentation purposes and therefore can contain hidden editing characters such as tabs, end-of-line characters, and extraneous spaces. If you cut and paste code from this guide, check for these characters and remove them before attempting to use the code in your application.

Related Documentation

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

- ThingWorx Apps Setup and Configuration Guide
- What’s New in ThingWorx 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 Apps Customization Guide* book title as well.
Customizing the ThingWorx Apps

As starter apps, the ThingWorx 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 equipment types and equipment relationships.
- Defining custom user roles.
- 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 from the **ThingWorx Composer** tile on the ThingWorx Apps console, or at the following URL:

Using Duplicate Files for Customization

Certain entities within the ThingWorx 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_8.4.0`. 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 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 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 tags:
    - PTC:Icon
    - PTC:Image
  - PTC:Menu
  - PTC:ModelTagVocabulary
  - PTC:Network
  - PTC:Organization
  - PTC:Project
  - PTC:Resource
• 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 with assets:

1. Click **Advanced** next to the filter field.
2. Click **▼**, and select the **PTC:Asset** 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 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:


2. Modify the data table of PTC.FactoryConsole.C_DataTable_[NewReleaseVersion] with all changes that were made in the PTC.FactoryConsole.C_DataTable_[OldReleaseVersion].

Standard License Limitations

Some functionality is limited or unavailable with a Standard license:

- Existing mashups can be edited, but cannot be duplicated. New mashups cannot be created. The following customizations involve the creation of new mashups, so are not applicable:
  - Adding Tab Pages in the Configuration and Setup Main Mashup on page 127
  - Implementing Custom Mashups for your New Server or Custom Connector on page 141

- The Production KPIs tile in the ThingWorx Apps console and the related Performance Metrics page for configuring individual pieces of equipment in Configuration and Setup > Equipment are not available. As a result, the following customization is not applicable: Customizing the KPI Evaluation Services on page 37.

- The Remote Access and Control feature documented in the ThingWorx Apps Setup and Configuration Guide is not available.

- Customizations to ThingWorx Apps performed on another system cannot be deployed into a system with a Standard license, either as an extension or through import of individual entities.
Customizing Information Content

The chapters in this section provide information on customizing the information that you are including in ThingWorx Apps.
Creating Custom Equipment Types and Structures

Defining Equipment Types ................................................................. 19
Defining Equipment Relationships ......................................................... 21
Creating Custom Thing Templates for Equipment Types .......................... 22
Creating a Configuration Menu for an Equipment Type ............................ 24
Defining Property Display for Equipment Types ....................................... 25
Configuring the Thing Name Format For Your Equipment Type ................ 27
Disabling vs. Deleting an Equipment Type ............................................ 28
Defining Mashups and Menus for Equipment Types .................................. 28

ThingWorx Apps comes with pre-defined equipment types and equipment relationships for a defined equipment structure. You can extend the pre-defined equipment types, create your own equipment types, and establish the relationships between equipment types to create an equipment structure which suits your needs. This is achieved by:

• Defining the equipment types.
• Defining the valid relationships between equipment types.

These definitions are performed on the Configuration page of the launch point configuration thing (PTC.Factory.C_LaunchPointconfigurationThing_[ReleaseVersion]).
Defining Equipment Types

Equipment types are defined in the EquipmentTypeSettings configuration table. Initially, this table includes the three pre-defined Site, Line, and Asset types. These types can be deleted or edited.

<table>
<thead>
<tr>
<th>Equipment Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site</td>
<td>A pre-defined equipment type for sites.</td>
</tr>
<tr>
<td>Line</td>
<td>A pre-defined equipment type for lines.</td>
</tr>
<tr>
<td>Asset</td>
<td>A pre-defined equipment type for assets.</td>
</tr>
</tbody>
</table>

Note

Do not edit the EquipmentType value for the pre-defined Site, Line, and Asset equipment types.

To define a new equipment type:

1. Click Add on the EquipmentTypeSettings configuration table.
2. In the EquipmentTypeSettings window, specify the following information:
   - **EquipmentType**—The name of the equipment type. This value is used when defining equipment relationships and is displayed by default as the name for the equipment type in ThingWorx Apps if no LocalizationToken is specified for the type. Must be unique.
   - **DerivesFrom**—The main type that this equipment is derived from. If a DerivesFrom value is specified, this equipment type automatically follows the relationships defined for the equipment type from which it derives. The value entered is case-sensitive, and must match the EquipmentType value of the main equipment type. Optional.
   - **ThingTemplate**—The thing template that is set as the Base Thing Template when creating an instance (thing) of this equipment type in ThingWorx Apps. For more information, see Creating Custom Thing Templates for Equipment Types on page 22.

Note

- A thing template can only be used for a single equipment type.
- Once equipment has been created for an equipment type, do not change the thing template associated with the equipment type in the EquipmentTypeSettings configuration table. You can modify the thing template itself as needed.
• **ResourceProvider**—The resource provider contains the logic used when creating, editing, and deleting instances of this equipment type. The same resource provider can be used for multiple equipment types. For assets, or equipment types that display in the Asset Advisor, we recommend using the `PTC.SCA.SCO.AssetResourceProvider`, as it contains additional logic specific to assets. A resource provider named `PTC.SCA.SCO.EquipmentResourceProvider` is provided for use by any equipment type. You can use these resource providers, or create a new resource provider specifically for your equipment type.

• **LocalizationToken**—The name of the localization token displayed as the name for this equipment type in ThingWorx Apps. If no **LocalizationToken** is specified, the **EquipmentType** value displays by default. For more information, see “Localization Tables” in the ThingWorx Help Center. Optional.

• **ConfigurationMenu**—The name of the configuration menu that opens after creating or when configuring an instance of this equipment type in ThingWorx Apps from the **Equipment** tab of **Configuration and Setup**. A generic configuration menu that includes only the **General Information** and **Equipment Structure** links is provided by default when defining new equipment (`PTC.SCA.SCO.EquipmentConfigurationMenu`). You can use the generic configuration menu, or you can create new custom configuration menus. For more information, see Creating a Configuration Menu for an Equipment Type on page 24.

• **EnableCreation**—When this checkbox is selected (the default), instances of this equipment type can be created in ThingWorx Apps. Equipment types without this checkbox selected are filtered out of the **Type** drop-down list when creating new equipment in ThingWorx Apps.

• **EnableTemplateAlerts**—When this checkbox is selected (the default), alerts can be defined for the thing template of this equipment type on the **Alerts** tab of **Configuration and Setup** in ThingWorx Apps. When this checkbox is not selected, the thing template corresponding to this equipment type is filtered out of the **Template** drop-down list when creating a template alert. For more information on template alerts, see “Configuring Alerts” in the ThingWorx Apps Setup and Configuration Guide.

3. Click **Add** to add the equipment type to the configuration table.

4. Click **Save** to save the launch point configuration thing.
Defining Equipment Relationships

Once an equipment type has been defined in the EquipmentTypeSettings configuration table, you can define the valid parent-child relationships for that equipment type in the EquipmentRelationshipSettings configuration table. These parent-child relationships are used when creating your equipment and equipment structure. Equipment types that derive from another equipment type automatically inherit the relationships defined for the type from which it derives.

In order for equipment of an equipment type to be created in ThingWorx Apps, the equipment type must be part of at least one valid equipment relationship definition, or be derived from an equipment type that is part of a valid equipment relationship definition.

The four equipment relationships defined for the Site, Line, and Asset equipment types shown in the following screenshot are provided with ThingWorx Apps. These relationships can be edited and removed.

To add an equipment relationship definition:

1. Click Add on the EquipmentRelationshipSettings configuration table.
2. Specify the Parent and Child equipment types for the relationship. Both equipment types must already be present in the EquipmentTypeSettings configuration table, and the values entered for Parent and Child fields must match the EquipmentType values for those types in the EquipmentTypeSettings configuration table. These values are case-sensitive.
3. If you want to be able to associate the same child equipment to more than one parent at a time, select the ManyToManyCardinality checkbox. For example, the ManyToManyCardinality checkbox must be selected for the Line-Asset parent-child relationship to associate Asset1 with both Line1 and Line2.
4. Select the Enabled checkbox to enable the relationship in ThingWorx Apps. Only enabled relationships can be created in ThingWorx Apps.
5. Click Add to add the relationship to the configuration table.
6. Click Save to save the launch point configuration thing. The relationships are validated when the launch point configuration thing is saved, not when the relationship is added to the configuration table.
Note

Equipment types that derive from another equipment type follow all the relationships defined for the equipment type from which they derive, unless a valid parent-child relationship is defined explicitly for them. Additional parent-child relationships can be defined explicitly for equipment types that otherwise derive their relationships from another type.

For example, if you define an equipment type named Drill that is derived from the Asset equipment type, then all parent-child relationships defined for the Asset type apply to the Drill type. If the parent-child relationships defined in EquipmentRelationshipSettings configuration table are defined for Site-to-Line, and Line-to-Asset, then Drills follow the Line-to-Asset relationship and can be created as children of Lines. You can define an additional explicit Site-to-Drill parent-child relationship, which would then allow Drills to be created as children of Assets and of Sites, while Assets can still only be created as children of Lines.

Deleting an equipment relationship definition has no impact on existing equipment or on the equipment structure. No further relationships between those particular parent and child equipment types can be created until a new relationship is defined between them.

Creating Custom Thing Templates for Equipment Types

A generic thing template (PTC.ISA95.EquipmentThingTemplate) is provided with ThingWorx Apps. This thing template implements GenericThing as its Base Thing Template, and implements the thing shapes required to support basic ThingWorx Apps functionality as its Implemented Shapes. When an instance (thing) of an equipment type is created, the thing template specified for the equipment type in the EquipmentTypeSettings configuration table is set as the Base Thing Template for the equipment thing. Any properties defined on the thing template used by an equipment type, or on a thing shape implemented by the thing template, are automatically present on all equipment of that equipment type.

Create a new thing template for each new equipment type. You can use PTC.ISA95.EquipmentThingTemplate as the Base Thing Template for your new thing template, or your new thing template can implement the required thing shapes and any additional thing shapes needed for specific functionality that you want on the equipment. An individual thing template should be associated with only one equipment type.
Thing templates for equipment types that you want to appear in Asset Advisor must implement the following thing shapes:

- PTC.Factory.PhysicalAssetThingShape
- PTC.SCA.SCO.AssetIdentifierThingShape
- PTC.SCA.SCO.StatusThingShape
- PTC.ISA95.IdentifierThingShape
- PTC.SCA.SCO.MonitoredPropertiesThingShape

Thing templates for all other equipment types must implement the following thing shapes:

- PTC.ISA95.DisplayNameThingShape
- PTC.ISA95.IdentifierThingShape
- PTC.SCA.SCO.IdentifierThingShape
- PTC.ISA95.GeneralEquipmentThingShape
- PTC.SCA.SCO.MonitoredPropertiesThingShape

Implement the following thing shapes in your thing template to include specific functionality in your equipment type:

- PTC.Factory.ShiftThingShape and PTC.SCA.SCO.KPIsThingShape—KPI calculations. For more information, see “Performance Metrics” in the ThingWorx Apps Setup and Configuration Guide.
- PTC.SCA.SCO.StatusThingShape—Status calculations. For more information, see the “Status” in the ThingWorx Apps Setup and Configuration Guide.
- PTC.SCA.SCO.AnomalyThingShape—Anomaly detection capabilities. For more information, see Anomaly Detection and Asset Advisor on page 147.
- PTC.SCA.SCO.AssetIdentifierNumberThingShape—Provides the modelNumber and serialNumber properties for equipment that appears in Asset Advisor.
- PTC.SCA.SCO.FileTransferHistoryHandlerThingShape—Provides file transfer capabilities for equipment that appears in Asset Advisor. For more information, see “Remote Access and Control” in the ThingWorx Apps Setup and Configuration Guide.
- RemoteAccessible—Provides remote access capabilities for equipment that appears in Asset Advisor (including Axeda assets). For more information, see “Remote Access and Control” in the ThingWorx Apps Setup and Configuration Guide.
• **PTC.SCA.SCO.RemoteTunnelingThingShape**—Provides remote access capabilities for equipment that appears in Asset Advisor (does not apply for Axeda assets). For more information, see “Remote Access and Control” in the *ThingWorx Apps Setup and Configuration Guide*.

• **PTC.ISA95.GeneralPhysicalAssetThingShape**—Defines links between an asset and other objects in the ISA 95 thing model.

## Creating a Configuration Menu for an Equipment Type

The configuration menu for an equipment type is specified in the `ConfigurationMenu` field on the `EquipmentTypeSettings` configuration table. The configuration menu includes the list of links to mashups that are displayed after an instance of the equipment type is created from the `Equipment` tab of `Configuration and Setup` in ThingWorx Apps. For example:

![thingworx configuration and setup](image)

A generic configuration menu that includes only the **General Information** and **Equipment Structure** links is provided by default when defining new equipment (**PTC.SCA.SCO.EquipmentConfigurationMenu**). You can use the generic configuration menu, or you can create new custom configuration menus.
A configuration menu template (PTC.SCA.SCO.EquipmentConfigurationMenuTemplate) is provided to be used as a starting point for creating your own equipment type configuration menus. This template links to all mashups required to enable the various configuration menu features:

- **General Information**—PTC.SCA.SCO.EquipmentGeneralInfo
- **Equipment Structure**—PTC.SCA.SCO.EquipmentStructure
- **Status**—PTC.SCA.SCO.EquipmentStatus
- **Additional Properties**—PTC.SCA.SCO.AdditionalProperties
- **Performance Metrics**—PTC.SCA.SCO.KPI
- **Alerts**—PTC.SCA.SCO.Alerts

For the mashups to be functional and features enabled, the appropriate thing shapes that are related to the functionality must be present on the equipment instance (thing). Ensure that the appropriate thing shapes are implemented on the thing template for the equipment type, so that each equipment instance automatically inherits them.

