



TIBCO ActiveMatrix® Service Grid

Policy Director Governance Custom Actions

Version 3.4.3 | February 2025

Contents

Contents	2
Introduction to Custom Actions	4
Setting up the Eclipse Environment for Plug-in Development	5
Implementing a Custom Action	8
Packaging the Action for ActiveMatrix® Service Grid	13
Creating an ActiveMatrix Service Grid Composite Wrapper for a Custom Action	14
Policy Templates	19
Creating an Action Template	19
Creating a Rule Template	22
User Interface	25
Deploying the Custom Policy	26
Deploying Policy Templates	26
Deploying User Interface	26
Restarting the System Node	27
Deploying the Action On ActiveMatrix Service Grid	27
Debugging and Logging	28
Debugging and Logging ActiveMatrix Service Grid Environment	28
Troubleshooting	29
Sample Action Source Code	30
Parameters and Action Templates	31

Sample Action Rule Templates and User Interface Files	34
Code Snippets	36
TIBCO Documentation and Support Services	37
Legal and Third-Party Notices	39

Introduction to Custom Actions

You can configure and enforce policies for TIBCO ActiveMatrix® Service Grid using ActiveMatrix® Service Grid Policy Director Governance. ActiveMatrix Service Grid nodes and ActiveMatrix Service Grid engines have embedded governance agents to enforce policies in the same JVM as the services. Policy enforcement for services in third-party containers is achieved using proxy applications deployed in ActiveMatrix Service Grid nodes.

Governance agents intercept service requests, responses, and fault flows and provide these as Policy Enforcement Points (PEPs) for enforcing policies. When you configure policies, the following information is passed on to the Governance Agent:

- The Governed Object for which you want to enforce the policy (service, reference, and so on.)
- The PEP where you want to run the action (Message In Flow, Out flow, Fault flow, and so on.)
- The action and configuration that makes up the policy (authentication, authorization, and so on.)

ActiveMatrix Service Grid Policy Director Governance supports many types of Governed Objects, PEPs, and actions out-of-the-box. To configure the commonly-used combinations of actions (and action configurations), the PEPs for execution, and the Governed Object types, which are applicable, ActiveMatrix Service Grid Policy Director Governance provides many unconventional Policy Templates. The Security and Logging policies in ActiveMatrix Service Grid Policy Director Governance are examples of out of the box templates.

However, there might be occasions when you want a custom action with a custom policy template to run for a given PEP and Governed Object. This document describes how you can create your own action and define the associated templates and User Interface (UI) to add to the palette of available policies in ActiveMatrix Service Grid Policy Director Governance.

For more details on the governance concepts, see *TIBCO ActiveMatrix® Service Grid Concepts*.

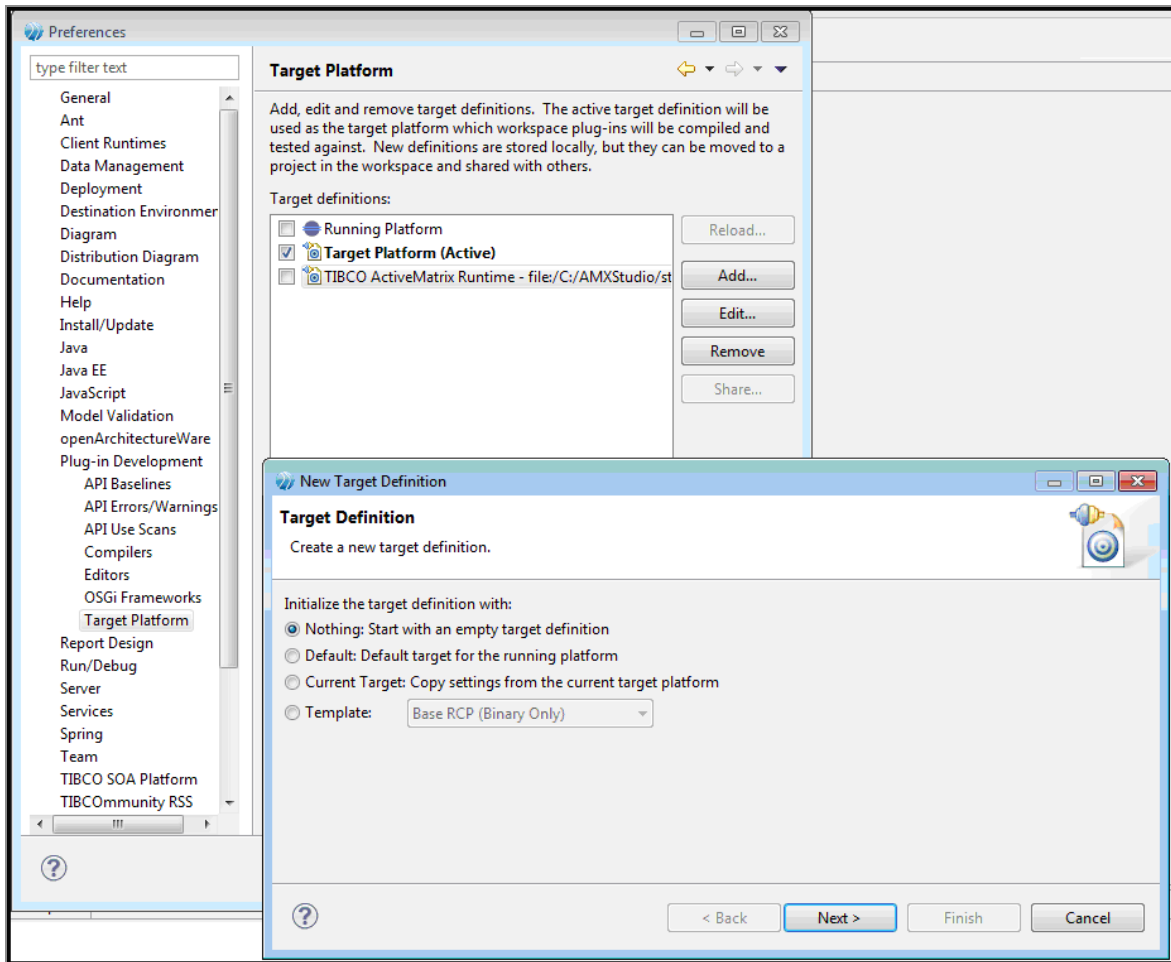
Setting up the Eclipse Environment for Plug-in Development

Before you begin

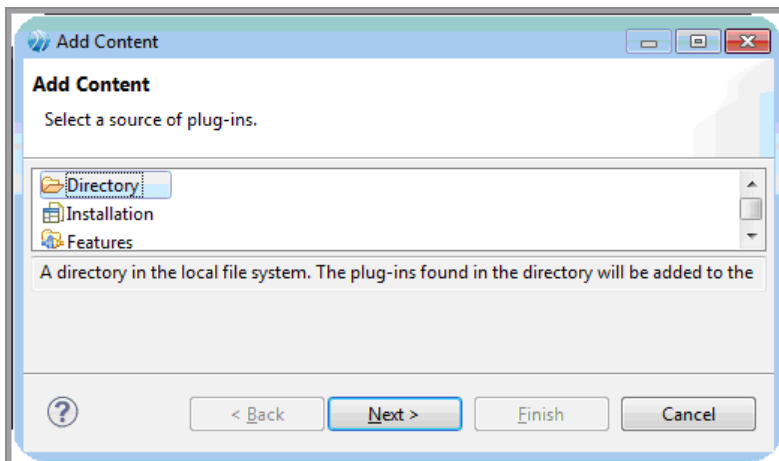
- Set up Eclipse for plug-in development.
- For this tutorial, TIBCO Business Studio™ - BPM Edition is used as the editor but any other editor can be used for developing the policy action.

Procedure

1. Navigate to **Window > Preferences > Plug-in Development > Target Platform** and create a target with an empty target definition. Click **Next**.