The following thing shapes are required to enable the following features in a configuration menu:

- **Status definition** (Status)—PTC.SCA.SCO.StatusThingShape
- **KPI calculation** (Performance Metrics)—PTC.Factory.ShiftThingShape and PTC.SCA.SCO.KPIsThingShape

## Defining Property Display for Equipment Types

Define the properties for an equipment type on the thing template for that equipment type, or on the thing shapes implemented by the thing template. This ensures that all equipment of that equipment type inherits the same properties.
In ThingWorx Apps, there are three categories of equipment properties: properties that can be bound, trended, or displayed. The properties displayed for each category are controlled by services that the equipment type thing template inherit from the PTC.SCA.SCO.MonitoredPropertiesThingShape thing shape:

- **GetTagPickerProperties** service—Returns properties of any base type that can be bound using the tag picker common component. The type of properties displayed in the tag picker common component can be configured further for each usage of the tag picker. For more information, see Customizing the Tag Picker Common Component on page 121.

- **GetMonitoredProperties** service—Returns properties that are trended on the Monitored Properties page in Asset Advisor. Only numerical and boolean base type properties that are logged are returned.

- **GetAdditionalProperties** service—Returns properties for which the name and value are displayed on the Additional Properties page for an asset in Asset Advisor.

**Note**

The Additional Properties table for equipment being configured from the Equipment tab of Configuration and Setup displays all properties returned by these three services, with no duplicates.

Each service can be customized independently on the thing template for an equipment type.

The properties returned by each service are calculated from the following property sets:

- **Inherited properties**—Properties defined in the thing template and implemented thing shapes that are inherited by the thing.

- **My properties**—Properties defined directly on the thing, added either through the ThingWorx Apps user interface or in ThingWorx Composer. These properties appear in the My Properties table on the Properties and Alerts page for the thing in ThingWorx Composer. These properties are always returned.

- **Service properties**—Properties specified in the myList variable in the service.

- **Excluded properties**—Inherited properties to be excluded by default from the ThingWorx Apps user interface if nothing is specified in the myList variable. These properties are listed in the GetExcludedProperties service, which is called by the other three services.
Each service also contains an `IncludeInheritedProperties` variable, which controls whether inherited properties (with the exception of any excluded properties) are included in the result set. This allows you to easily include or exclude inherited properties from being displayed for the equipment.

The service output is determined as follows:

- **When `IncludeInheritedProperties` is set to true, and no properties are specified in the `myList` variable, all properties in the My Properties table on the thing and all inherited properties (with the exception of any excluded properties) are returned. This is the default behavior for the services.**
- **When `IncludeInheritedProperties` is set to true, and any properties are specified in the `myList` variable, all properties in the My Properties table on the thing and specified in the `myList` variable are returned.**
- **When `IncludeInheritedProperties` is set to false, and no properties are specified in the `myList` variable, only properties in the My Properties table on the thing are returned.**
- **When `IncludeInheritedProperties` is set to false, and any properties are specified in the `myList` variable, all properties in the My Properties table on the thing and specified in the `myList` variable are returned.**

### Configuring the Thing Name Format For Your Equipment Type

When a new instance of an equipment type is created from ThingWorx Apps, a thing is created in ThingWorx with the thing template specified for the equipment type in the `EquipmentTypeSettings` configuration table as its Base Thing Template. By default, the thing name is determined by prefixing the `EquipmentType` value to the `Name` of the equipment as entered on the create window, as follows: `<EquipmentType>_ <Name>`. For example, `Asset_1-2_GantryRobot`.

You can change the thing name pattern by customizing the `SetEquipmentEntityName` service on the resource provider used by an equipment type.

- **If you use the default resource provider (PTC.SCA.SCO.EquipmentResourceProvider) for your equipment type, and customize the `SetEquipmentEntityName` service on that resource provider, the thing name pattern is changed for all equipment types using that resource provider.**
- For the Site, Line, and Asset equipment types, you can change the thing name pattern by customizing the `SetEquipmentEntityName` service on their respective resource providers.
- You can specify the thing name pattern for equipment types on a case-by-case basis by creating specific resource providers for specific equipment types.
Disabling vs. Deleting an Equipment Type

If you already have existing equipment of an equipment type, disable the equipment type rather than deleting it.

An equipment type can be disabled at any time by clearing the `EnableCreation` checkbox for the equipment type in the `EquipmentTypeSettings` configuration table. This prevents further instances of this equipment type from being created in ThingWorx Apps. Only equipment types with the `EnableCreation` checkbox selected are displayed in the Type drop-down list when creating new equipment in ThingWorx Apps. Existing equipment of that equipment type is unaffected.

When an equipment type is deleted from the `EquipmentTypeSettings` configuration table, the association between the equipment type, ThingTemplate, ResourceProvider, and ConfigurationMenu is lost. As a result, although existing equipment of the disabled equipment type is not deleted, it no longer functions as expected.

Defining Mashups and Menus for Equipment Types

The default mashups and menus used in ThingWorx Apps for specific equipment types are defined in the `LaunchPointEntitySettings` on the Configuration page of the launch point configuration thing. Certain areas of ThingWorx Apps can be defined to display different mashups or menus for individual equipment types, including for equipment types that derive from another equipment type. These are defined in the `EquipmentUserInterfaceSettings` configuration table.

To add an entry for an equipment type to the configuration table:

1. Click Add on the `EquipmentUserInterfaceSettings` configuration table.
2. In the `EquipmentUserInterfaceSettings` window:
   a. Enter the `EquipmentType` value for the equipment type. The value entered must match the `EquipmentType` value in the `EquipmentTypeSettings` configuration table.
b. Search for and select the desired mashups or menus for the equipment type:

- **PlantStatusFooterMashup**—Footer portion of Production KPIs when equipment of the equipment type is selected. If no value is defined for an equipment type, the default **PlantStatusFooterMashup** mashup in the **LaunchPointEntitySettings** is used.

- **PlantStatusDetailMashup**—Detail page displayed in Production KPIs when equipment of the equipment type is viewed. If no value is defined for an equipment type, then the View Production History Data icon (_information_ ) on the Production KPIs toolbar is disabled when equipment of this equipment type is selected.

- **AssetDetailContainerMashup**—Detail page displayed in Asset Advisor for equipment of this equipment type. If no value is defined for an equipment type, the default **AssetDetailContainerMashup** mashup specified in the **LaunchPointEntitySettings** is used. Applies only for equipment types configured to appear in Asset Advisor. If no **AssetDetailContainerMashup** is specified for an equipment type which derives from another equipment type that appears in Asset Advisor, the **AssetDetailContainerMashup** defined for the equipment it derives from is used.

- **AssetDetailMenu**—Menu displayed on the detail page for equipment of this equipment type in Asset Advisor. If no value is defined for an equipment type, the default mashup specified for the **AssetDetailMenu** field in the **LaunchPointEntitySettings** is used. Applies only for equipment types with the necessary thing shapes implemented to appear in Asset Advisor.

3. Click **Add** to add the entry to the configuration table.

4. Click **Save** to save the launch point configuration thing.
Customizing Equipment Status

Customizing the Default Status Expressions for Custom Equipment Templates .......... 31
Customizing Equipment Status Evaluation ................................................................. 32
Detecting Bad Quality Data in Custom Code............................................................. 33

This chapter discusses customizing the default status expressions for custom equipment templates. Information on overriding the equipment status evaluation for custom templates or individual pieces of equipment is also provided.
Customizing the Default Status Expressions for Custom Equipment Templates

The status expressions for individual pieces of equipment are configured from the Equipment tab of Configuration and Setup, by viewing the individual equipment and selecting the Status page. When new pieces of equipment are created, defaults are defined for all status expressions. You can define the default status expressions for your custom equipment templates, such as templates you have added or updated to use for custom equipment types on page 18. All equipment created using the custom equipment template automatically inherits these default status expressions.

To define the default status expression configuration for a custom template:

1. In ThingWorx Composer, open your custom thing template.
2. Under Services, click to override the GetDefaultStatusExpressionsConfiguration service.
3. In the script pane, you can edit the expression and order of execution for each of the four customizable statuses: PLANNED_DOWNTIME, WARNING, RUNNING, and UNPLANNED_DOWNTIME.

To change the order of execution, copy and paste the lines into the desired order. To edit the expression, make changes to the gray expression text. For more information on specifying status expressions, see “Specifying Status Expressions” in the ThingWorx Apps Setup and Configuration Guide.
4. Click Done.
5. Save the updated thing template.

6. Validate your customized status expression configuration. The status expression configuration defined on the thing template must be valid for new equipment to be created based on the thing template.
   b. Under `Services`, execute the `ValidateDefaultStatusExpressionsByTemplate` service. As input, enter the name of the thing template with your customized status expression configuration.

   This service validates the following:
   • That the expressions defined are valid.
   • That the order of execution for the expressions is valid (there are no duplicate rows).
   • That there are no errors in the implementation of the `GetDefaultStatusExpressionsConfiguration` service.

   If any of these validations fail, the service output displays an error message providing information on the failure.
   c. Fix any errors and rerun the service until it returns no errors.

**Customizing Equipment Status Evaluation**

To determine the state of a piece of equipment, the system evaluates the status expressions defined on the `Status` page for the equipment in `Configuration and Setup ▶ ▶ Equipment`. The expressions are evaluated in the order in which they appear in the `Define Status` table. The first expression to evaluate to `true` determines the state of the equipment.

The status expressions defined on each piece of equipment are the sole source of information used for evaluating the equipment state. The thing template for each equipment type with status implements the `PTC.SCA.SCO.StatusThingShape`. This thing shape in turn calls the status evaluation logic that is implemented by the `CalculateStatus` service on the `PTC.Factory.StatusExpressionUtils` thing. Each equipment type thing template and equipment thing inherits the `CalculateStatus` service. This service evaluates the status expressions defined on the equipment’s `Status` page in `Configuration and Setup ▶ ▶ Equipment`. For more information, see “Specifying Status Expressions” in the `ThingWorx Apps Setup and Configuration Guide`. 
You can override the **CalculateStatus** service on a custom thing template or individual thing to implement your own status evaluation logic. If you are overriding the **CalculateStatus** service, keep the following in mind:

- The result of any customization for **CalculateStatus** must be set to the `result` variable and must be an integer.
- If the customization does not involve the `thingId` and `item` input parameters, you can reconstruct these by using the following code:
  ```javascript
  var thingId = thingId;
  var item = Things["PTC.Factory.StatusExpressionResourceProvider"].GetById({id:thingId});
  ```
- Customizations on one custom thing template are independent of customizations on any other thing template.
- Customizations for individual pieces of equipment are independent of each other. A customization on an individual piece of equipment (thing) takes priority over any customization on the thing template for that equipment type.

### Detecting Bad Quality Data in Custom Code

If the custom code used to override the **CalculateStatus** service explicitly uses property tags, and you want data quality to be checked, your code needs to address this.

The `IS_DATA_OK` function that is used in status expressions can also be used to detect bad quality data in custom code. The function calls the **GetPropertyQuality** service which checks the quality of the data. This service is available on every thing entity, inherited from the GenericThing thing template. This service takes a `propertyName` string as its only parameter, and returns one of the following string values:

- `UNKNOWN`
- `GOOD`
- `BAD`
- `OUT_OF_RANGE`
- `UNVERIFIED_SOURCE`
To be used in the **GetPropertyQuality** service, the `propertyName` must satisfy two conditions:

- The property must either exist on the thing being used to execute the service, or be a remote property that is remotely bound to the thing being used to execute the service.
- The property name must contain no dots, only double dashes, for example `Channel1--DryingMachine--Status`. If the property name contains dots, your custom code must address this, as shown in the following example.

In most cases, a property tag is composed of two portions, the source thing portion and the property portion, separated by a colon (:`

```
"PTCSCA.SCO.Demo.KepServer.CheeseCake.Fryer.TotalCount"
```

<table>
<thead>
<tr>
<th>Source Thing Portion</th>
<th>Property Portion</th>
</tr>
</thead>
</table>

The property portion must be extracted from the property tag string, and then be used in the execution of the **GetPropertyQuality** service. Both portions are required to implement bad quality data detection when customizing the **CalculateStatus** service.

**Implementation Example**

The following code is an example of custom JavaScript code using the **GetPropertyQuality** service to detect bad quality data on a specific tag.

```javascript
// The source thing tag to be tested for bad quality data. This tag hypothetically
// points to some value on some server

// Split the property tag into 2 portions:
// 1. The source thing portion
// 2. The property portion
var splitResult = propertyTag.split(":");
var sourceThingName = splitResult[0]; // The source thing portion
var propertyName = splitResult[1]; // The property portion
var sourceThing = Things[sourceThingName]; // Get the source thing that has the property
// Convert the property name to a Thingworx friendly name
var propertyName_friendly = propertyName.replace(/\./g, "--");
var statusEvaluationResult;
if (sourceThing !== null) {
  // Additional code...
```

**Note**

The code example converts the property name, which uses dots, to use double dashes, as required by the **GetPropertyQuality** service.
params = {
    propertyName: propertyName_friendlyc /* STRING */
};

var quality = sourceThing.GetPropertyQuality(params); // Calling the service (API) to check the quality

// Here do some processing according to the result of the quality check
if (quality === "GOOD") {
    // Set a certain status when the quality is GOOD
    statusEvaluationResult = 2; // Status "2" = "Running"
} else if (quality === "BAD") {
    // Set a certain status when the quality is BAD
    statusEvaluationResult = 4; // Status "4" = "Unplanned Downtime"
} else {
    // Set a certain status when the quality is anything other than GOOD or BAD (i.e. UNKNOWN, OUT_OF_RANGE, UNVERIFIED_SOURCE)
    statusEvaluationResult = 5; // Status "5" = "Unavailable"
}

result = statusEvaluationResult;
This chapter discusses how to customize the evaluation of key performance indicators (KPIs), and how to customize the help text associated with the customized KPIs on the Performance Metrics page.
Customizing the KPI Evaluation Services

Instead of using the default KPI evaluation formulas provided out-of-the-box, you can specify your customized formulas to calculate KPIs and equipment status.

**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.

The values of the KPIs are displayed in the Production KPIs and Asset Production History Data page in the application, as shown in the following figure.

There are two ways to customize the KPI calculations: overriding the service implementations for individual performance metric calculations, or overriding the overall KPI calculation service implementation. Choose one of these methods, but not both. Customizing the overall KPI calculation service implementation overrides any customizations for the individual performance metric calculations.

To override any of the original service implementations, complete the following steps. More information on each of the provided services for customization is provided in the following sections.

1. In ThingWorx Composer, open the PTC.Factory.StatusExpressionResourceProvider thing for editing.
2. Under *Services*, search for the service to customize.
3. Click 🖼 to override the service.
4. In the script pane, enter the appropriate JavaScript for your customization.

5. Click **Done** to save your customization of the service.

   The customized service displays in the **My Services** table.

6. Click **Save** to save the thing.

---

**Customizing Individual Performance Metric Calculations**

The following services are provided on the

*PTC.Factory.StatusExpressionResourceProvider* thing to
customize individual performance metric calculations. For these services, the
input **thingId** is the name of the thing representing the asset or line whose
status or KPIs are evaluated. The **quality**, **performance**, or
**availability** inputs are the default KPI values calculated out-of-the-box.

**CustomizedAvailabilityCalculation**

This service customizes the calculation for the **Availability** performance metric. It
receives **thingId** and **availability** as inputs.

- The new availability calculation must be assigned to the **result** variable.
  For example, an implementation of “`result = 0.5;`” causes the availability
to be hard-coded to 50%.

- The **thingId** input can be used to retrieve the equipment itself. For more
  information, see Customizing Overall KPI Calculation on page 39.

**CustomizedOEECalculation**

This service customizes the calculation for the **OEE** performance metric. It
receives **thingId**, **availability**, **performance**, and **quality** as inputs.
The new OEE calculation must be assigned to the **result** variable. This service
is otherwise similar to the **CustomizedAvailabilityCalculation**.

**CustomizedPerformanceCalculation**

This service customizes the calculation for the **Performance** performance metric. It
receives **thingId** and **performance** as inputs. The new performance
calculation must be assigned to the **result** variable. This service is otherwise
similar to the **CustomizedAvailabilityCalculation**.

**CustomizedQualityCalculation**

This service customizes the calculation for the **Quality** performance metric. It
receives **thingId** and **quality** as inputs. The new quality calculation must
be assigned to the **result** variable. This service is otherwise similar to the
**CustomizedAvailabilityCalculation**.
Customizing Overall KPI Calculation

The CustomizedKPIsCalculation service is provided on the PTC.Factory.StatusExpressionResourceProvider thing to customize overall KPI calculation.

This service receives thingId and a one-row infotable (item) with information about the equipment, including KPI values. There is no output (result) for this service. KPIs must be set directly in the equipment itself, for example:

```javascript
var quality = item.getRow(0).quality; // Original quality
Things[thingId].quality = quality / 2; // Result is half of original value
Things[thingId].performance=0.3;
Things[thingId].availability=0.4;
Things[thingId].oee=0.5;
```

Note

Do not use CustomizedKPIsCalculation to set status. This results in two schedulers both attempting to set the status, leading to inconsistent results.

Customizing the Help Text for the Performance Metrics Page

The Performance Metrics page for a line or asset, available from the Equipment tab of Configuration and Setup, contains multiple instances of on-screen help text to guide users as they configure the performance indicator thresholds and parameters for performance calculations for the piece of equipment. You can customize this help text as needed for your purposes.
The following graphic identifies the localization tokens corresponding to the help text:

1. PTC.Factory.KPI.Availability.Help
2. PTC.Factory.KPI.Quality.Help
3. PTC.Factory.KPI.Quality.DefaultExpression
4. PTC.Factory.KPI.Performance.Help
5. PTC.Factory.KPI.Performance.DefaultExpression
6. PTC.Factory.KPI.OEE.Help
7. PTC.Factory.KPI.GoodCountHelp
8. PTC.Factory.KPI.TotalCountHelp
9. PTC.Factory.KPI.IdealRunRateHelp

To Customize the Help Text
1. In ThingWorx Composer, open System ▶ Localization Tables.
2. Choose the localization table corresponding to the current language.
3. Under **Localization Tokens**, search for the token that you want to customize.
4. Edit the token value as needed.
5. To localize the custom value, repeat this process in the localization table for each appropriate language.
6. Click **Save**.

For more information, see “Localization Tables” in the *ThingWorx Help Center.*

---

>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.
Excluding Property Base Types when Exporting Equipment

When exporting equipment data from the Equipment tab of Configuration and Setup, properties with a base type of IMAGE, BLOB, PASSWORD, XML, and DASHBOARDNAME are excluded by default.

To customize the list of excluded property base types from export:

1. In ThingWorx Composer, open the PTC.SCA.SCO.Common.ImportExportEquipmentUtils thing.

2. Under Services, find GetExcludedPropertyBaseTypes, and click to override the service.

3. In the Javascript pane, add the base type that you want to exclude from export to the list of base types, enclosed in single quotes. For example, the following code adds the INFOTABLE base type to the list of excluded base types:

   ```javascript
   var result = ['IMAGE', 'BLOB', 'PASSWORD', 'XML', 'DASHBOARDNAME', 'INFOTABLE'];
   ```

4. Click Done to save the change to the service.

5. Click Save to save the thing.

Properties with the INFOTABLE base type are now also excluded from the spreadsheet when equipment is exported.

For more information, see “Importing and Exporting Equipment” in the ThingWorx Apps Setup and Configuration Guide.
This chapter details the overridable services related to importing equipment from the Equipment tab of Configuration and Setup. Example customizations for certain services are provided.
Overridable Import Services

The following services in the import process are overridable:

- On the `PTC.SCA.SCO.Common.ImportExportEquipmentUtils` thing:
  - `GetImportSupportedBaseTypes`

- On the `PTC.SCA.SCO.Common.ParseExcelProcessor` thing:
  - `ParseImportFile`

- On the `PTC.SCA.SCO.Common.ImportValidator` thing:
  - `ValidateEquipment`
  - `ValidateEquipmentHierarchyTab`
  - `ValidateGeneralInformationTab`
  - `ValidateImportFile`
  - `ValidateLocallyBoundProperty`
  - `ValidateNoneBoundProperty`
  - `ValidatePropertiesTab`
  - `ValidateProperty`
  - `ValidateRemotelyBoundProperty`

- On the `PTC.SCA.SCO.Common.ImportProcessor` thing:
  - `Import`
  - `ImportEquipmentHierarchy`
  - `ImportEquipmentInformation`
  - `ImportGeneralInformation`
  - `ImportLocallyBoundProperty`
  - `ImportNoneBoundProperty`
  - `ImportProperties`
  - `ImportRemotelyBoundProperty`
  - `ImportThing`
  - `PreImport`
  - `PostImport`
Service Customization Examples

The following sections provide example customizations for certain overridable services.

Note

Line breaks and spaces have been added to the following code examples for presentation purposes. If you copy-and-paste the code, remove these characters before attempting to use the code in your application.

GetImportSupportedBaseTypes

The GetImportSupportedBaseTypes service allows you to control the property base types supported for import. This example customization excludes a property base type from import.

1. In ThingWorx Composer, open the PTC.SCA.SCO.Common.ImportExportEquipmentUtils thing for editing.

2. Under Services, find GetImportSupportedBaseTypes, and click to override the service.

3. In the script pane, view the list of supported base types.

   ```javascript
   var result = ['Boolean', 'DateTime', 'Integer', 'Long', 'JSON', 'Number', 'String'];
   ```

4. To exclude a base type from import, delete or comment the base type which you want to exclude from the list. For example, to exclude the JSON base type, update the service as follows:

   ```javascript
   var result = ['Boolean', 'DateTime', 'Integer', 'Long', 'Number', 'String'];
   ```

5. Click Done to save the change to the service.

6. Click Save to save the thing.
After this change, if properties with the JSON base type are present in the import spreadsheet, they are not imported. When the spreadsheet is uploaded and validated, properties with the JSON base type are counted as invalid and are skipped during the import.

**Note**

To view details of invalid data for import, access the script logs:

1. In ThingWorx Composer, click **Monitoring** in the left pane.
2. Under **Logs**, click **ScriptLog**.
3. Select a row in the **ScriptLog** table to view the **Log Messages** for that entry.

**ParseImportFile**

This example customizes the import parsing process so that parsing messages are displayed in the **ScriptLog** with a **Warn** level.

1. In ThingWorx Composer, open the `PTC.SCA.SCO.Common.ParseExcelProcessor` thing.
2. Under **Services**, find **ParseImportFile**, and click ** OVERRIDE ** to override the service.
3. In the script pane, change the logger level to **WARN**, as follows:

   ```javascript
   var result = (function () {
      var constants = Things["PTC.SCA.SCO.Common.ImportExportEquipmentUtils"].GetImportExportConstants();
      
      logger.warn("Parsing process started");
      
      // Invoke service which parses equipment data
      var parsedInfoTable = Resources["PTC.Factory.CommonUtilities"].ParseImportFile({
         fileName: fileName /* STRING */,
         dataShape: constants.importExportDataShape /* DATASHAPENAME */
      });

      logger.warn("Parsing process completed");

      return parsedInfoTable;
   })();
   ```

4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.
ValidateEquipment

This example customizes the equipment validation so that equipment is created even when the project specified in the import spreadsheet does not exist.

1. In ThingWorx Composer, open the `PTC.SCA.SCO.Common.ImportValidator` thing.

2. Under **Services**, find **ValidateEquipment**, and click ✍️ to override the service.

3. In the script pane, comment out lines 447 through 480, as shown in the following graphic. This causes the code to skip validating the existence of the project specified for the equipment.

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.

ValidateEquipmentHierarchyTab

This example customization causes the **ScriptLog** to log the start of the **ValidateEquipmentHierarchyTab** service as a **Trace** entry.

1. In ThingWorx Composer, open the `PTC.SCA.SCO.Common.ImportValidator` thing.

2. Under **Services**, find **ValidateEquipmentHierarchyTab**, and click ✍️ to override the service.

3. In the script pane, add the following code at the end of the script:

   ```javascript
   logger.trace("Validation complete for EquipmentHierarchyTab custom service");
   ```

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.

6. Navigate to **Monitoring ▶▶ ScriptLog**, and change the log level settings to **Trace**. The log now includes a **Trace** entry for entering the **ValidateEquipmentHierarchyTab** custom service.
**ValidateImportFile**

This example customization causes the Validation Results table to display the worksheet (tab) name as Hierarchy, rather than Equipment Structure.

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportValidator` thing.

2. Under Services, find ValidateImportFile, and click to override the service.

3. Comment out lines 86 through 90, and change the value of the equation in line 91 from `equipmentHierarchyToken` to `Hierarchy`, as shown in the following graphic:

4. Click Done to save the change to the service.

5. Click Save to save the thing.

**ValidateLocallyBoundProperty**

This example customizes the validation of locally bound properties so that only the specified `PTC.SCA.SCO.Demo.KepServer` thing can be used for locally bound properties:

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportValidator` thing.

2. Under Services, find ValidateLocallyBoundProperty, and click to override the service.

3. In the script pane, replace lines 320 through 333 with the following code:

   ```javascript
   // Check if server thing or property for bounding exists and if thing is same
   // as required
   if (!boundThingExists || thing !== "PTC.SCA.SCO.Demo.KepServer" ||
       property.length === 0) {
       validationResult = me.GenerateReportForEquipment({
           operation: constants.invalid
       });
   }
   ```
if (!boundThingExists) {
    logger.warn("Import Validation Process: [Properties Tab] [Equipment Name: 
    " + thingName + "] Invalid reason:
    Thing does not exist");
}

if (thing !== "PTC.SCA.SCO.Demo.KepServer") {
    logger.warn("Import Validation Process: [Properties Tab] [Equipment Name: 
    " + thingName + "] Invalid reason:
    Property can be locally bounded only to specified thing");
}

if (property.length === 0) {
    logger.warn("Import Validation Process: [Properties Tab] [Equipment Name: 
    " + thingName + "] Invalid reason:
    Property not specified");
}

4. Click Done to save the change to the service.
5. Click Save to save the thing.

ValidateNoneBoundProperty

This example customizes the validation of properties so that for any property with
a None value in the Binding column, the Units value is not checked for changes.

1. In ThingWorx Composer, open the
   PTC.SCA.SCO.Common.ImportValidator thing.

2. Under Services, find ValidateNoneBoundProperty, and click to override
   the service.

3. In the script pane, comment lines 275 through 278, as shown in the following
   graphic:

   ![Commented code]

   This disables the checking of units for properties with the None value for the
   Binding column.
4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.

**ValidatePropertiesTab**

This example customizes the validation of the **Properties** tab of the import spreadsheet when importing equipment, so that if the property name does not start with **InternalPrefix** then it is considered a valid property name.

1. In ThingWorx Composer, open the PTC.SCA.SCO.Common.ImportValidator thing.
2. Under **Services**, find **ValidatePropertiesTab**, and click **✓** to override the service.
3. In the script pane, insert the following code at line 222:

   ```javascript
   isNameValid = !propertyName.startsWith("InternalPrefix");
   ```
4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.

**ValidateProperty**

This example customizes the validation so that any properties specified in the import spreadsheet as having a **Remotely Bound** value in the **Binding** column are considered invalid, and are not created or updated by the import process.