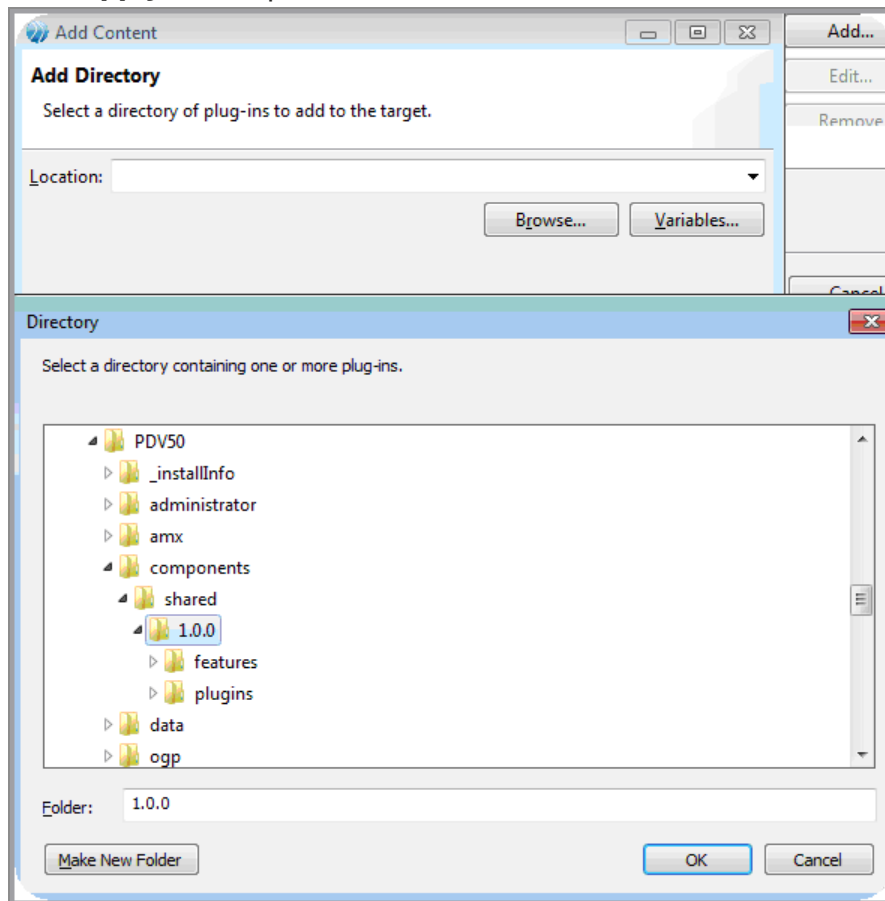


2. Select **Directory** as the source for plug-ins. Click **Next**.



3. Click **Browse** and locate TIBCO_HOME/components/shared/1.0.0 from the directory.

Click **Apply** to complete.



After the Target platform is set and applied, the editor is ready for developing the custom actions.

Implementing a Custom Action

To implement a working custom action, two classes must be implemented:

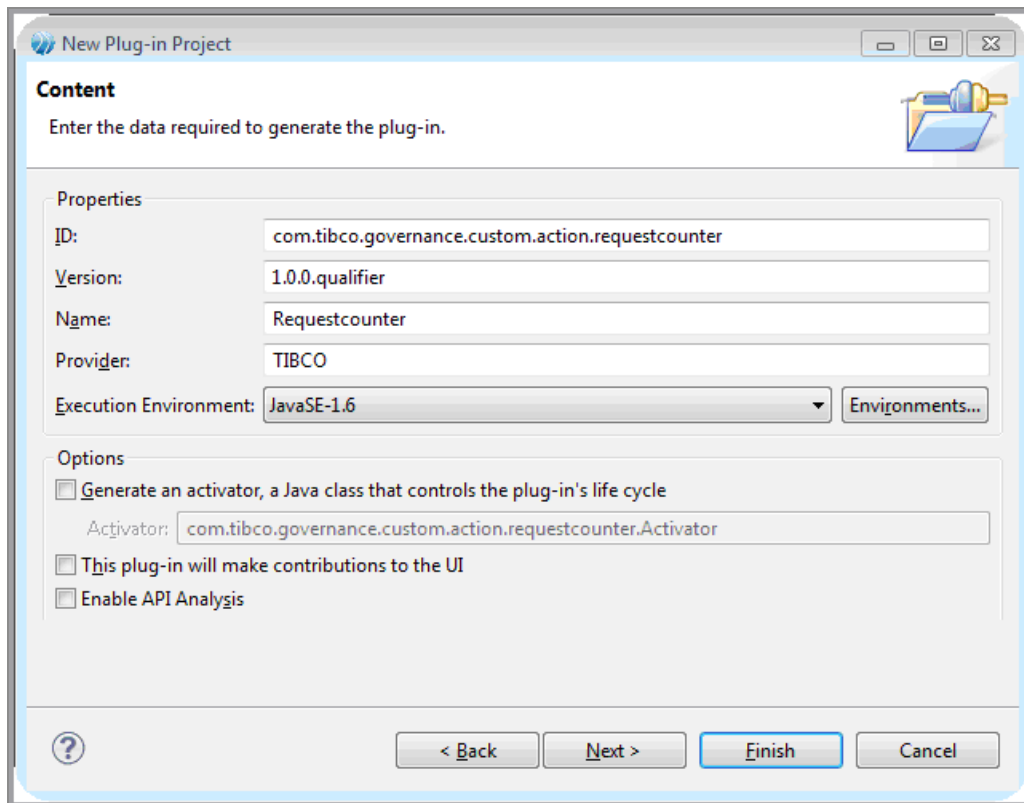
- `com.tibco.governance.agent.action.Action`
- `com.tibco.governance.agent.action.ActionConfigurationProcessor`

The `com.tibco.governance.agent.action.Action` class instance is created and initialized by the configuration processor when an action is first deployed and started. It also accepts an action context parameter by a `run` method. This `run` method is called each time the policy is invoked. This method may implement the complete execution logic.

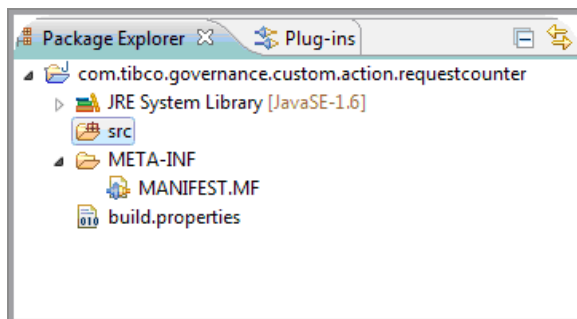
To create a custom action, create a new **Java-plugin** project first.

Procedure

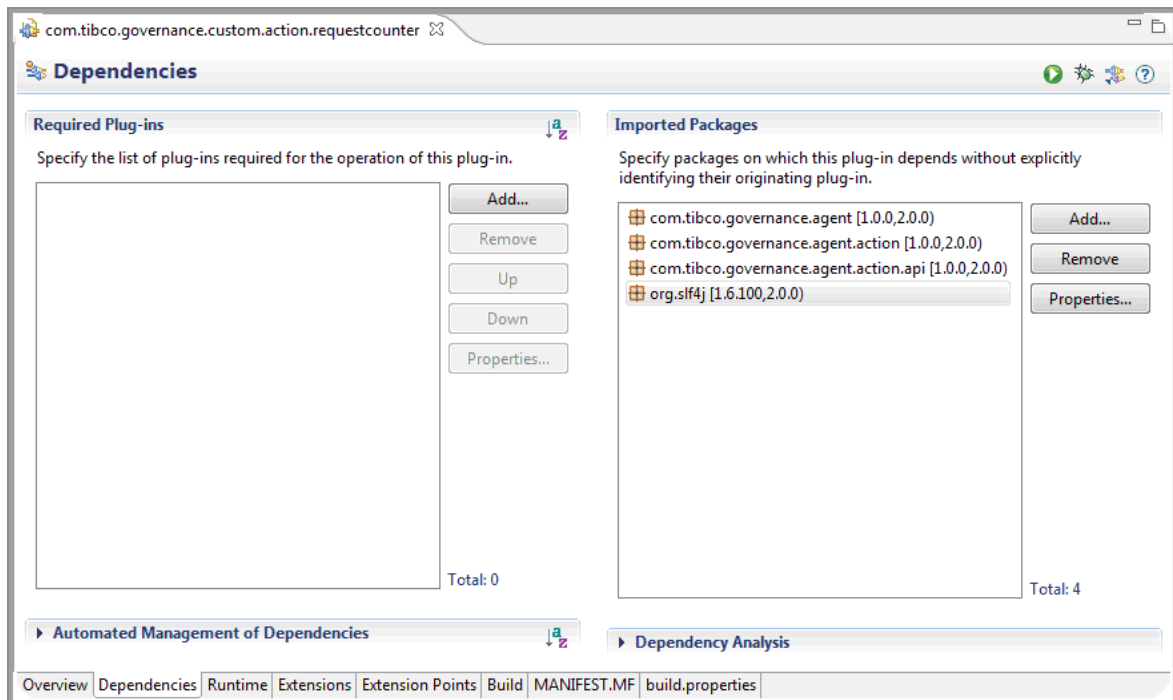
1. Clear the **Generate an activator, a Java class that controls the plug-in's life cycle** and **This plug-in will make contributions to the UI** checkboxes. Click **Finish**.



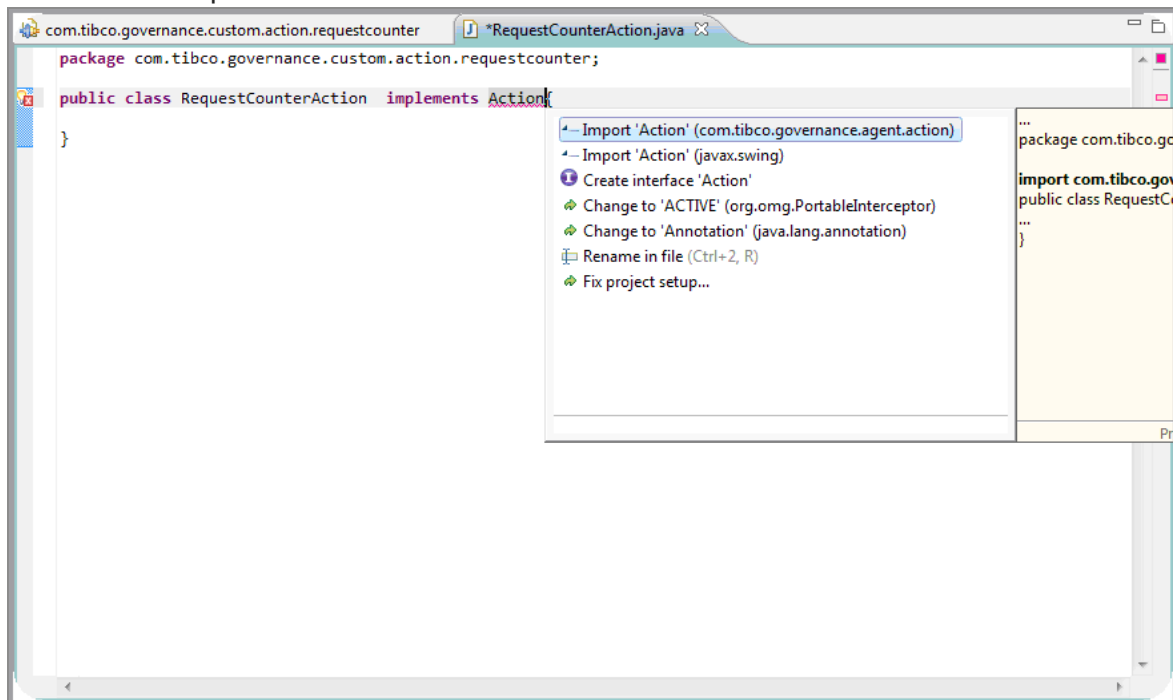
An empty plug-in project is created. **MANIFEST.MF** is created in the **META-INF** folder.



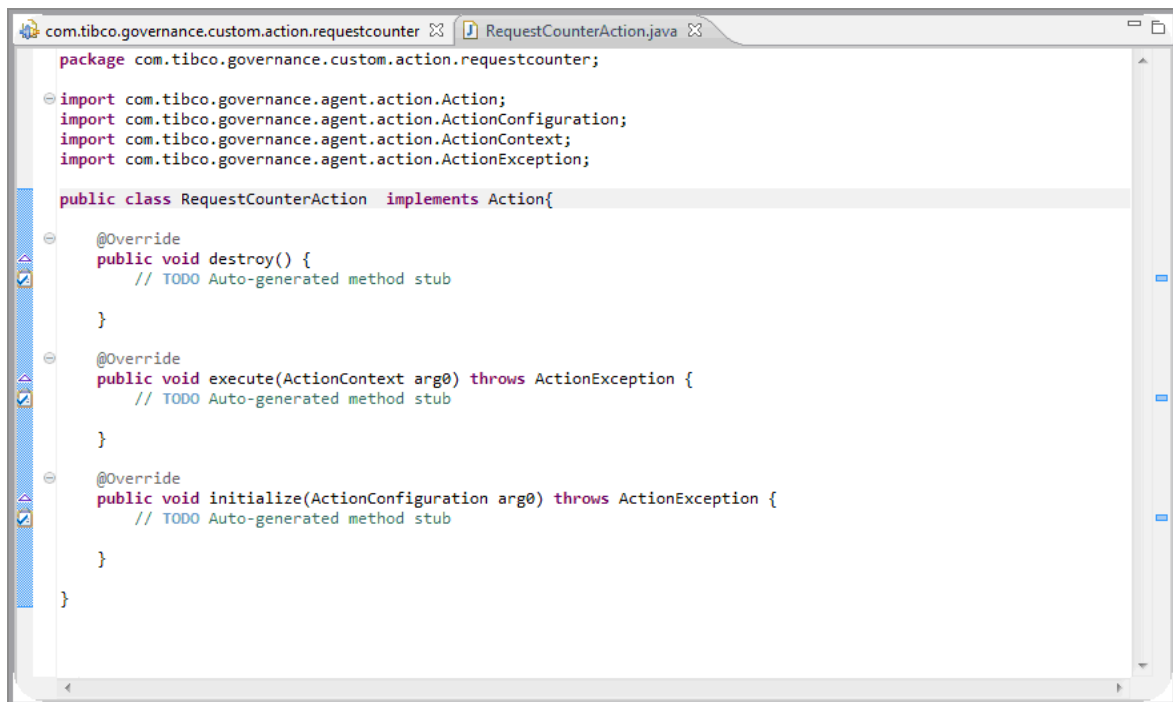
2. Open the **MANIFEST.MF** file. Under **Imported Packages**, click **Add**. Add the `com.tibco.governance.agent`, `com.tibco.governance.agent.action`, and `com.tibco.governance.agent.action.api` packages. **Save** the file.



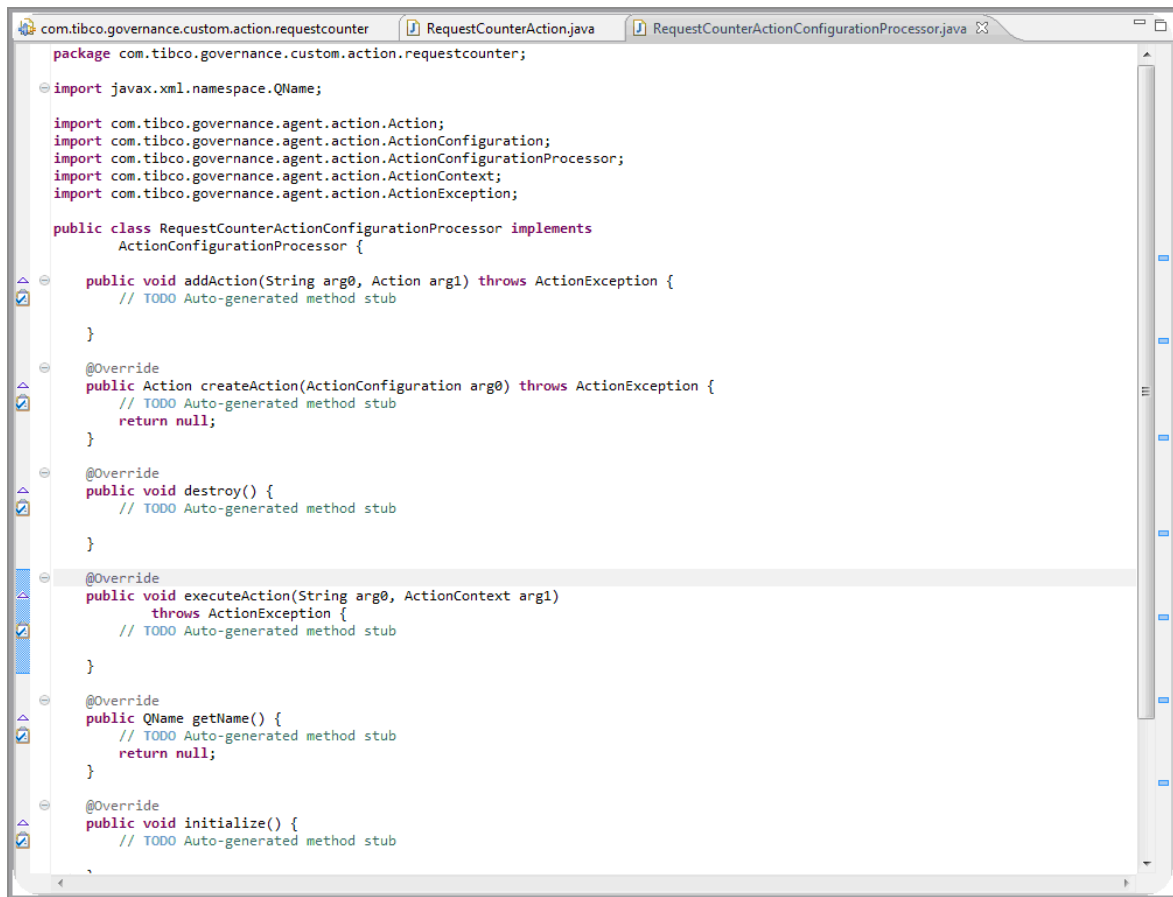
3. Create a **CustomAction** class to implement the `com.tibco.governance.agent.action.Action` class. Edit the class to implement the Action Interface and import the package `com.tibco.governance.agent.action`. Also, add the not implemented methods.