1. In ThingWorx Composer, open the PTC.SCA.SCO.Common.ImportValidator thing.
2. Under **Services**, find **ValidateProperty**, and click **✓** to override the service.
3. Comment lines 33 through 36, as shown in the following graphic:

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.

**ValidateRemotelyBoundProperty**

This example customizes the validation so that if the **Remote Property Name** value for an existing remotely bound property is changed in the import spreadsheet, the validation process does not recognize this change, and the property value is not updated during import.

1. In ThingWorx Composer, open the
   PTC.SCA.SCO.Common.ImportValidator thing.

2. Under **Services**, find **ValidateRemotelyBoundProperty**, and click **done** to override the service.
3. In the script pane, comment lines 179 through 181, as shown in the following graphic:

4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.

**ValidateGeneralInformationTab**

This example customizes the validation so that any equipment defined in the spreadsheet that does not have the required `Equipment_` prefix on its name is considered to be invalid and is not created during import.

To customize the validation of the **General Information** tab when importing equipment:

1. In ThingWorx Composer, open the

   `PTC.SCA.SCO.Common.ImportValidator` thing.

2. Under **Services**, find **ValidateGeneralInformationTab**, and click **✓** to override the service.

3. In the script pane, insert the following code into line 157:

   ```javascript
   if (name.substring(0, 10) !== "Equipment_") {
       reportInfoTable = markEquipment(reportInfoTable, reportDataShapeFields,
                                       constants.invalid);

       logger.warn("Import Validation Process: [General Information Tab]
                   [Row number: " + rowNum + "] Invalid reason:
                   Name does not have required prefix");

       continue;
   }
   ```

4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.
**Import**

This example customization prevents the post-import process from running.

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.

2. Under **Services**, find **Import**, and click to override the service.

3. In the script pane, comment lines 114 through 117, as shown in the following graphic:

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.

**ImportEquipmentHierarchy**

This example customization causes the **ScriptLog** to log the start of the **ImportEquipmentHierarchy** service as a **Trace** entry.

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.

2. Under **Services**, find **ImportEquipmentHierarchy**, and click to override the service.

3. In the script pane, add the following code at the end of the script:

   ```javascript
   logger.trace("Entering ImportEquipmentHierarchy custom service");
   ```

4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.

6. Navigate to **Monitoring ▶▶ ScriptLog**, and change the log level settings to **Trace**. The log now includes a **Trace** entry for entering the **ImportEquipmentHierarchy** custom service.

**ImportEquipmentInformation**

This example customization causes the prefix “E_” to be added to the default display name for any piece of equipment that has no display name value specified in the import spreadsheet. The default display name is the thing name in ThingWorx Composer.

1. In ThingWorx Composer, open the **PTC.SCA.SCO.Common.ImportProcessor** thing.

2. Under **Services**, find **ImportEquipmentInformation**, and click to override the service.

3. In the script pane, insert the following code after the bracket in line 359:

   ```javascript
   else {
     thing.displayName = "E_" + name;
   }
   ```

   The code now appears as follows:

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.
**ImportGeneralInformation**

This example customization causes information about the number of rows processed during the import to be displayed when you view the **ScriptLog** after importing, and change the displayed log level to **Trace**.

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.

2. Under **Services**, find **ImportEquipmentInformation**, and click **✓** to override the service.

3. In the service pane, add the following line of code after line 27 to send log information about processed rows:

   ```java
   logger.trace("Import Process: [General Information Tab] Already processed rows: "+ (i + 1));
   ```

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.

To change the log level displayed in the **ScriptLog**:

1. In the **ScriptLog**, click **Configure**.

2. In the **Configure Log** window, select the desired level from the **Log Level** drop-down list.

3. Click **Apply**.

**ImportLocallyBoundProperty**

This example customization causes locally bound properties created by the import process to be automatically created with a **Units** value of `<local>`.

To customize the import of locally bound properties when importing equipment:

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.

2. Under **Services**, find **ImportLocallyBoundProperty**, and click **✓** to override the service.

3. In the script pane, add the following code to the `createNewProperty` function in line 250:

   ```java
   if (params.units == null) {
       params.units = "<local>";
   }
   ```
The updated code appears as follows:

```javascript
// Original code
var logged = getBooleanValue(equipmentProperty[loggedHeader]);

// Edited code
var logged = true;
```

4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.

### ImportNoneBoundProperty

This example customization causes any properties that are specified in the import spreadsheet with a **Binding** value of **None** to be created in ThingWorx Composer with a **Logged** value of true, regardless of the **Logged** value that is specified in the import spreadsheet.

1. In ThingWorx Composer, open the **PTC.SCA.SCO.Common.ImportProcessor** thing.
   
2. Under **Services**, find **ImportNoneBoundProperty**, and click ✎ to override the service.
   
3. In the script pane, edit line 251 as shown:
   
   - Original line 251:
     
     ```javascript
     var logged = getBooleanValue(equipmentProperty[loggedHeader]);
     ```
   
   - Edited line 251:
     
     ```javascript
     var logged = true;
     ```
   
4. Click **Done** to save the change to the service.
5. Click **Save** to save the thing.
ImportProperties

This example customization causes no locally bound properties to be created by
the import process, even if they are specified in the import spreadsheet. They are
also not reported as invalid rows by the validation process.

To customize the importing of properties when importing equipment:
1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.
2. Under Services, find `ImportProperties`, and click to override the service.
3. Comment lines 39 through 42.
4. Click Done to save the change to the service.
5. Click Save to save the thing.

ImportRemotelyBoundProperty

This example customization causes any property created by the import process
that was specified in the import spreadsheet with a Binding value of Remotely
Bound to have a When Disconnected value of Fold all the changes into a single last
changed value on the Properties and Alerts page for the piece of equipment in
ThingWorx Composer.

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.
2. Under Services, find `ImportRemotelyBoundProperty`, and click to override
   the service.
3. In the script pane, modify lines 235 and 315 to change the value from NONE to
   FOLD.
4. Click Done to save the change to the service.
5. Click Save to save the thing.

PreImport

This example customization causes the ScriptLog to log the start of the PreImport
service as a Trace entry.

1. In ThingWorx Composer, open the
   `PTC.SCA.SCO.Common.ImportProcessor` thing.
2. Under Services, find `PreImport`, and click to override the service.
3. In the script pane, add the following code at the end of the script to enable the
   PreImport custom service:

   `logger.trace("Entering PreImport custom service");`
4. Click Done to save the change to the service.
5. Click **Save** to save the thing.

6. Navigate to **Monitoring ▶▶ ScriptLog**, and change the log level settings to **Trace**. The log now includes a **Trace** entry for entering the **PreImport** custom service.

**PostImport**

This example customization causes the **ScriptLog** to log the start of the **PostImport** service as a **Trace** entry.

1. In ThingWorx Composer, open the `PTC.SCA.SCO.Common.ImportProcessor` thing.

2. Under **Services**, find **PostImport**, and click ![override](image) to override the service.

3. In the script pane, add the following code at the end of the script to enable the **PostImport** custom service:

   ```
   logger.trace("Entering PostImport custom service");
   ```

4. Click **Done** to save the change to the service.

5. Click **Save** to save the thing.

6. Navigate to **Monitoring ▶▶ ScriptLog**, and change the log level settings to **Trace**. The log now includes a **Trace** entry for entering the **PostImport** custom service.
Adding Properties and Fields on Assets, Lines, and Sites

Additional properties and fields, such as Customer and Group, can be added on assets, lines, and sites in the ThingWorx Apps user interface by customizing the appropriate mashups.

Adding Properties and Fields to the Asset Advisor

To display additional properties and fields, such as Customer and Group, on the asset detail page in Asset Advisor, complete the following steps:

1. Modify the thing template used for your assets, so that the assets have the necessary properties, such as customerName and group.

2. Edit the PTC.SCA.SCO.AssetMonitor.C_AssetDetailContainerMashup_[ReleaseVersion] mashup. For the layout containing the PTC.SCA.SCO.AssetMonitor.C_AssetSummaryMashup_[ReleaseVersion], increase the HeaderHeight to allow more space for the property table to display additional properties.

3. In the PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion], under Configuration, edit the AssetDetailContainerMashup entry to point to PTC.SCA.SCO.AssetMonitor.C_AssetDetailContainerMashup_[ReleaseVersion].

4. On the PTC.SCA.SCO.AssetMonitor.AssetDetail.AssetDetailServiceController thing, override the GetIdentityInfo service, using the following code as an example:

```javascript
var asset = Things[assetId];
var result = Resources["InfoTableFunctions"].(CreateInfoTableFromDataShape...)
```
if (asset !== null) {
    var newEntry = new Object();
    newEntry.name = assetId;
    newEntry.displayName = asset.displayName;
    newEntry.image = asset.assetImage;
    if(asset.assetImage !== null){
        newEntry.customClass = "";
    } else{
        newEntry.customClass = "defaultImage";
    }
    var assetAttributes = me.GetAssetAttributes({
        assetId: assetId /* THINGNAME */
    });
    newEntry.attributes = assetAttributes;

    var newAttributeEntry = new Object();
    newAttributeEntry.name = "Customer:";
    newAttributeEntry.baseType = "STRING";
    newAttributeEntry.value = asset.customerName;
    newEntry.attributes.AddRow(newAttributeEntry);

    newAttributeEntry = new Object();
    newAttributeEntry.name = "Group:";
    newAttributeEntry.baseType = "STRING";
    newAttributeEntry.value = asset.group;
    newEntry.attributes.AddRow(newAttributeEntry);

    result.AddRow(newEntry);
}

Adding Properties and Fields to the General Information Page for Equipment.

To display additional properties and fields to the General Information page in Configuration and Setup ▶️ Equipment, complete the following steps:

1. Duplicate the mashup to override:
   - For an asset: PTC.Factory.ManufacturingElementGeneralInfo
   - For a line: PTC.SCA.SCO.LineGeneralInfo
   - For a site: PTC.Factory.SiteGeneralInfo
2. Update the duplicate mashup as necessary, and save it with a new name.
3. Open the PTC.SCA.SCO.UIProvider thing.
4. Under **Services**, click to override the **GetResourceConfigurationView** service.

5. In the script pane, add content similar to the following. Where `mashupName` is set, replace the default value with the name of your customized duplicate mashup, as appropriate.

   ```javascript
   var isSite = Resources["PTC.Factory.CommonUtilities"].isSite({
     thingId: displayId
   });
   var isLine = Resources["PTC.Factory.CommonUtilities"].IsLine({
     thingId: displayId
   });
   var isAsset = Resources["PTC.Factory.CommonUtilities"].IsAsset({
     thingId: displayId
   });
   var mashupName = "PTC.Factory.ManufacturingElementGeneralInfo";
   if (isSite)
     mashupName = "PTC.Factory.SiteGeneralInfo";
   else if (isLine)
     mashupName = "PTC.SCA.SCO.LineGeneralInfo";
   else if (isAsset)
     mashupName = "PTC.Factory.ManufacturingElementGeneralInfo";
   var result = Resources["InfoTableFunctions"].CreateInfoTable({
     infoTableName: "infoTable"
   });
   var newEntry = new Object();
   result.AddField({
     name: "mashup", baseType: "STRING"
   });
   result.AddRow({
     mashup: mashupName
   });
   ```

6. **Click Done** to save the overridden service.

7. **Click Save** to save the **PTC.SCA.SCO.UIProvider** thing.
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 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 User Groups under Security.
2. Click New 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 Edit.
8. Click 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 window.
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 User Groups under Security.
2. Click 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 Edit Members.
4. Select your custom role from list on the left, and move it to the list on the right.
5. Click 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 do not want to appear in the application.
Adding a Custom Notification Handler

Two notification delivery methods are provided with the ThingWorx 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 for the My Services table to add a new service with a Service Name of Notify. Under Inputs, add the following input parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Base Type</th>
<th>Data Shape</th>
<th>Infotable Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>definition</td>
<td>NOTIFICATIONDEFINITIONNAME</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>event</td>
<td>INFOTABLE</td>
<td>Event</td>
<td>Just Infotable</td>
</tr>
<tr>
<td>config</td>
<td>INFOTABLE</td>
<td>Select the data shape created in step 1, for example MyNotificationHandlerContent</td>
<td>Just Infotable</td>
</tr>
</tbody>
</table>

a. Click Add under Inputs.
b. On the New Input 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:

```xml
<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.
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 Apps Setup and Configuration Guide.
10

Customizing the Schedulers

Editable Schedulers........................................................................................................................................68
To configure the launch schedule for a scheduler: ......................................................................................68
To enable or disable the scheduler: ..............................................................................................................69
To set the age of the data to keep after each purge: .................................................................69
Editable Schedulers

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.

- **PTC.SCA.SCO.AssetInitializationScheduler**—Used to initialize monitored properties on assets. It runs every 30 minutes.

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. Edit the scheduler in ThingWorx Composer.
2. Under **Configuration**, modify the **schedule** field to set the launch schedule for a scheduler.

   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](https://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm).

3. Click **Save** to save the modified scheduler.
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:

2. Under **Properties and Alerts**, click **next to the value for the** TimeCutoff **property to set the age of the data to be kept after each purge.**
   
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 `3d12h`.
   
   Case, spaces, and order do not affect the value.
3. Click to set the property value.
4. Click Save.
Implementing the NI TestStand Connector

Prerequisites ........................................................................................................................................... 72
Download the NI TestStand Connector ............................................................................................... 72
Set the Environment Variable ............................................................................................................. 72
ThingWorx Composer Configurations ................................................................................................ 73
Setup the Edge Micro Server (EMS) Environment ............................................................................. 74
Install the NI TestStand Plugin .......................................................................................................... 74
Adding Steps in a Test Stand Sequence .............................................................................................. 78

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:

- NI TestStand 2016

Download the NI TestStand Connector

1. Download the “National Instruments TestStand Connector” from the PTC Marketplace.
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 location of the `<NI_TestStand_Connector>\TestStandPlugin\config.ini` file.

For example, if the path for the `config.ini` file is: `D:\codebase\ni-teststand-plugin\config.ini`, then the environment variable value is `D:\codebase\ni-teststand-plugin`.  

**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 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 uses 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. 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**.

4. Edit the remote thing created in step 1 to set **General Information ▶▶ Value Stream** to the name of the newly created value stream.

**Tip**
A value stream is necessary for NI TestStand connections to display as expected in Controls Advisor. For NI TestStand connections created in prior to 8.4, ensure that a value stream has been created and added to the remote thing for the connection.
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 73.
   - port—The port for the EMS connection.
   - username—if authentication was configured for EMS, specify the username. 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**.

![Image of Result Processing window with PTC selected]

**PTC** is inserted as a sequence.

![Image of Result Processing window with enabled checkbox for PTC sequence selected]

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

**Configure the Sequence Model File**

1. In the NI TestStand Sequence Editor, 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 the path `<NI_TestStand_Connector>\TestStandPlugin\NITestStandPlugin.dll`, as shown in the screenshot below. The following table identifies the sequences and steps to be updated, and what each step does.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Step</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Plugin – Initialize</td>
<td>initConnection</td>
<td>Reads the config.ini and sets the parameters from the file, such as <code>host</code>, <code>port</code>, and so on. Returns an error code of -1 if there is a problem with the environment variable.</td>
</tr>
<tr>
<td>Model Plugin – Begin</td>
<td>GetUtilization</td>
<td>Sends a PUT query to ThingWorx for the <strong>Utilization</strong> property, and sets it to true.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Step</td>
<td>Usage</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Returns an error code of -1 if something is wrong with the EMS configuration.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Plugin — UUT Start</th>
<th>isConnectionHealthy</th>
<th>Sends a ping to ThingWorx to check the connection. Sets the error code to -1 if the connection is not healthy.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Model Plugin – UUT Done</th>
<th>sendThingworx</th>
<th>Updates the TestResult and TestName properties in ThingWorx from the current running unit under test (UUT).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Model Plugin — UUT Done</th>
<th>sendingJSONResult</th>
<th>Updates the TestStandJSONResult property in ThingWorx with a JSON object consisting of information retrieved from NI TestStand.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Model Plugin — End</th>
<th>GetUtilization</th>
<th>Sends a PUT query to ThingWorx for the Utilization property, and sets it to false. Returns an error code of -1 if something is wrong with the EMS configuration.</th>
</tr>
</thead>
</table>

3. Save your updated PTC.seq file.
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 File on page 75.
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing to which you are sending a property value.</td>
<td>&quot;TestStandRemoteThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the property to which you are sending the value.</td>
<td>&quot;PumpEnabled&quot;</td>
</tr>
<tr>
<td>arg3</td>
<td>Boolean</td>
<td>Boolean value of sent from NI TestStand.</td>
<td>Locals.PumpEnabled</td>
</tr>
</tbody>
</table>
- **sendNumberValue**—

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing to which you are sending a property value.</td>
<td>&quot;TestStandThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the property to which you are sending the value.</td>
<td>&quot;PumpValue&quot;</td>
</tr>
<tr>
<td>arg3</td>
<td>Double</td>
<td>Numeric value set from NI TestStand.</td>
<td>Locals.PumpValue</td>
</tr>
</tbody>
</table>

- **sendStringValue**—

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing to which you are sending a property value.</td>
<td>&quot;TestStandRemoteThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the property to which you are sending the value.</td>
<td>&quot;SerialNumber&quot;</td>
</tr>
<tr>
<td>arg3</td>
<td>String</td>
<td>String value sent from NI TestStand.</td>
<td>Locals.SerialNumber</td>
</tr>
</tbody>
</table>
To get values from ThingWorx

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

- **getBooleanValue**—

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Boolean</td>
<td>A <strong>Locals</strong> variable to which to bind the return value.</td>
<td>Locals.PumpEnabled</td>
</tr>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing from which you are retrieving a property value.</td>
<td>&quot;TestStandRemoteThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the property from which you are retrieving the value.</td>
<td>&quot;PumpEnabled&quot;</td>
</tr>
</tbody>
</table>
getNumericValue—

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return Value</td>
<td>Double</td>
<td>A Locals variable to which to bind the return value.</td>
<td>Locals.PumpValue</td>
</tr>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing from which you are retrieving a property value.</td>
<td>&quot;TestStandRemoteThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the property from which you are retrieving the value.</td>
<td>&quot;PumpValue&quot;</td>
</tr>
</tbody>
</table>
- **getStringValue**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing from which you are retrieving a property value.</td>
<td>&quot;TestStandRemoteThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the property from which you are retrieving the value.</td>
<td>&quot;SerialNumber&quot;</td>
</tr>
<tr>
<td>arg3</td>
<td>String</td>
<td>As strings cannot be returned to NI TestStand, this value is the location where the value is stored.</td>
<td>&quot;Locals.SerialNumber&quot;</td>
</tr>
</tbody>
</table>
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**—

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>arg1</td>
<td>String</td>
<td>The name of the thing on which you are executing a service.</td>
<td>&quot;TestStandRemoteThing&quot;</td>
</tr>
<tr>
<td>arg2</td>
<td>String</td>
<td>The name of the service to execute.</td>
<td>&quot;GetEventSubscriptions&quot;</td>
</tr>
<tr>
<td>arg3</td>
<td>String</td>
<td>The payload being sent, such as a JSON payload matching the ThingWorx service input parameters, if needed.</td>
<td>&quot;{&quot;TestName&quot;:&quot;Pump1&quot;}&quot;</td>
</tr>
<tr>
<td>arg4</td>
<td>String</td>
<td>The output, such as a local variable in NI TestStand where the result is stored.</td>
<td>Locals.JSONResult</td>
</tr>
</tbody>
</table>
Displaying Azure IoT Hub Data as Property Values on an Asset

Prerequisites................................................................................................................................................87
Displaying Azure IoT Hub Data in ThingWorx Apps................................................................................87

This chapter describes how to display Azure IoT Hub data as property values on an asset using ThingWorx Composer.
Prerequisites

- A minimum Java Runtime Environment (JRE) of 1.8
- ThingWorx Composer 8.3.0 or higher with ThingWorx Apps installed.
- Azure IoT Connector Extension installed on ThingWorx.
  - Download the extension from the PTC Marketplace.
  - For more information, see “ThingWorx Azure IoT Hub Connector” in the ThingWorx Connection Services Help Center.
- Azure IoT Hub installed.
- A ThingWorx application key created for the ThingWorx Azure IoT Hub Connector, for example azure_appkey.

Displaying Azure IoT Hub Data in ThingWorx Apps

Note

To display an Azure IoT Hub thing as an asset in ThingWorx Apps, without making Azure IoT Hub data able to be browsed from the tag picker common component, you only need to edit the thing to add the Implemented Shapes listed in step 3, and can skip the rest of the following procedure.

To display Azure IoT Hub data for asset properties in ThingWorx Apps:

1. In ThingWorx Composer, create a new resource provider thing template, to add the Azure IoT Hub connection to the Equipment Type drop-down list in the tag picker common component:
   - **Name**—The name for the thing template, for example AzureIot_ResourceProvider_ThingTemplate.
   - **Base Thing Template**—GenericThing
   - **Implemented Shapes**—
     - PTC.SCA.SCO.RemoteConnectionResourceProviderThing Shape
Under **Services**, click 🔄 to override the **GetConnectedServers** service with the following code. (Line breaks and spaces have been added to the following code for presentation purposes. If you cut-and-paste the code, remove these characters before attempting to use the code in your application.)

```javascript
var params = {
    infoTableName : "InfoTable",
    dataShapeName : "PTC.Factory.OPCDAServerNames"
};
var infoTableWithName = Resources["InfoTableFunctions"].
    CreateInfoTableFromDataShape(params);

var params = {
    columnName: "ResourceProviderName",
    value: me.name
};

// result: INFOTABLE dataShape: PTC.SCA.SCO.RemoteConnectionSettingsTableDataShape
var result = Things["PTC.Factory.LaunchPointConfigurationThing"].
    GetRemoteConnectionTableRowByStringProperty(params);

var thingTemplateName;
if (result) {
    thingTemplateName = result.getRow(0).RemoteConnectionThingTemplateName;
}

var paramsForThingTemplate = {
    maxItems : 1000,
    nameMask : undefined,
    query : undefined,
    thingTemplate : thingTemplateName,
    tags : undefined
};
var thingsWithTemplate = Resources["SearchFunctions"].
    SearchThingsByTemplate(paramsForThingTemplate);
for (var i = 0; i < thingsWithTemplate.rows.length; i++) {
    var row = thingsWithTemplate.rows[i];
    // infotable dataShape iteration
    var name = row.name;
    var isThingEnabled = Things[name].IsEnabled();
    if (name !== "" && name !== null && isThingEnabled) {
        var serverName = row.ServerName;
        var rowObject = new Object();
```
rowObject.Name = name;
if (serverName === null || serverName === undefined ||
    serverName === "") {
    serverName = name;
}
rowObject.ServerName = serverName;
infoTableWithName.AddRow(rowObject);
}
}
result = infoTableWithName;

For information on the other services related to the
PTC.SCA.SCO.RemoteConnectionResourceProviderThing
Shape thing shape, see step 1 under Browse Data from Custom
Connectors on page 122.

2. Create a resource provider thing implementing the thing template created in
step 1:
   • **Name**—For example, AzureIot_ResourceProviderThing.
   • **Base Thing Template**—Specify the new thing template created in step 1. In
     this example, AzureIot_ResourceProvider_ThingTemplate.

3. Create a thing template to be used as for creating Azure IoT Hub assets
   (things) with the following settings:
   • **Name**—For example, AzureIot_RemoteThing_ThingTemplate.
   • **Description**—A description of the thing template.
   • **Base Thing Template**—AzureIotThing (or any thing template that
     extends from AzureIotThing)
   • **Implemented Shapes**—Implement all required thing shapes listed for
     assets in Creating Custom Thing Templates for Equipment Types on page
     22.

4. Define an equipment type for your Azure IoT Hub assets in the
   EquipmentTypeSettings configuration table on the PTC.Factory.C_
   LaunchPointconfigurationThing_[ReleaseVersion] following the instructions in Creating Custom Equipment Types and
   Structures on page 18.
   • For the **ThingTemplate** value, specify the thing template created in step 3.
   • For the **ResourceProvider** value, specify the resource provider created in
     step 2.

5. Add the Azure IoT Hub connection type to the launch point configuration
   thing.
   a. Open PTC.Factory.C_LaunchPointConfigurationThing_
      [ReleaseVersion].

Displaying Azure IoT Hub Data as Property Values on an Asset 89
b. Add a new row to the RemoteConnectionSettings table with the following settings:

- **ConnectionType**—Enter the name that you want displayed for the Azure IoT Hub connection type on the Create Connection window in Controls Advisor. For example, AzureIot.
- **ResourceProviderName**—Enter the resource provider thing created in step 2. In this example, AzureIot_ResourceProviderThing.
- **RemoteConnectionThingTemplateName**—Select the AzureIotThing thing template.
- **EnabledInControlsAdvisor**—Ensure that this checkbox is cleared.
- **EnabledInTagPicker**—Select this checkbox to make the connection type visible in the tag picker.

**Note**

If you create an asset based on the AzureIotThing thing template as a thing in ThingWorx Compositor, you must set the displayName property on the thing for the asset to display correctly in ThingWorx Apps. If assets are created from Configuration and Setup ▶️ Equipment in ThingWorx Apps, this property is automatically set.
Customizing Presentation

The chapters in this section provide information on customizing how your information is presented.
13

Using the Launch Point Configuration Thing

Example Customization Using the Launch Point Configuration Thing ...........................................93
Changing the Tiles in the Main Application Console .................................................................96
Changing the Logo and Text on the Welcome Page ...............................................................104
Changing or Adding New Logos in the Application Headers ..................................................106
Changing the TagCountAboveLimit Alert Default .......................................................................107
Other Customizable Launch Point Mashups ..................................................................................107

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.

2. Edit the `gridadvanced-DeviceList` widget to keep only the **Device Name**, **Channel**, and **Device ID** columns displayed in the grid.
4. Go to **Properties and Alerts**, 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. You can also change the descriptive help text provided for each tile. The tiles in the application are controlled by
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. Ensure that the **TilesDataTable** launch point value in PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion] points to PTC.FactoryConsole.C_DataTable_[ReleaseVersion].
2. Open the PTC.FactoryConsole.C_DataTable_[ReleaseVersion] datatable, and click Mashup to view the data table editor mashup.

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]
- **Controls Advisor tile**: PTC.Factory.KEPServerEX.C_ListServers_[ReleaseVersion]
- **Production KPIs tile**: PTC.Factory.C_PlantStatus_[ReleaseVersion]
To Change the Descriptive Help Text for a Tile

Each tile in the main console can display descriptive help text when the user hovers over the information icon 🔄. When the user clicks the information icon, the descriptive help text displays until the user dismisses it by clicking the close icon ✗.

Use the following steps to change, add, or remove descriptive help text for a tile.

1. Ensure that the TilesDataTable launch point value in PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion] points to PTC.FactoryConsole.C_DataTable_[ReleaseVersion].

2. Open the PTC.FactoryConsole.C_DataTable_[ReleaseVersion] datatable, and click Mashup to view the data table editor mashup.

3. In the Data Table Entries table, select the row for the desired tile.

4. In the Entries Editor on the right, enter or edit the values for TileDescription and LocalizedDescription as needed:
   - **TileDescription**—If you are using localized text, enter the localization token, enclosed by double brackets. For example [[MyTileToken]]. This field can be left blank. For more information, see “Localization Tables” in the ThingWorx Help Center.
   - **LocalizedDescription**—Enter help text directly in this field. If there is no localized value for the current locale in the localization token specified for TileDescription, or if TileDescription is left blank, then the LocalizedDescription value is displayed. This field can be left blank.
   - If both TileDescription and LocalizedDescription are blank, no information icon displays on the tile.
5. Click **Add/Update**.
6. Click **Save**. You can now view the help text for the tile in the main apps console.

---

**Note**

The available display space for the descriptive help text on a tile is four lines of approximately 28 characters each, depending on the width of the characters in the display font.

If your text string exceeds the display limit for a line, the space character that is closest to the character limit for the line is used as a line break.

Any characters that fall beyond the character limit for the line, but before the space used for the line break, are not displayed.

View your specified descriptive help text on the tile for each appropriate locale to ensure that it displays as expected.
Example

Following is a simple example modifying the PTC.FactoryConsole.C_DataTable_[ReleaseVersion] to add a new tile with the Production KPIs label, which launches a mashup with the Refresh Now button made visible.

1. Edit PTC.Factory.C_PlantStatus_[ReleaseVersion], select autoRefreshTreeGrid, and select the Visible checkbox. Click Save. This makes the Refresh Now button visible.
2. Complete the following steps to add the new **Production KPIs** tile.
   a. Open the `PTC.FactoryConsole.C_DataTable_[ReleaseVersion]` datatable, and click **Mashup** to view the data table editor mashup.
   b. In the **Data Table Entries** table on the left, select the entry for which the **TileMashupOrURL** value is `PTC.Factory.PlantStatus`.
   c. In the **Entries Editor** on the right, set **TileMashupOrURL** to `PTC.Factory.C_PlantStatus_[ReleaseVersion]`, and enter a unique **TileOrder**.
   d. Click **Add/Update**. This adds a new entry to the **Data Table Entries** table, and the new tile to the console.

3. Click **Save** to save the datatable.
4. Note that the role-based visibility of a tile is the same as its **TileMashupOrURL**. Open `PTC.Factory.KEPServerEX.C_PPlantStatus_[ReleaseVersion]` in ThingWorx Composer, and then click **Permissions**. The current visibility is visible for `PTC.Factory.MachineVisibility:PlantStatus`, which is the same as the default mashup `PTC.Factory.PlantStatus`. As a result, this **Production KPIs** tile in the console is visible to a Production Manager in the application console, but not to a Maintenance Engineer. Use the **Search Organizations** field to find and add another organization to the **Visibility** table. Then remove the original visibility setting. In this way, the role-based visibility of both the mashup and the tile are customized at the same time.

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


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.

Changing the TagCountAboveLimit Alert Default

When a KEPServerEX server is connected for the first time, a TagCountAboveLimit alert is automatically created for the server. This alert is triggered when the number of tags being monitored on the server exceeds the specified limit, potentially impacting the server performance. The default limit for this alert is 10,000. Users with the Controls Engineer, Maintenance Manager, and Maintenance Engineer roles can view and edit this alert definition in the Alerts tab of Configuration and Setup. Administrators can change the default limit setting in ThingWorx Composer.

To change the default limit for this alert:

2. In Configuration, find the AlertCustomizableSettings table, and enter a new value for defaultTagCountAboveLimit.