The class is created as shown:



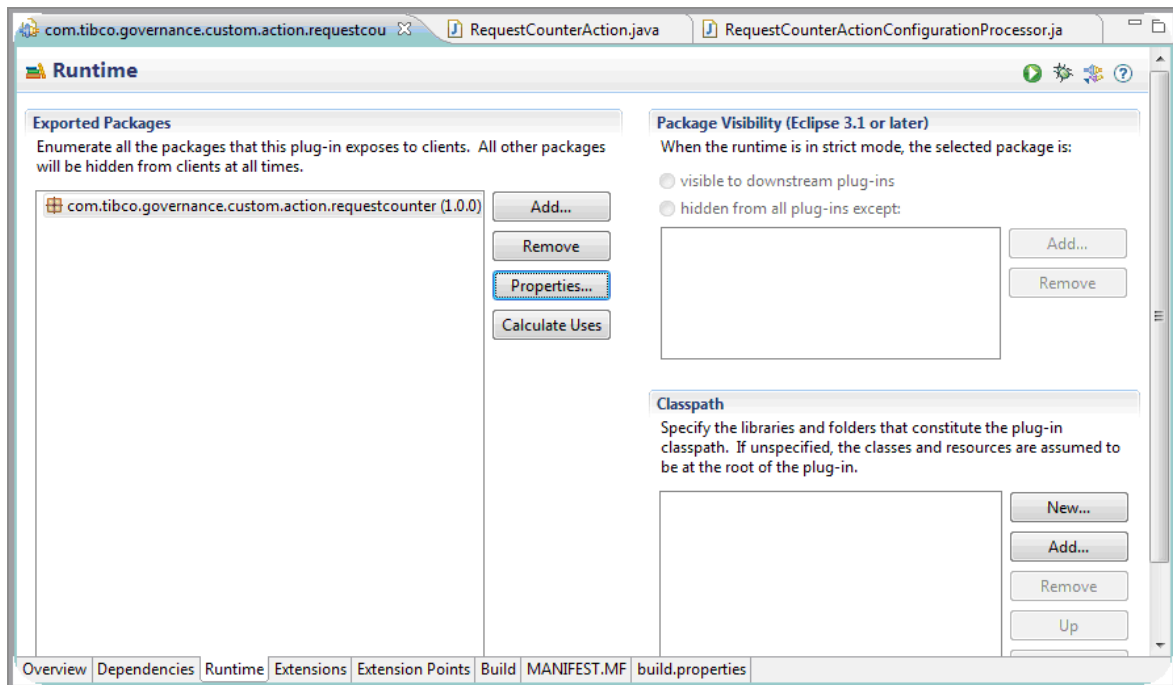
4. In addition to the action class, create CustomActionConfigurationProcessor which implements the com.tibco.governance.agent.action.ActionConfigurationProcessor. This class invokes and initializes the action. ActionConfigurationProcessor is the action manager and it initializes the action class.



5. After these two classes are created, add the business logic.

For more details on the location of the sample Request Counter, see Request Counter in [Sample Action Source Code](#).

6. After the business logic has been added, ensure that the custom action package is exported into **MANIFEST.MF**.



The action can now be packaged into an ActiveMatrix composite (to be deployed on ActiveMatrix).

Packaging the Action for ActiveMatrix® Service Grid

To deploy the action on ActiveMatrix Service Grid, it must be wrapped as the ActiveMatrix Service Grid component. A DAA is generated from the ActiveMatrix Service Grid composite. Which is then deployed on ActiveMatrix Service Grid.

For the ActiveMatrix Service Grid composite to identify that it is an ActiveMatrix Service Grid extension:

1. Open the composite file using a text editor.
2. Add the following **scaext: extension** tags to the composite file.

```
<scaext:extension xmi:id="uniqueId" name="amx component name"
requiredVersion="1.0.0"
extensionPoint="com.tibco.governance.agent.amxcomponent.extensionpoint.actionconfigurationprocessor"/>
```

Modify **sca:composite** to include

```
xmlns:scaext="http://xsd.tns.tibco.com/amf/models/sca/extensions" .
```

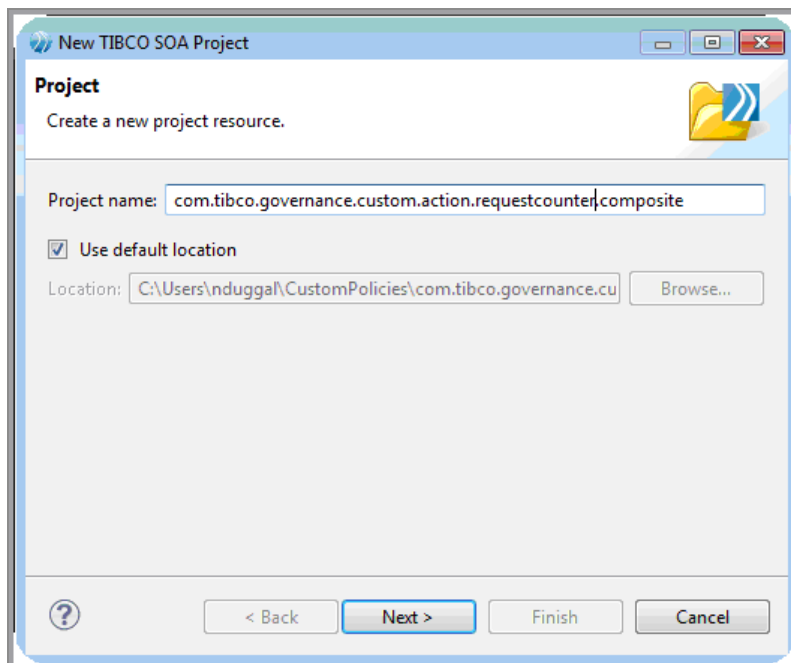
If the **scaext:extension** tag is specified, the ActiveMatrix Service Grid component only creates an instance of the Action Configuration Processor when deployed (if it does not exist).

For creating the ActiveMatrix Service Grid composite wrapper, see [Creating an ActiveMatrix Service Grid Composite Wrapper for a Custom Action](#).

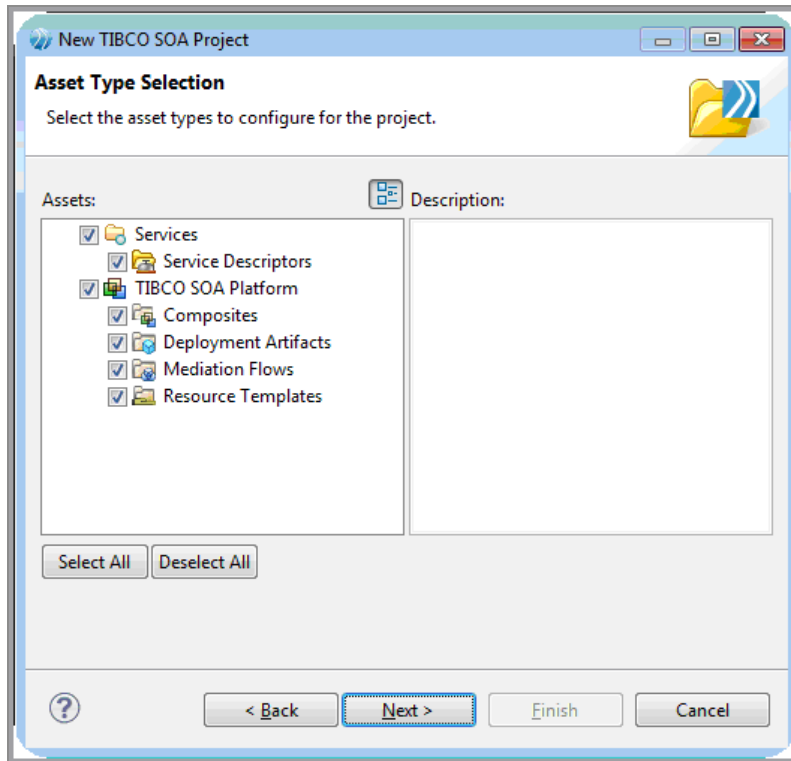
Creating an ActiveMatrix Service Grid Composite Wrapper for a Custom Action

Procedure

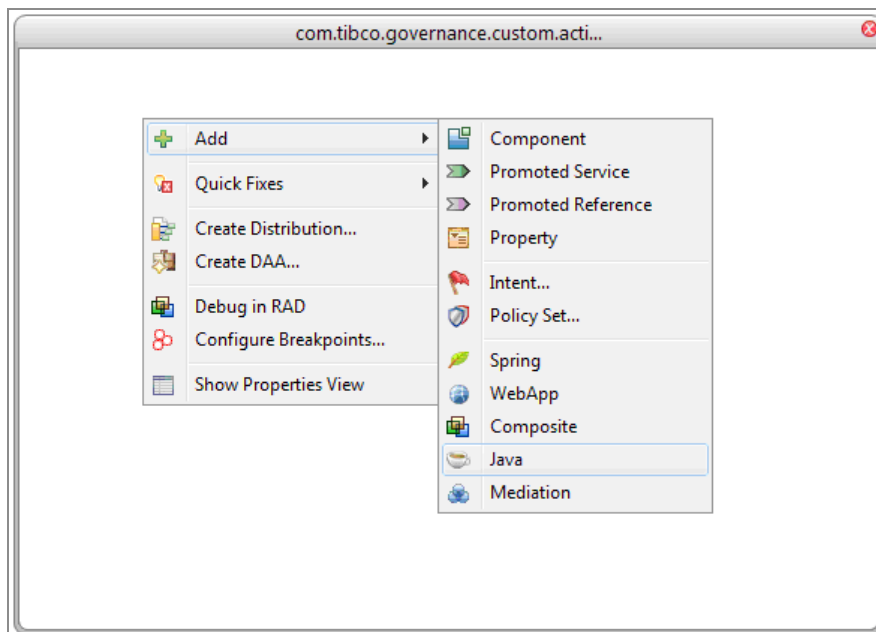
1. Create an SOA project using TIBCO Business Studio - BPM Edition.