```
AlertCustomizableSettings

<table>
<thead>
<tr>
<th>defaultTagCountAboveLimit</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
</tr>
</tbody>
</table>
```

3. Click Save.

The new limit is applied to TagCountAboveLimit alerts for servers created and connected after the default limit has been changed. The TagCountAboveLimit alerts for any existing connected servers remain unchanged.

Other Customizable Launch Point Mashups

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

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:**

![Image of Asset Monitor](image-url)
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:**

![Image of Asset Detail Action Menu](image-url)
Changing Labels in the Application

Labels in the application are controlled by the localization tables. To change a label:

1. In ThingWorx Composer, open System ▶ Localization Tables.
2. Choose the localization table corresponding to the current language.
3. Under Localization Tokens, search for the token that you want to change. Click the token name to open the editing pane.
4. Edit the token value, and click ✔.
5. To localize the custom value, repeat this process in the localization table for each appropriate language.
6. Click Save.

For more information, see “Localization Tables” in the ThingWorx Help Center.

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.
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:
Changing Site and Line Labels

You can change the **Site** and **Line** labels in the application to labels that are more meaningful for your purposes, for example to **Customer** and **Group**. The following table lists the localization tokens to modify for this change to be made throughout the application.

<table>
<thead>
<tr>
<th>Localization Token</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTC.SCA.SCO.AssetMonitor.AssetList.</td>
<td>Select a Related Line</td>
</tr>
<tr>
<td>SelectARelatedLine</td>
<td></td>
</tr>
<tr>
<td>PTC.SCA.SCO.AssetMonitor.AssetList.</td>
<td>Select a Related Site</td>
</tr>
<tr>
<td>SelectARelatedSite</td>
<td></td>
</tr>
<tr>
<td>PTC.ISA95.ProductionLineThingTemplate.</td>
<td>Line</td>
</tr>
<tr>
<td>displayName</td>
<td></td>
</tr>
<tr>
<td>PTC.ISA95.SiteThingTemplate.displayName</td>
<td>Site</td>
</tr>
</tbody>
</table>

Changing the Logos in the Application Console

To change the PTC logos in the application console: .................................................. 117
To keep the PTC logos and add an additional logo in the master console: ................. 118

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.

2. Click **Change** and choose a new image for the **Image** field.

3. Click **Save**.

**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.
Customizing the Duration Display Format

Duration is displayed in two locations in the ThingWorx Apps:

- **Asset Advisor**—

  ![Asset Advisor Screenshot](image1)

  - Model Number: ws10a
  - Serial Number: 400i9y9y
  - Description: 1-1 Line
  - Location: Needham Factory
  - Planned Downtime: 6 hrs 38 mins
  - Alerts:
    - No active
    - Weekly total: 0

- **Alert Monitoring**—

  ![Alert Monitoring Screenshot](image2)

  Duration is displayed in the following format:

  - For times greater than 1 day, in days and hours: **6 days 3 hrs**
  - For times greater than 1 hour, in hours and minutes: **3 hrs 27 mins**
• For times greater than 1 minute, in minutes and seconds: 1 min 42 secs
• For times greater than or less than 1 second, in seconds and milliseconds: 6 secs 78 ms, or 0 secs 12 ms

Note
Milliseconds only display when the includeMS input on the service implementation is set to true. By default, includeMS is set to false.

To customize the format for displaying duration:
1. In ThingWorx Composer, open the PTC.SCA.SCO.TimeFormatHelper thing.
2. Under Services, find the FormatDuration service and click to override and edit the service.
3. Enter custom Javascript code for your desired duration format. For example:
   • To display the duration in seconds only, in the format “## seconds”:

     ```javascript
     result = duration/1000 + " seconds";
     ```
   • To display the duration in seconds and milliseconds using the includeMS input, in the format “## seconds ## milliseconds”:

     ```javascript
     var numOfSeconds = Math.floor(duration / 1000);
     var numOfMilliseconds = duration % 1000;
     
     if (includeMS = true) {
       // Show seconds AND milliseconds
       result = numOfSeconds + " seconds " + numOfMilliseconds + " milliseconds";
     }
     if (includeMS = false) {
       // Show only seconds
       result = numOfSeconds + " seconds";
     }
     ```
4. Click Save and Continue, then click Done to save the updates to the service.
5. Click Save to save the thing.
You can verify your change in ThingWorx Composer by clicking Test for the FormatDuration service, or by viewing Asset Advisor or Alert Monitoring in ThingWorx Apps.
Customizing the Tag Picker Common Component

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 resource provider 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.RemoteConnectionResourceProviderThingShape`
2. 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.
      • **EnabledInTagPicker**—Select this checkbox to make the connection type visible in the tag picker.
As a best practice, disable an equipment type by clearing the 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 PTC Marketplace.

---

**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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MashupName</strong></td>
<td>Search for and select PTC.Factory.CommonTagPicker.</td>
</tr>
<tr>
<td><strong>TargetWindow</strong></td>
<td>Select <strong>Modal Popup</strong>.</td>
</tr>
</tbody>
</table>
| **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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>providerClasses</strong></td>
<td>If this field is left empty, only remote connections display, such as KEPServerEX. To additionally display other 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.</td>
</tr>
<tr>
<td><strong>resourceType</strong></td>
<td>Provides an additional filter for the <strong>Equipment Type</strong> drop-</td>
</tr>
</tbody>
</table>

---

124 **ThingWorx Apps Customization Guide**
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Value</td>
<td>down list. If this field left empty, all available equipment types display for the <strong>Equipment Type</strong> 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.</td>
</tr>
<tr>
<td>Determine what displays for the <strong>Equipment</strong> field on the tag picker:</td>
<td></td>
</tr>
<tr>
<td><strong>filterName</strong></td>
<td>If left blank, all available equipment for the selected <strong>Equipment Type</strong> displays in the <strong>Equipment</strong> list. If you want to limit <strong>Equipment</strong> to only certain equipment, enter a regular expression. This value is case-insensitive. This property filters on the <strong>Name</strong> of the thing as shown in ThingWorx Composer, rather than filtering on its display name.</td>
</tr>
<tr>
<td>Determine whether users can select a single tag or property on the tag picker, or can select multiple tags or properties:</td>
<td></td>
</tr>
<tr>
<td><strong>isMultiSelect</strong></td>
<td>Select to enable multi-selection on the tag picker, and to make the <strong>Apply</strong> button visible.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>To limit the number of selectable properties, create an <strong>Expression</strong> widget in the mashup itself.</td>
</tr>
<tr>
<td><strong>informationMessage</strong></td>
<td>Optionally enter a message to display next to the <strong>Equipment</strong> drop-down list. For example, <strong>Select up to 5.</strong></td>
</tr>
<tr>
<td>Determine the data types of tags or properties to display on the tag picker:</td>
<td></td>
</tr>
<tr>
<td><strong>typeFilter</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
Customizing Tab Pages in the Configuration and Setup Tile

Changing the Main Mashup ................................................................. 127
Adding Tab Pages in the Configuration and Setup Main Mashup .................. 127
Granting Access Control to the Tab Page ........................................... 129
Modify the Existing Tab Pages ......................................................... 130

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 96.

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 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 4a. Set the validator to be triggered by the Load event and RefreshRequested event of the mashup. The validator verifies whether the current tab page is the selected one. If yes, it triggers all the behaviors for loading and refreshing the mashup.

5. 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.
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**: 

![thingworx: configuration and setup](image-url)
- **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:**

![Image showing the Users tab in ThingWorx configuration and setup](image_url)
Customizing Controls Advisor

Customizing the Calculation of Remote Server Status ................................................. 135
Including New Server Types or Custom Connectors ..................................................... 137

This chapter provides information on Controls Advisor customizations, including expanding Controls Advisor to include new server types or custom connectors, implementing a custom footer mashup for new servers or connectors on the Controls Advisor page, and customizing the calculation of remote server status. Implementing custom mashups for new servers or connectors is also covered.
Customizing the Calculation of Remote Server Status

The status displayed for a server in Controls Advisor is determined by a combination of two factors:

• Is the server connected?
• If connected, is the data flow sufficiently quick?

The resulting status displays in the Status column on the main Controls Advisor page, and on the server details page:

• ✔—The server is connected, and the data flow is quick.
• !—The server is connected, but the data flow is slow or nonexistent.
• ❌—The server is not connected.

Both factors of the server status calculation can be customized.

Customizing the Calculation of Remote Server Status for KEPServerEX Server Types

A server is considered to be a KEPServerEX server if it was:

• Created in Controls Advisor with a selected Connection Type of KEPServerEX or Legacy and Non-Kepware OPC servers (aggregator server)
• Discovered from a connected aggregator server

To customize the calculation of status for a KEPServerEX server:

1. In ThingWorx Composer, create a new resource provider thing for KEPServerEX connections. This can be done by duplicating the existing resource provider thing for KEPServerEX, PTC.Factory.KepServerResourceProvider.
   a. Ensure that this thing implements the PTC.Factory.KepServerResourceProviderThingTemplate thing template.
   b. Under Services, click ✔️ to override the following services, and add your logic as custom Javascript:
      • CalculateServerStatus—Determines whether the server is connected.
      • CalculateIsConnectionHealthy—Determines whether the data flow is sufficiently quick.
      
      Ensure that the customized services work as expected.
   c. Click Save and Continue to save the customized service.
d. Click **Save** to save the thing template.

2. **Open the** PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion].
   a. Under **Configuration**, find the **RemoteConnectionSettings** configuration table.
   b. Edit the row for the KEPServerEX **ConnectionType**. In the **ResourceProviderName** column, enter the name of the resource provider thing created in step 1.
   c. Click **Save**.

---

**Note**

For servers discovered from a connected aggregator server that display a **Connection Type** of **OPC Server** in Controls Advisor, the data flow speed calculation can be customized by applying a remote binding to the **Server Status** property.

1. In Controls Advisor, select the OPC server, and click ![Edit](image)
2. On the **Edit Connection** window, click ![Select](image), and select a remote tag.
3. Click **OK**.

---

### Customizing the Calculation of Status for Other Server Types

To customize the calculation of status for other server types, such as NI TestStand:

1. In **ThingWorx Composer**, create a resource provider thing template for the server type. For example, **MyServerTypeResourceProviderThingTemplate**.
   a. For **Base Thing Template**, select **GenericThing**.
   b. For **Implemented Shapes**, select **PTC.SCA.SCO.RemoteConnectionResourceProviderThing Shape**.
   c. Under **Services**, click ![Override](image) to override the following services, and add your logic as custom Javascript:
     - **CalculateServerStatus**—Determines whether the server is connected.
     - **CalculateIsConnectionHealthy**—Determines whether the data flow is sufficiently quick.

   Ensure that the customized services work as expected.
   d. Click **Save and Continue** to save the customized service.
e. Click Save to save the thing template.

2. Create a resource provider thing for the server type. For example, MyServerTypeResourceProviderThing.
   a. For Base Thing Template, select the resource provider thing template created in step 1.
   b. Click Save.

3. Create a thing template for the server type. For example, MyServerTypeThingTemplate. This thing template will be used to create a remote thing for this server type.
   a. For Base Thing Template, select RemoteThing.
   b. For Implemented Shapes, select PTC.SCA.SCO.RemoteConnectionThingShape.
   c. Click Save.

   a. For ConnectionType, provide a name for the server type. For example, MyServerType.
   b. For ResourceProviderName, enter the name of the resource provider thing created in step 2. For example, MyServerTypeResourceProviderThing.
   c. For RemoteConnectionThingTemplateName, enter the name of the server type thing template created in step 3. For example, MyServerTypeThingTemplate.
   d. Click Save.

Including New Server Types or Custom Connectors

Controls Advisor can be expanded to display information from new server types or custom connectors.
Adding a New Server Type or Custom Connector

1. In ThingWorx Composer, create a new resource provider thing template with the following settings:
   - **Name**—The name for the thing template, for example MyConnector_ResourceProviderThingTemplate.
   - **Base Thing Template**—GenericThing
   - **Implemented Shapes**—PTC.SCA.SCO.RemoteConnectionResourceProvider-ThingShape

2. Create a resource provider thing implementing the thing template created in step 1:
   - **Name**—For example, MyConnector_ResourceProviderThing.
   - **Base Thing Template**—Specify the resource provider thing template created in step 1. In this example, My_Connector_ResourceProvider_Thing_Template.
   - **Implemented Shapes**—Leave this field empty.

3. Create a thing template for your new server type or custom connector:
   - **Name**—For example, MyConnector_ThingTemplate.
   - **Base Thing Template**—RemoteThing
   - **Implemented Shapes**—PTC.SCA.SCO.RemoteConnectionThingShape.

4. Add your new server type or custom connector to the launch point configuration thing.
   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 your new server type or custom connector on the Create Connection window in Controls Advisor. For example, MyConnector.
      - **ResourceProviderName**—Enter the resource provider thing created in step 2. In this example, MyConnector_ResourceProviderThing.
      - **RemoteConnectionThingTemplateName**—Enter the name of the thing template created in step 3. In this example, MyConnector_ThingTemplate.
      - **EnabledInControlsAdvisor**—Select this checkbox to make the connection type visible in Controls Advisor.
As a best practice, disable a connection type by clearing the checkbox for the table row, rather than deleting the row.

- **EnabledInTagPicker**—Select this checkbox to make the connection type visible in the tag picker common component.

---

**Note**

The values for **ConnectionType**, **ResourceProviderName**, and **RemoteConnectionThingTemplateName** must be unique for each row in the **RemoteConnectionSettings** table.

---

Now, when you click `➕` in Controls Advisor to create a new server, your new server type or custom connector appears in the **Connection Type** drop-down list.

### Populating Controls Advisor Columns with Data from the New Server or Custom Connector

When you create a new server or custom connector, only the link to the server details page, and the **Name**, **Status**, and **Connection Type** values are populated on the Controls Advisor main page. The other column values can be populated by overriding the default implementation of the **GetServerList** service on the resource provider thing created in step 2 of **Adding a New Server Type or Custom Connector** on page 138.

When overriding the **GetServerList** service, use the default service implementation as a reference. Extensive comments are provided in the default implementation code to guide you in your customization.

---

**Note**

- The **Name** and **connectionType** properties are mandatory.
- The table on the Controls Advisor main page is controlled by the `PTC.SCA.SCO.RemoteConnectionServerListDataShape` data shape.
The following properties are automatically populated for every new server or custom connector, but can be overridden if needed:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Corresponding Column in Controls Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>connectionType</td>
<td>Connection Type</td>
</tr>
<tr>
<td>serverStatus</td>
<td>Status</td>
</tr>
<tr>
<td>DetailsPageLink</td>
<td>View</td>
</tr>
</tbody>
</table>

The following properties require some modifications to the overridden `GetServerList` service to display data in the Controls Advisor main page.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Corresponding Column in Controls Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentServerTime</td>
<td>Connection Time</td>
</tr>
<tr>
<td>AlertIcon</td>
<td>Alert</td>
</tr>
</tbody>
</table>
| hasAlerts             | (Not displayed)—Used for AlertIcon
determination |

The following properties pertain specifically to servers or connectors which interact with devices. In most cases, these properties are only used for KEPServerEX type servers.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Corresponding Column in Controls Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayDevicesStatus</td>
<td>Device Status</td>
</tr>
<tr>
<td>NumberOfDevices</td>
<td>Devices</td>
</tr>
<tr>
<td>clientCount</td>
<td>Client Count</td>
</tr>
<tr>
<td>aggregateThing</td>
<td>Aggregator Name</td>
</tr>
</tbody>
</table>
| isCustomRemoteOPCServer             | (Not displayed)—Used for internal
calculations                            |
| TagCount                             | Tag Count                                |
| isRemoteServer                      | (Not displayed)—used for internal
calculations                             |
| devicesStatus                       | (Not displayed)—Used for
displayDeviceStatus
determination |
The following screenshot shows the table definition section found in the default implementation for the `GetServerList` service, for reference:

```
40 // Function that creates 1 row of server info (1 line in the Controls Advisor table)
41 * function createServerInfo(name, connectionType, iconLink) {
42    var rowentry = new Object();
43    rowentry.serverName = name; // STRING - server name - Example value: "open ..."/a"/open"
44    rowentry.displayedName = name; // STRING - server display name - Example value: "open ..."/a"/open"
45    rowentry.description = description; // STRING - description of server - Example value: "This is a description for server".
46    rowentry.detailsPageLink = iconLink; // STRING - details page link - Example value: "open ..."/a"/open"
47    rowentry.serverStatus = undefined; // BOOLEAN - server status - Example value: false.
48    rowentry.serverType = connectionType; // STRING - server type - Example value: "TCP/Server"
49    rowentry.aggregatesCount = undefined; // NUMBER - number of aggregates - Example value: 0.
50    rowentry.serverStatus = undefined; // BOOLEAN - server status - Example value: "false.
51    rowentry.tagCount = undefined; // NUMBER - number of tags - Example value: 0.
52    rowentry.currentServerTime = undefined; // STRING - displayed server time - Example value: "2018-06-26 16:24:07".
53    rowentry.alertCount = undefined; // NUMBER - number of alerts - Example value: 0.
54    rowentry.deviceStatus = undefined; // BOOLEAN - device status - Example value: false.
55    return rowentry;
```

```
```

**Implementing Custom Mashups for your New Server or Custom Connector**

Initially when you create a new server with the `Connection Type` of your new server or custom connector, the `Create Connection` window, the Controls Advisor footer, and the detail page for your newly created server all display a message indicating that they have not yet been set up. The following sections provide instructions for implementing the necessary custom mashups.

To display custom mashups for your new server or custom connector:

1. In ThingWorx Composer, open the `PTC.Factory.C_LaunchPointConfigurationThing_[ReleaseVersion].thing`.
2. Under `Configuration`, find the `RemoteConnectionSettings` table.
3. In the table row for your new server or custom connector, remove the default mashups specified in the following columns by clicking the X to the right of the default mashup name, and add your own custom mashup for each:
   - **FooterMashupName**—Mashup for the Controls Advisor footer
   - **ServerDetailMashupName**—Mashup for the detail page
   - **ConnectionInfoMashupName**—Mashup for the `Create Connection` window
4. Click `Save`.
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:**
  

**Image:**

![Customizer](image_url)
- 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`


Image:
• Default Mashup:
PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertiesWithChart


Image:

• Default Mashup:
PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertyWithChart


Image:
Anomaly Detection and Asset Advisor

Anomaly detection can be enabled and configured through the ThingWorx platform. For more information, see “Anomaly Detection” in the ThingWorx Analytics Help Center.

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 the Configuration tab.
3. Select the enabled checkbox.
4. In the schedule field, set a new update rate or accept the default value.
5. Ensure that the runAsUser field is set to Administrator.
6. Click Save.
7. Click the Services tab.
8. 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.

<table>
<thead>
<tr>
<th>Monitored Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Name</td>
</tr>
<tr>
<td>Current Value</td>
</tr>
</tbody>
</table>

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 147.

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` and `PTC.SCA.SCO.AssetInitializationScheduler` have both been enabled, and the `runAsUser` field on the Configuration page for each scheduler is set to Administrator.
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 Composer:

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

```css
.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.
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.

   ![Configure Service Window]

   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.

   ![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:
Dygraph Widget

The Dygraph widget is useful for displaying any time series chart. In ThingWor x Apps, the widget is used in a number of places:

- In Trending and Troubleshooting, for the chart on a trend detail page (PTC.Factory.TrendTroubleshooting.TrendDetail)
- In Controls Advisor, for the Client Count and Tag Count Trend charts (PTC.Factory.KEPServerEX.DetailPage)
- In Production KPIs, for the Status History and KPI Trends charts on the Production History Data page (PTC.Factory.ProductionHistoricalData, PTC.Factory.C_ProductionHistoricalData_[ReleaseVersion])

These implementations can be referenced as examples.

The Dygraph widget implements a subset of features from the Dygraphs charting library, and has been customized for use in ThingWor x Apps. While the Dygraphs documentation (http://dygraphs.com/) can be a useful resource, only the features that are documented in this chapter are supported for use with the Dygraph widget.

Relevant Properties

- **JSONData**—The data used in the chart is a JSON, sent in string format. A string is used to send the information, rather than an infotable; this is because infotables are passed to the widget line by line, requiring further manipulation of the data for it to be fed to the chart. Using a string allows the data to be sent all at once to the Dygraph library to render the chart. The format for this JSON is CSV. You can create a service to convert an infotable to CSV. For more information, see: http://dygraphs.com/data.html#csv.
An example of the data sent as a string to the **GetJsonStringTrendHistory** service:

```json
{"isCustom":true,"trendType":"PerformanceHistory","displayId":"Asset_CheeseCake_CleaningAndDryingMachine","endDate":1501819199000,"startDate":1501732800000,"isZoomed":false,"zoomedMax":0,"zoomedMin":0}
```

- **ChartTitle, XLabel, YLabel, Y2Label**—Labels for the chart and each axis. If nothing is specified, then no label displays. Padding is automatically added to the chart for the label text.
- **Stepped**—When the checkbox is selected, the chart line is stepped between points, rather than a straight line.
- **LegendPosition**—Determines the position of the legend over the chart. The default value is **On Mouse Over**.
- **CustomBars**—When the checkbox is selected, each CSV cell is parsed as "low;middle;high". Error bars are drawn for each point between low and high, with the series itself going through the middle. This is mainly used as a way to show the range when an averaged value is shown on the graph. This allows display of fewer points, while still showing where a point was higher or lower than expected.
- **FillArea**—When the checkbox is selected, the area defined by line is filled with color. This property is not compatible with a custom bar.
- **StackedGraph**—When the checkbox is selected, the graph lines are stacked.
- **UsePercentageFormat**—When the checkbox is selected, the Y-axis values display as percentages. Applies to the left-side Y-axis only.
- **EnableZoomSynchronizing**—When the checkbox is selected, the chart can synchronize its zoom with another chart.
- **DisableVerticalZoom**—When the checkbox is selected, vertical zoom for the chart is disabled.
- **HideY-AxisValues**—When the checkbox is selected, the numbers for the Y-axis are hidden, but the axis lines still display.
- **zoomMax** and **zoomMin**—When a zoom is performed on the chart, these properties get a new value indicating the minimum and maximum taken from the X-axis, using the JSON data. If using a date-time data format, the time is usually returned in milliseconds.
- **isZoomed**—Allows you to track if the chart is currently zoomed in or not. This is useful on the mashup or service side to handle the different zoom use cases (zoom when already zoomed, update data when zoomed and keeping the zoom.)
- **useValueRange, valueRangeMin, and valueRangeMax**—When **useValueRange** is selected, you can set a minimum (**valueRangeMin**) and maximum (**valueRangeMax**) value for the Y-axis to be rendered.
• **showDecimal**—When the checkbox is selected, labels for decimal values display on the Y-axis.

• **useDateFormat, dateFormat**—Allows you to format the X-axis date. The following function from the Moment.js Javascript library is used: `moment(date).format(XXX)`

• **useAnnotation**—Select this checkbox to use `annotationsJSON` or `annotationsDivName`.

• **annotationsJSON**—The JSON that is parsed to add the annotation to the chart.

• **annotationsDivName**—The HTML `div` element where the annotation information is shown. This is an element displayed outside of the widget itself. In the mashup, using an HTML element and creating the `div` in that element allows you to place the `div` wherever you want it to be placed. An example is the annotations shown when a user clicks a change event box in the KPI Trends chart.

• **DrawGridX, DrawGridY, DrawGridY2**—When the checkbox is selected, the vertical grid line is shown on the chart for that axis.

• **DrawAxisX, DrawAxisY, DrawAxisY2**—When the checkbox is selected, the horizontal axis line with values is shown on the chart for that axis.

• **DisplayTagName**—When the checkbox is selected, the full name of the tag or property displays in the chart legend, along with the value. For long tag or property names, this can result in data being truncated. When the checkbox is cleared, a colored dash coordinating to the colored square for the tag or property name as shown in the information panel to the right of the chart displays instead.

• **ConnectSeparatedPoints**—When the checkbox is selected, any gaps in the data are connected on the chart.