2. Ensure all the options are selected and click **Next**.



3. Create an empty SOA project and click **Finish**. This creates an empty composite. Add a **Java** component to the composite.



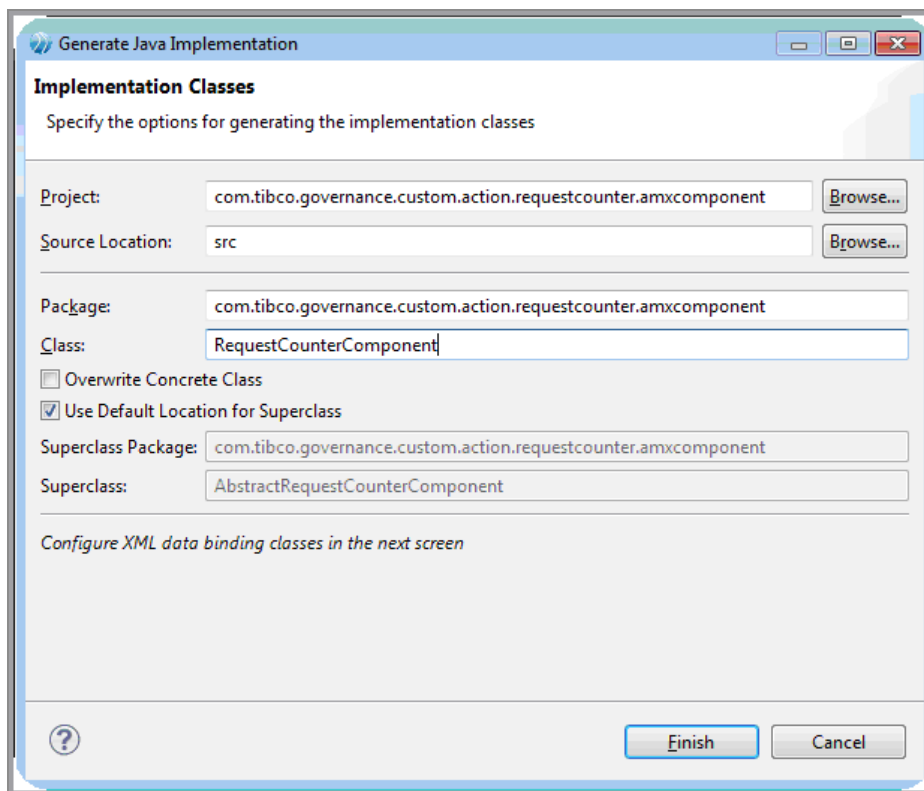
4. Open the **composite file** in a text editor and add the given tags after substituting the

customA action name in to the tags:

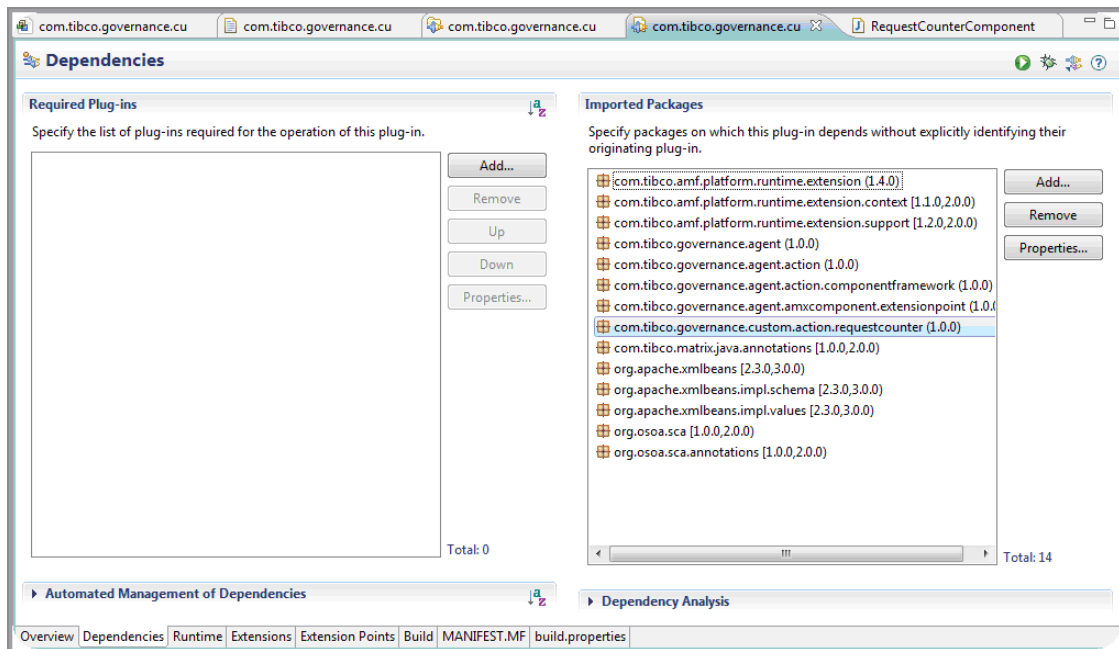
```
<scaext:extension xmi:id="_wefwedADASDsada"
name="com.tibco.governance.custom.action.customAction.amxcomponent.CustomActionComponent" requiredVersion="1.0.0"
extensionPoint="com.tibco.governance.agent.amxcomponent.extensionpoint.actionconfigurationprocessor"/>
```

Also edit the **sca:composite** tab to define the scaext namespace. For example:
xmlns:scaext=http://xsd.tns.tibco.com/amf/models/sca/extensions

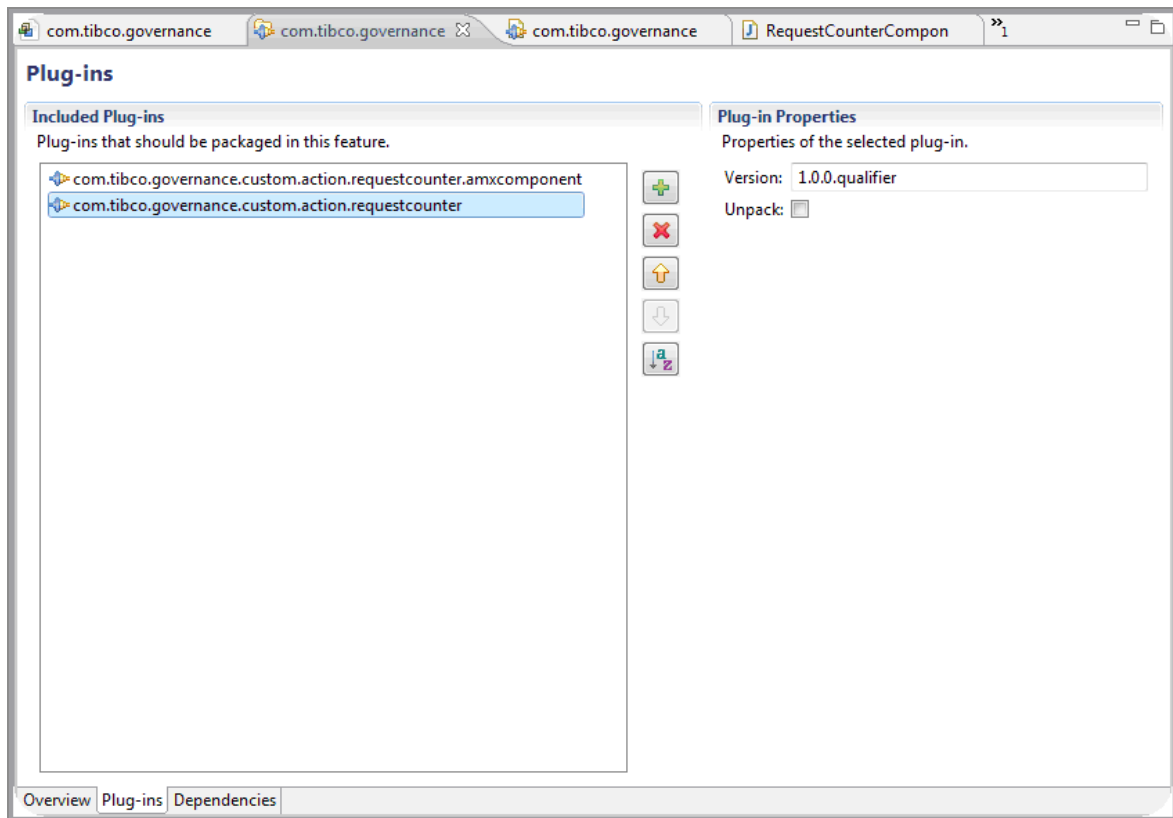
- Right-click the **Java component** and generate the **Implementation**. Enter the name of the project and click **Finish**. This generates two classes: AbstractCustomActionComponent and CustomActionComponent.