• **DrawPoints**—When the checkbox is selected, all points are permanently displayed on the chart, instead of only when hovering.

• **DataLabel<#>**—If specified, provides the label used for the series. If left blank, the data label can be provided by the `JSONData`. If `DataLabel<#>` is specified, the second Y-axis features cannot be used with that series.

• **SeriesStyle<#>**—The color used for the series data displayed in the chart, and as the colored square in the information panel to the right of the chart.

• **SeriesVisibility<#>**—Determines whether the series data displays in the chart. This property maps to the visibility selection for the series in the information panel to the right of the chart.

• **SeriesSecondAxis<#>**—Determines whether the Y-axis display for the series displays on the left or the right side of the chart. When the checkbox is
selected, the Y-axis displays on the right. When the checkbox is cleared, the Y-axis displays on the left. This property maps to the left or right Y-axis display selection for the series in the information panel to the right of the chart.

- **HideSeries<#>InLegend**—When the checkbox is selected, the series is hidden from the chart legend.

The following graphic shows an example of values specified for **ChartTitle**, **XLabel**, **YLabel**, and **Y2Label**, as well as **ConnectSeparatedPoints** set to true.

![Chart Example](image)

**Events**

- **DoubleClicked**—This event is fired with a double-click is performed. Double-clicking on a chart zooms out using the original **zoomMin** and **zoomMax** values provided with the data.
- **Zoomed**—This event is fired when a zoom is performed. Use this event with **zoomMin** and **zoomMax** to go deeper into the original data.
- **SyncRequest**—If the **EnableZoomSynchronizing** property is enabled, this event is fired when a zoom-in or zoom-out is performed. Use this event to synchronize the zoom feature with another graph. This event must be bound to the **sync** service of the target graph to be synchronized.

**Services**

- **resetZoom**—Manually reset the zoom back to the default state.
- **sync**—Receptor for the **SyncRequest** event to synchronize the graph’s zoom with the calling graph.
Using CustomBars to Add More Information when Zooming

When averaging a large data set, some information can be lost. One strategy to address this is to add the custom range on top of averaging. This shows a transparent colored range behind the lines on the chart, indicating the range in which values are situated. For an example, select **Show Value Range** on a trend detail chart to see a display similar to the following:

If the average is 5 and values go from 1 to 10, then the lines are at 5 and the transparent range shows from 1 to 10. To use this feature, the **CustomBars** property needs to be enabled, and the data needs to be changed accordingly.

Example of the JSON data without **CustomBars** enabled:

```
Date,point1,point2
```

For example:

```
Date,trend1,trend2
2017-03-21,1,9
2017-03-22,2,2
2017-03-23,7,0
2017-03-24,3,5
```

Example of the JSON data with **CustomBars** enabled:

```
Date,minPoint1;point1,maxPoint1,minPoint2;point2,maxPoint2;
```

For example:

```
Date,trend1,trend2
2017-03-21,0;1;8,4;9;9
2017-03-22,1;2;6,0;2;8
2017-03-23,5;7;9,0;0;4
```
Synchronizing the Zoom Level Between Multiple Graphs

To synchronize the zooms between multiple graphs, use the `SyncRequest` event of the source graph, and bind it to the `sync` service of the target graph.

Select the source widget, drag-and-drop the `SyncRequest` onto the target widget, and select the `sync` service.

For bi-directional zoom synchronization, repeat these steps in the opposite direction.
Making an Interactive Zoom

The following steps describe the flow for implementing interactive zoom for the Dygraph Widget in ThingWorx Composer. The chart on a trend detail page is an example of this implementation.

1. Start with an original large data set.
2. If the data set is determined to be too large, average or reduce the number of points on the chart.
3. When the user zooms in, take the `zoomMin` and `zoomMax`, and recalculate the data using this new range, validating as in step 2. If the number of points is small enough, show the raw data instead of averaging or reducing the number of data points.
4. When the user zooms out, return to the original data set with its start time and end time.

Using CSS for the Legend and Labels

The display of the chart legend and labels is formatted using CSS, which can be overridden by providing your custom CSS on the mashup. The default CSS for the chart legend and labels is provided below for your reference. For more information, see “Custom CSS” in the ThingWorx Help Center.

Default CSS for the chart legend:

```css
.dygraph-legend {
  position: absolute;
  z-index: 10;
  font-family: Arial;
  font-size: 10px;
  opacity: 0.75;
  border: 1px solid black
  border-radius: 2px;
  padding: 6px;
  width: 300px; /* labelsDivWidth */
  /*
  dygraphs determines these based on the presence of chart labels.
  It might make more sense to create a wrapper div around the chart proper.
  top: 0px;
  right: 2px;
  */
  background: white;
  line-height: normal;
  text-align: left;
  overflow: overlay;
  left: 600px !important
}
.dygraph-legend-line {
```
Default CSS for labels:

```
.dygraph-legend {
    position: absolute;
    z-index: 10;
    font-family: Arial;
    font-size: 10px;
    opacity: 0.75;
    border: 1px solid black
    border-radius: 2px;
    padding: 6px;
    width: 300px; /* labelsDivWidth */
    /*
    dygraphs determines these based on the presence of chart labels.
    It might make more sense to create a wrapper div around the chart proper.
    top: 0px;
    right: 2px;
    */
    background: white;
    line-height: normal;
    text-align: left;
    overflow: overlay;
    left: 600px !important
}

.dygraph-legend-line {
    display: inline-block;
    position: relative;
    bottom: .5ex;
    padding-left: 1em;
    height: 1px;
    border-bottom-width: 2px;
    border-bottom-style: solid;
    /* border-bottom-color is set based on the series color */
}
```
Deprecation Entities, Services, and Properties

The following entities, services, and properties are deprecated as of the 8.4 release. They will be removed from ThingWorx Apps in release 9.

**Deprecated Entities**

**Deprecated in 8.4:**
- PTC.SCA.SCO.KEPServerEX.ServerThingConnectionInstructions
- PTC.SCA.SCO.KEPServerEX.RetrieveAppKey
- PTC.SCA.SCO.AssetMonitor.AssetList.PageInfoDataShape
- PTC.ISA95.EquipmentAssetMappingThingShape
- PTC.Factory.Administration.ManageLineDetail.Master
- PTC.Factory.Administration.ManagePhysicalAssetDetailTitle
- PTC.Factory.Administration.ManagePhysicalAssetDetail.Master
- PTC.Factory.Administration.ManageSiteDetail.Master

**Deprecated in 8.3.1:**
- PTC.SCA.SCO.Common.ExportProcessHistoryDataShape
- PTC.SCA.SCO.Common.ExportProcessHistoryDataTable
Deprecated in 8.3:
- PTC.SCA.SCO.ConfiguratioAndSetup.ImportExportActions
- PTC.SCA.SCO.Common.AssetHierarchyDataShape
- PTC.SCA.SCO.Common.AssetInformationDataShape
- PTC.SCA.SCO.Common.AssetPropertiesDataShape
- PTC.SCA.SCO.Common.ImportResultDataShape
- PTC.SCA.SCO.Common.ReportDataShape

Deprecated in 8.2:
- 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
- PTC.SCA.SCO.AssetMonitor.AssetDetail.AssetDetailController
- PTC.SCA.SCO.AssetMonitor.AssetList.AssetListController
- PTC.ISA95.EquipmentCapabilityTestResultThingTemplateList
- PTC.ISA95.EquipmentCapabilityTestSpecificationThingTemplateList
- PTC.ISA95.EquipmentClassPropertyAssociationDataShape
- PTC.ISA95.PhysicalAssetCapabilityTestResultThingTemplateList
- PTC.ISA95.PhysicalAssetCapabilityTestSpecificationThingTemplateList
- PTC.ISA95.PhysicalAssetClassPropertyAssociationDataShape
- PTC.ISA95.PhysicalAssetPropertyAssociationDataShape
- PTC.ISA95.ExtendedEquipmentThingShape
- PTC.ISA95.ExtendedPhysicalAssetThingShape
- PTC.ISA95.PropertyManagerThingShape
• PTC.ISA95.EquipmentCapabilityTestResultBaseThingTemplate
• PTC.ISA95.EquipmentCapabilityTestResultThingTemplate
• PTC.ISA95.EquipmentCapabilityTestSpecificationBaseTemplate
• PTC.ISA95.EquipmentCapabilityTestSpecificationThingTemplate
• PTC.ISA95.PhysicalAssetCapabilityTestResultBaseTemplate
• PTC.ISA95.PhysicalAssetCapabilityTestResultThingTemplate
• PTC.ISA95.PhysicalAssetCapabilityTestSpecificationThingTemplate

**Deprecated Services**

Deprecated in 8.4:

- On the PTC.Factory.StatusExpressionResourceProviderTemplate entity:
  - CustomizedStatusEvaluation

- On the PTC.SCA.SCO.AssetMonitor.AssetList.AssetListServiceController entity:
  - GetFilteredAssetList

- On the PTC.SCA.SCO.AssetRemoting.RACHelper entity:
  - GetAssetCapability
  - GetFirstTunnelName
  - IsAssetConnected

- On the PTC.Factory.LaunchPointConfigurationThing entity:
  - IsTemplateExcludedFromResourceCreation

- On the Resources_PTC.Factory.CommonUtilities.xml entity:
  - GetMenu

- On the PTC.Factory.CommonTagPickerUtilities entity:
  - GetAllEquipmentTypes
  - GetSourceThingList

- On the PTC.Factory.TreeHandler resource:
  - GetItemList
○ Methods deprecated along with this service:
  ◆ getTreeAllChild
    (AbstractThingworxAssetResourceProvider.java)
  ◆ getFiltersFromSelectedNode (TreeStructure.java)
  ◆ parseNode (TreeStructure.java)
  - On the PTC.SCA.SCO.UIProvider entity:
    ○ GetListForDropDownMenu
    ○ Methods deprecated along with this service:
      ◆ com.ptc.sca.sco.CommonUIProvider#addTemplateListToDropDown
      ◆ com.ptc.sca.sco.CommonUIProvider#createTemplateRow
  - On the PTC.SCA.SCO.CommonUIProvider entity:
    ○ GetListForDropDownMenu
  - On the PTC.SCA.SCO.UIManager entity:
    ○ GetListForDropDownMenu
  - On the PTC.SCA.SCO.AssetMonitor.AssetList.AssetFiltersController entity:
    ○ GetFiltersClassName
  - On the PTC.SCA.SCO.Common.ImportValidator entity:
    ○ ValidateAsset
    ○ ValidateLine
    ○ ValidateSite
  - On the PTC.SCA.SCO.Common.ImportProcessor entity
    ○ ImportAsset
    ○ ImportLine
    ○ ImportSite
  - On the PTC.SCA.SCO.Common.ImportExportEquipmentUtils entity:
    ○ GetUnlocalizedEquipmentType
  - On the PTC.SCA.SCO.MonitoredPropertiesThingShape entity:
    ○ InitializeMonitoredProperties
• On the
  PTC.SCA.SCO.AssetMonitor.AssetDetail.MonitoredPropertiesController entity:
  ○ GetActiveAlertCount
• On the PTC.SCA.SCO.Asset.AssetHelper entity:
  ○ CreateNewAsset
• On the
  PTC.SCA.SCO.AssetMonitor.AssetList.AssetListController entity:
  ○ synchronizeProperties
• On the PTC.SCA.SCO.AnomalyThingShape entity:
  ○ InitializeMonitoredProperties
  ○ InitializeAnomalyAlertProperties
  ○ InitializeLastCalibrationCompleteTimestamp

Deprecated in 8.3:
• On the
  PTC.Factory.Administration.KepServerMonitorUtils ThingTemplate entity:
  ○ CreateKepServerThing
  ○ DeleteServers
  ○ UpdateServerProperties
• On the PTC.Factory.KepServerMonitorUtils entity:
  ○ GetConnectionTypes
  ○ GetRemoteConnectedServers
  ○ UpdateServerProperties
  ○ DeleteServers
  ○ CreateKepServerThing
• On the PTC.Factory.AlertMonitorUtilities entity:
  ○ QueryAlertSummary
  ○ QueryAlertSummaryByKey
• On the PTC.SCA.SCO.Common.LicenseManager entity:
  ○ GetLicensingStatus
• On the PTC.SCA.SCO.ManageResourceUtils entity:
  ○ SaveExtraProperties
• On the PTC.Factory.CommonUtilities entity:
  ○ GetDefaultShiftForToday
• **ImportData**
  - On the `PTC.SCA.SCO.Common.ImportProcessorTemplate` entity:
    - ImportEquipment
    - ParseExcelFile
    - ValidateEquipment
  - On the `PTC.SCA.SCO.Common.ImportProcessor` entity:
    - ImportEquipment
    - ParseExcelFile
    - ValidateEquipment

**Deprecated in 8.2:**
- 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
  - browseGroups
  - CloneOPCTagPropertiesInfoTable

- On the PTC.Factory.CommonTagPickerUtilities entity:
  - GetAllBindSource

**Deprecated Properties**

Depreciated in 8.4:

- `equipmentCapabilityTestSpecificationID` in the PTC.ISA95.GeneralEquipmentThingShape entity.
- `ResourceCreationSettings` configuration table on the PTC.Factory.LaunchPointConfigurationThing entity.
- `ServiceSessionID` in the PTC.SCA.SCO.AssetService.Connector.ServiceMaxApiPayloadDataShape entity.
- `getThingPropertyList` in the AbstractThingWorxAssetResourceprovider.java entity.
- `monitoredPropertyNames` in the PTC.SCA.SCO.MonitoredPropertiesThingShape entity.
Deprecated in 8.3:

- OPCDAThingName in the PTC.Factory.KepServerProperties entity
- OPCDAThingName in the PTC.Factory.KepServerThingShape entity

Deprecated in 8.2:

- 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.