- Navigate to **MANIFEST.MF** of the generated implementation and ensure that the packages are added. If not, manually add the packages.



7. Once the setup is complete, in the CustomActionComponent class, import the class `com.tibco.amf.platform.runtime.extension.Extension` and add the **@Extension** notation at the top of the class declaration.
8. Import `ActionConfigurationProcessor`, `CustomActionConfigurationProcessor`, and `ActionConfigurationProcessorExtensionPoint` (`com.tibco.governance.agent.action.ActionConfigurationProcessor`). This component class must implement the `ActionConfigurationProcessorExtensionPoint` (`com.tibco.governance.agent.amxcomponent.extensionpoint.ActionConfigurationProcessorExtensionPoint`) interface. From the `init()` method of the component class, call `initialize` of the `CustomActionConfigurationProcessor`.
9. Navigate to the **customfeature** file under the `com.tibco.governance.custom.action.customaction/DeploymentArtifacts` and ensure that both the **amxcomponent** plug-ins and the **custom action** plug-in are included in the Included Plug-ins tab.



10. Generate the DAA. The DAA containing the action can now be deployed on ActiveMatrix Service Grid.

For a completely implemented ActiveMatrix Service Grid component, see Request Counter in [Sample Action Source Code](#).

Policy Templates

Two types of templates need to be created for any action: Action template and Rule template. An additional `Action.xsd` schema file must be created. `Action.xsd` is the schema for all the parameters that are expected by the policy. For more information on where these templates and `Action.xsd` files are to be copied, see [Deploying Policy Templates](#).

Both policy templates consist of two parts: parameters and configuration.

Parameters take the user inputs, which are then used by the configuration to formulate policy logic.

The configuration portion of the action template consists of:

- Action configuration fragment
- Allowed enforcement point placement information

The configuration portion of the rule template consists of:

- Exact enforcement point information
- The types of objects on which the policy can be applied
- A mapping of the parameters of the actions contained in the rule to the parameters defined in the rule template

The main difference between the action template and rule templates is that the action template defines all allowed values of `ActionExecution` settings and the rule template defines the one to be used by a particular policy.

For further information on the parameters and the allowed tags, see [Parameters and Action Templates](#).

Creating an Action Template

This is the first template that needs to be created. It consists of: Parameters, Action Configuration Fragment, and Enforcement Point Placement.

To create the action template, use the CustomAction.atp file from [Sample Action Rule Templates and User Interface Files](#).

Procedure

1. Copy the contents of the CustomAction.atp file and rename the file with the custom action name. For example, in the sample custom action Request Counter, we have named the action template file RequestCounter.atp.
2. Change **Qname** of the action **version**, specify the **category** and the **subcategory**:

```
<!--Details on the rule template: QName, version, category and subcategory of the action-->
<ogsei_base:qName>atp:CustomActionAction</ogsei_base:qName>
<ogsei_base:version>1.0.0</ogsei_base:version>
<ogsei_base:category>Custom</ogsei_base:category>
<ogsei_base:subcategory>Custom</ogsei_base:subcategory>
```

3. Specify the parameters of the action. These parameters are defined inside a parameter group.

```
<!--Parameters that are used to define the policy. Parameters can be grouped into parameter groups-->
<ogsei_base:parameterGroups>
  <ogsei_base:parameterGroup>
    <ogsei_base:name>$PARAMETERGROUPNAME</ogsei_base:name>
    <ogsei_base:parameters>
      <!--Define parameters: name, display name that is used by the UI, description, type and
      specify if its hidden or not-->
      <ogsei_base:parameter>
        <ogsei_base:name>$INPUT1</ogsei_base:name>
        <ogsei_base:displayName>$INPUT 1 DISPLAY NAME</ogsei_base:displayName>
        <ogsei_base:description>$FIRST INPUT</ogsei_base:description>
        <ogsei_base:hidden>false/true</ogsei_base:hidden>
        <ogsei_base:type>$TYPE_OF_INPUT</ogsei_base:type>
      </ogsei_base:parameter>
    </ogsei_base:parameters>
  </ogsei_base:parameterGroup>
</ogsei_base:parameterGroups>
```

- a. Change '**\$PARAMETERGROUPNAME**' to the name of the parameter group name.
- b. Change '**\$INPUT1**' to the name of the first input parameter.
- c. Change the '**\$INPUT 1 DISPLAY NAME**' to the name specified in the previous step or to any alternate name that should be displayed to the user.
- d. Change the '**\$FIRST INPUT**' to enter the description of the parameter that is being defined.
- e. Set hidden to true/false. If set to true, the parameter is not visible to the user on the UI, if set to false it is visible on the UI.
- f. Change '**\$TYPE_OF_INPUT**' to the type of the input parameter.

For a list of supported types, see the [Parameters and Action Templates](#).

The parameter tags can be repeated to add additional parameters that are needed to define the action.

4. Change the '**CustomActionAction**' to the name of the custom action. Change the **description**, the **implementation Id** and the **Intents**.

The implementation Id should be the same as the action qname specified in the action configuration processor.

```
<ogsei:name>CustomActionAction</ogsei:name>
<ogsei:description>Action template for Custom Action</ogsei:description>
<ogsei:createdBy>$CREATEDBY</ogsei:createdBy>
<ogsei:createdOn>TIMESTAMP<!--FORMAT OF TIMESTAMP 2012-12-12T12:12:12--></ogsei:createdOn>
<ogsei:implementationId>tpa:CustomAction</ogsei:implementationId>
<ogsei:providedIntents>
  <ogsei:intent>scaext:CustomActionIntent</ogsei:intent>
</ogsei:providedIntents>
<ogsei:specVersion>1.0.0</ogsei:specVersion>
<ogsei:implVersion>1.0.0</ogsei:implVersion>
<ogsei:builtIn>>false</ogsei:builtIn>
```

5. Change **config fragment** as expected by the action. configFragment is interpreted by the Velocity Engine at run time to produce the XML fragment that is used to initialize the action. The **\$INPUT1** and **\$INPUT2** are the two inputs, which are passed in a format that can be interpreted by the velocity engine.

```
<!--Configuration fragment that is delivered to the Policy Agent -->
<ogsei:configFragment><![CDATA[
  <tpa:customAction xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2012"
    input1="$($INPUT1)" input2="$($INPUT2)">
  </tpa:customAction>
]]></ogsei:configFragment>
```

6. Change the '**\$Path_To_CustomAction.xsd**' to point to the path of the Custom Action schema. 'useWSPolicySchema' can be set to true if the custom policy adheres to the WS Policy schema.

```
<ogsei:useWSPolicySchema>>false</ogsei:useWSPolicySchema>
<!--Do not change-->
<ogsei:resourcePath>factory</ogsei:resourcePath>
<!--Path to the Custom Action Schema-->
<ogsei:schemaResourcePath>$Path_To_CustomAction.xsd</ogsei:schemaResourcePath>
```

7. Change the **\$HOST**. The host can be a pipeline or host. **\$ENFORCEMENTPOINT**, **\$STAGE**, and **\$INTERVAL** need to be changed depending on where the action needs to be invoked.

```

<!--Specify allowed enforcement points for the action-->
<ogsei:allowedEnforcementPlacements>
  <ogsei:allowedEnforcementPlacement>
    <!--Define the host where policy enforcement should take place-->
    <ogsei:host>govhost:$HOST</ogsei:host>
    <ogsei:sectionPlacement>
      <!--DO NOT CHANGE THE SECTION |-->
      <ogsei:section>default</ogsei:section>
      <ogsei:enforcementPointPlacement>
        <!--Define the point of enforcement-->
        <ogsei:enforcementPoint>$ENFORCEMENTPOINT</ogsei:enforcementPoint>
        <ogsei:stagePlacement>
          <!--Define the stage placement-->
          <ogsei:stage>$STAGE</ogsei:stage>
          <ogsei:interval>$INTERVAL</ogsei:interval>
          </ogsei:stagePlacement>
        </ogsei:enforcementPointPlacement>
      </ogsei:sectionPlacement>
    </ogsei:allowedEnforcementPlacement>
  </ogsei:allowedEnforcementPlacements>
  <!--Define the merge placements and the merge cardinality-->
  <ogsei:mergePlacement>$MERGEPLACEMENT</ogsei:mergePlacement>
  <ogsei:mergeCardinality>$MERGECARDINALITY</ogsei:mergeCardinality>

```

For a complete set of allowed host, section, enforcement points, stage, and interval, see the [Sample Action Source Code](#).

The last two tags specify the merge placement and cardinality.

AllowedEnforcementPlacements define all the allowed locations where this action could be used.

The actual place where this is used is determined in the Rule Template by using this Action Template. The mergePlacement and mergeCardinality are defined only in the action template. The Rule template cannot define the placement or cardinality of the action it is using.

On completion, the custom action template is ready to be used. For a complete action template refer to the following:

CustomPolicies/samples/requestCounter/template/RequestCounter.atp.

The CustomPolicies folder is at:

TIBCO_HOME/pd/1.2/samples/CustomPolicies

Creating a Rule Template

Rule templates can contain one or more actions.

To create a rule template, use the CustomAction.rtp file from [Sample Action Rule Templates and User Interface Files](#).

Procedure

1. Replace the string **CustomAction** with the actual custom action name and modify the version, category, and subcategory as required.

```
<!--Details on the rule template: QName, version, category and subcategory of the rule-->
<ogsei_base:qName>rtp:CustomActionRule</ogsei_base:qName>
<ogsei_base:version>1.0.0</ogsei_base:version>
<ogsei_base:category>Custom</ogsei_base:category>
<ogsei_base:subcategory>Custom</ogsei_base:subcategory>
```

Parameters : Similar to action template. Action template requires the rule template to instantiate the policy fragment (unless some parameters are hidden and have a default value in the ActionTemplate). All parameters which the action template use in the policy fragment should have a value set or the velocity engine produces an exception.

2. **Enforcement Point Placement:** Unlike the action template, which specifies the allowed enforcement point placement, the rule template specifies the exact enforcement point placement for each action referred to in the rule template.

```
<ogsei:productScope>PolicyDirector</ogsei:productScope>
<ogsei:eca xsi:type="ogsei_per:ActionsOnlyECA">
  <ogsei_per:actions>
    <ogsei:governanceRuleAction>
      <ogsei:template>atp:CustomActionAction</ogsei:template>
      <ogsei:templateVersion>1.0.0</ogsei:templateVersion>
      <ogsei:paramGroupSettings>
        <!--Each action referred in the rule can be configured using -->
        <ogsei_base:parameterGroupSetting>
          <ogsei_base:paramGroupName>$CustomActionPARAMETERGROUPNAME</ogsei_base:paramGroupName>
          <ogsei_base:paramSettings>
            <ogsei_base:paramSetting>
              <!--Parameter setting has to be provided for each parameter that exists for each action-->
              <ogsei_base:singleParamSetting>
                <!--Rule template parameter $INPUT1-->
                <ogsei_base:paramName>$INPUT1</ogsei_base:paramName>
                <!--Action template parameter $INPUT1-->
                <ogsei_base:refParentParamName>$INPUT1</ogsei_base:refParentParamName>
              </ogsei_base:singleParamSetting>
            </ogsei_base:paramSetting>
          </ogsei_base:paramSettings>
        </ogsei_base:parameterGroupSetting>
      </ogsei:paramGroupSettings>
      <ogsei:actionExecutions>
        <!--Provide the exact execution location for the action-->
        <ogsei:actionExecution>
          <ogsei:host>govhost:$HOST</ogsei:host>
          <ogsei:section>default</ogsei:section>
          <ogsei:enforcementPoint>$ENFORCEMENTPOINT</ogsei:enforcementPoint>
          <ogsei:stage>$STAGE</ogsei:stage>
          <ogsei:interval>$INTERVAL</ogsei:interval>
        </ogsei:actionExecution>
      </ogsei:actionExecutions>
    </ogsei:governanceRuleAction>
  </ogsei_per:actions>
</ogsei:eca>
```

A rule template can contain multiple action templates. These actions are configured

using the `parameterSetting` tags. There should be a `groupParameterSetting` for each action that is contained in the rule template. And for each action, a relationship between the rule template parameter and the action template parameter is typically defined using the `singleParameterSetting` tag as shown in the figure. It is desirable to have the same parameter and parameter group names in the action and rule templates.

3. **Target Object Group information:** In addition to the parameter group related to the action, the rule template has another parameter group that is used to define the type of Object Groups the custom policy can be applied to. The parameter group `Default` should not be changed. The valid object types can be specified by replacing **\$TYPEOF OBJECT** in the template.

```
<!--Specify the type of objects that the Request counter policy can be applied to-->
<ogsei:togSelectors>
  <ogsei_base:targetObjectGroupSelector>
    <ogsei_base:leafObjectType>amx3:$TYPEOF OBJECT</ogsei_base:leafObjectType>
  </ogsei_base:targetObjectGroupSelector>
</ogsei:togSelectors>
```

For a list of all valid object types, see [Parameters and Action Templates](#).

4. **UI handle:** The Rule template additionally has three tags that serve as the UI handle.

```
<!--Specify the relative path for UI. If the hidden parameter is set to true the policy
template is not available in the policy wizard-->
<ogsei_base:uiResourcePath>pd/templates/customAction</ogsei_base:uiResourcePath>
<ogsei_base:uiHandlerType>default</ogsei_base:uiHandlerType>
<ogsei_base:hidden>false</ogsei_base:hidden>
```

Change the `uiResourcePath` variable to the relative path of the UI resource. The `uiHandlerType` is set to default for UIs based on GI. The hidden parameter determines if the template is visible to the user or not. If set to `false`, the template is not visible in the policy creation wizard.

5. Finally, it is a good practice to elevate the version of the template to a higher number (`ogsei: templateVersion` element value)

After these changes are made, the rule template is ready for use. For the complete rule template, see [Sample Action Rule Templates and User Interface Files](#).

User Interface

Currently, the user interfaces are manually created by the General Interface (GI). The folder structure consists of `Action.js`, `Descriptor.xml`, `Controller.js`, `UI- action.xml`, and `JSS- locales.xml`.

**Note:**

Copy the files from the given sample to a corresponding folder for custom action:

`CustomPolicies\samples\requestCounter\ui\requestCounter.`

Rename the `requestCounter` folder to the name of the custom action and change the files where applicable, as per comments in the template UI files.

- `Action.js`: Reads the user input from the wizard and returns the populated `paramGroup`.
- `Descriptor.xml`: Specifies the controller class needed for the UI. It also specifies the UI form file, the `action.js` file, and the locale file.
- `Controller.js`: Allows custom code to talk to the wizard manager. It is an implementation of the `IDynamicForms` interface.
- `ui` (folder)
 - `action.xml`: This is the actual GI form for the UI.
- `jss` (folder)
 - `locales.xml`: This file contains externalized strings for localization.

Deploying the Custom Policy

To deploy a custom policy, you must do the following:

1. Deploy the policy templates
2. Deploy the user interface
3. Restart the system node
4. Deploy the action on ActiveMatrix Service Grid

Deploying Policy Templates

Procedure

1. Copy the action template into the CONFIG_
HOME/admin/amxadmin/shared/repo/governance/templates/actions/custom folder.
Create it if it does not exist.
2. Copy the rule template file into the CONFIG_
HOME/admin/amxadmin/shared/repo/governance/templates/rules/custom folder.
Create it if it does not exist.
3. Copy the action schema file to CONFIG_
HOME/admin/amxadmin/shared/repo/governance/actionschema/custom folder.

Deploying User Interface

Copy the **UI** folder to the Administration repository at the following location (create the folder hierarchy if it does not exist):

CONFIG_
HOME/admin/amxadmin/shared/repo/governance/ui/rules/pd/templates/\$YourCustomActionName

For example, for a request counter, copy:

CustomPolicies\samples\requestCounter\ui\requestCounter

To:

CONFIG_
HOME/admin/amxadmin/shared/repo/governance/ui/rules/pd/templates/requestCounter

Restarting the System Node

After you deploy the Policy Templates and UI, restart **SystemNode** for the changes to take effect.

Deploying the Action On ActiveMatrix Service Grid

Deploy the DAA that was created on the ActiveMatrix Service Grid Policy Director Governance Administrator server.

For a completely implemented ActiveMatrix Service Grid component, see Request Counter Sample in [Sample Action Source Code](#).

Debugging and Logging

You can add a debug point in the custom action and remotely debug the action using Eclipse or TIBCO Business Studio - BPM Edition.

Debugging and Logging ActiveMatrix Service Grid Environment

Procedure

1. Navigate to the node directory on which the action is deployed and add the given line to the `DeployedNode.tra` file. For example, the custom action is deployed on the `DevNode`, so add the given line to the file `~/confighome/tibcohost/Admin-amxadmin-instanceOne/data_3.2.x/nodes/DevNode/bin/DevNode.tra`

```
java.extended.properties=-Xmx512m -Xms128m -XX:MaxPermSize=192m -  
XX:+HeapDumpOnOutOfMemoryError  
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdwp:transport=dt_socket,server=y,address=5005,suspend=y
```

2. Add a debug point on the `execute()` method in the custom action and remotely debug the action using Eclipse or TIBCO Business Studio - BPM Edition by connecting to the port specified.

Troubleshooting

Issue	Resolution
Policy does not appear in the governance control wizard template even after the template is copied and tibcohost is restarted.	Check whether the hidden parameter in the rule template is set to <code>false</code> . If it is set to <code>true</code> , change it to <code>false</code> and restart tibcohost .
<ul style="list-style-type: none">• Errors with the UI.• The UI does not load and a blank screen appears when you select the policy template in the governance-control creating wizard.	<ol style="list-style-type: none">1. Check the <code>descriptor.xml</code> UI file and ensure that the values for the <code>component</code> and <code>controller.js</code> files match the actual file.2. Ensure that the rule template specified is correct.3. Use Firebug or any other client debugger to get detailed errors and debug the client code.

Sample Action Source Code

Sample Custom Action

```
CustomPolicies/TemplateCustomActionProject/action/src/com/tibco/custom/action/customAction/CustomAction.jav
```

```
CustomPolicies/TemplateCustomActionProject/action/src/com/tibco/custom/action/customAction/CustomActionConfigurationProcessor.java
```

Request Counter

```
CustomPolicies/samples/requestCounter/action/com.tibco.governance.custom.action.requestcounter/src/com/tibco/governance/custom/action/requestcounter/RequestCounterAction.java
```

```
CustomPolicies/samples/requestCounter/action/com.tibco.governance.custom.action.requestcounter/src/com/tibco/governance/custom/action/requestcounter/RequestCounterActionConfigurationProcessor.java
```

Hack Alert Action

```
CustomPolicies/samples/hackAlert/action/com.tibco.governance.custom.action.hackalert/src/com/tibco/governance/custom/action/hackalert/HackAlertAction.java
```

```
CustomPolicies/samples/hackAlert/action/com.tibco.governance.custom.action.hackalert/src/com/tibco/governance/custom/action/hackalert/HackAlertActionConfigurationProcessor.java
```

Parameters and Action Templates

Parameters

Parameters are governed by the Rule Template schemas located at:

```
CustomPolicies/schemas/RuleTemplate_base.xsd  
CustomPolicies/schemas/RuleTemplate.xsd
```

- Parameters extract the user-configurable portion out of the configuration. Also, it carries the information needed for the UI to complete the job of collecting a valid value for the parameter.
- A parameter has an attribute type that can be used to indicate the type of the parameter. It can be used to trigger some special processing on the parameter when needed. The type could also be used for value validation. Parameter values are always stored as a string. It is up to the validation method to validate the value. The following is a list of all supported types:
 - INTEGER
 - STRING
 - COMPLEX
 - DOUBLE
 - BOOLEAN
 - RESOURCE_INSTANCE
 - OBJECT_GROUP
- Parameters could have default values. Parameters could be configured as hidden. For a hidden parameter, the default value must exist and be used as the value of the parameter.
- The parameter can be set to optional. If a parameter is not optional (the default is `false`), its value must be set when the instance is created.
- Parameter can have `valueChoices` and if it does, the parameter value must be set with a value in the `valueChoices` list unless the flag "openChoice" is set on the

parameter. For this type of 'choice' parameter, if it is unbounded it is multiple choice and otherwise, it is single choice. These kinds of parameters are presented on the UI as checkboxes or pull-down menu. Each value choice for a parameter has an optional 'label' that can be used by the UI to be presented to the users instead of the actual parameter value if the value is too internal. If the label is not set, then the value is used.

- A parameter value type could be COMPLEX where the parameter value must be a list of "PropertyVal". Each PropertyVal has a name, type, and optional shared resource association, similar to a regular parameter.
- Parameters can be organized in groups. Parameters in a group could have values collected together and can be presented on the UI, for example, as a feature tab. By default, each template has a 'Default' parameter group. Normally, it is used for holding general configuration parameters. However, if a template does not need to separate parameters into multiple groups, all the parameters can be contained in the "Default" group.

Valid object groups can be created with object of types:

- RestHttpServiceBindingInstance
- RestHttpReferenceBindingInstance
- SoapHttpServiceBindingInstance
- WebAppComponentInstance
- SoapServiceBindingInstance
- SoapJmsServiceBindingInstance
- SoapHttpReferenceBindingInstance
- SoapReferenceBindingInstance
- VirtualizationServiceBindingInstance
- VirtualizationReferenceBindingInstance

Action Fragment

The action template could have a "template for transformation" stored in "configFragment". The configFragment is an XML fragment with some transformation directives in it, based on parameters. A typical transformation includes substituting substitution variables or optionally ruling out some fragments based on a parameter value.

Policy Enforcement Placement

The action template defines policy enforcement placements, which inform the agent on where the policies are to be enforced. A policy can be applied at various stages and phases.

For more information, see *TIBCO ActiveMatrix® Service Grid Concepts*.

All the currently supported Policy Enforcement Points are listed in:

CustomPolicies/PolicyConfigurationData.xlsx

The CustomPolicies folder is located at:

TIBCO_HOME/pd/1.2/samples/CustomPolicies

Sample Action Rule Templates and User Interface Files

**Note:**

The CustomPolicies folder is located at:

TIBCO_HOME/pd/1.2/samples/CustomPolicies

Action Template (atp) files

Refer to any of the sample action template files located:

```
CustomPolicies/TemplateCustomActionProject/templates/CustomAction.atp  
CustomPolicies/samples/requestCounter/templates/RequestCounter.atp  
CustomPolicies/samples/hackAlert/templates/HackAlert.atp
```

Rule template (rtp) files

Refer to any of the sample rule template files located:

```
CustomPolicies/TemplateCustomActionProject/templates/CustomAction.rtp  
CustomPolicies/samples/requestCounter/templates/RequestCounter.rtp  
CustomPolicies/samples/hackAlert/templates/HackAlert.rtp
```

Controller Javascript files

Refer to any of the sample controller javascripts files located:

```
CustomPolicies/TemplateCustomActionProject/ui/controller.js  
CustomPolicies/samples/requestCounter/ui/requestCounter/controller.js  
CustomPolicies/samples/hackAlert/ui/hackalert/controller.js
```

Action Javascript files

Refer to any of the sample javascript files that push the value of each user input and set it to the parameters of the corresponding rule template:

```
CustomPolicies/TemplateCustomActionProject/ui/customaction.js
CustomPolicies/samples/requestCounter/ui/requestCounter/requestcounter.js
CustomPolicies/samples/hackAlert/ui/hackalert/hackalert.js
```

Descriptor XML files

The sample descriptor XML files are located:

```
CustomPolicies/TemplateCustomActionProject/ui/descriptor.xml
CustomPolicies/samples/requestCounter/ui/requestCounter/descriptor.xml
CustomPolicies/samples/hackAlert/ui/hackalert/descriptor.xml
```

Action XML files

The sample action XML files to generate the UI are located:

```
CustomPolicies/TemplateCustomActionProject/ui/ui/customaction.xml
CustomPolicies/samples/requestCounter/ui/requestCounter/ui/requestcounter.xml
CustomPolicies/samples/hackAlert/ui/hackalert/ui/hackalert.xml
```

String externalization

The sample locale files for externalizing strings are located:

```
CustomPolicies/TemplateCustomActionProject/ui/jss/locales.xml
CustomPolicies/samples/requestCounter/ui/requestCounter/jss/locales.xml
CustomPolicies/samples/hackAlert/ui/hackalert/jss/locales.xml
```

To externalize strings using the `locales.xml` file, use the dynamics element of General Interface in your custom action XML.

Example from `requestcounter.xml`:

Remove the `fieldtitletext="Number of Requests"` attribute in the strings element:

```
<strings isrequired="1" hideoptionalstring="0" jsxname="textFieldNoOfRequests"
jsxtitledisplay="" validator="@isPositiveInteger" cdfattribute="NoOfRequests"
fieldtitletext="Number of Requests"/>
```

Replace it with the following strings and dynamics elements: `<strings isrequired="1" hideoptionalstring="0" jsxname=" textFieldNoOfRequests " jsxtitledisplay="" validator="@isPositiveInteger" cdfattribute="PollingTime"/> <dynamics fieldtitletext=" requestCounter@NoOfRequests" />` The `fieldtitletext` element points to the following entry of your `locales.xml` file: `<record jsxid="requestCounter@NoOfRequests" jsxttext="Number Of Requests" />`

Code Snippets

Using the Action Context Object

A good way to figure out the contents of the Action Context is to turn debugging on and put a break point in the execute method. For information on how to turn debugging on, see the [Debugging and Logging](#).

It is useful to inspect the contents of the following objects to know everything you have at your disposal:

- `_contextDocumentsByQName`
- `_contextObjectsByQName`
- `_contextPropertiesByQName`

Extracting the SOAP envelope in Custom Action's execute method

Here is a code snippet to extract and parse the message payload when your `action.execute()` method is invoked. The code snippet is written assuming that the payload is a soap envelope:

```
org.w3c.dom.Document envelope = (Document)( actionContext.getDocument
(ActionConstants.MESSAGE_ENVELOPE));
if (envelope != null) {
String envelopeString =
com.tibco.governance.agent.core.utils.DOMUtils.getInstance().getString
(envelope));
Element soapBody = DOMUtils.getFirstDescendantElementNS
(envelope.getDocumentElement(),
"http://schemas.xmlsoap.org/soap/envelope/", "Body");
String soapBodyString = DOMUtils.getInstance().getString(soapBody);
```

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the [TIBCO ActiveMatrix® Service Grid Product Documentation](#) page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature

requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix, Business Studio, Enterprise Message Service, and Hawk are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.tibco.com/patents>.

Copyright © 2010-2025. Cloud Software Group, Inc. All Rights Reserved.