



TIBCO ActiveMatrix® Service Grid

Policy Director Governance Custom Actions

*Software Release 3.4
April 2019*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

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 TIBCO Software Inc.

TIBCO, the TIBCO logo, Two-Second Advantage, TIB, Information Bus, ActiveMatrix, Business Studio, Enterprise Message Service, Hawk, and Rendezvous are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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.

This 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. Please see the readme.txt file for the availability of this software version 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. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

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 "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2010-2019. TIBCO Software Inc. All Rights Reserved.

Contents

TIBCO Documentation and Support Services	5
Introduction to Custom Actions	7
Setting up the Eclipse Environment for Plug-in Development	8
Implementing a Custom Action	10
Packaging the Action for ActiveMatrix	14
Creating an ActiveMatrix Composite Wrapper for a Custom Action	14
Policy Templates	19
Creating an Action Template	19
Creating a Rule Template	21
User Interface	24
Deploying the Custom Policy	25
Deploying Policy Templates	25
Deploying User Interface	25
Restarting the System Node	25
Deploying the Action on ActiveMatrix	25
Debugging and Logging	26
Debugging and Logging ActiveMatrix Environment	26
Troubleshooting	27
Sample Action Source Code	28
Parameters and Action Templates	29
Sample Action Rule Templates and User Interface Files	31
Code Snippets	33

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

Product-Specific Documentation

Documentation for TIBCO ActiveMatrix® Service Grid is available on the <https://docs.tibco.com/products/tibco-activematrix-service-grid> page.

Use of the following features, installation profiles and development tools requires a TIBCO ActiveMatrix Service Grid license:



- TIBCO ActiveMatrix Policy Director Governance, TIBCO ActiveMatrix SPM Dashboard, and TIBCO ActiveMatrix SPM Runtime Server profiles; and
- TIBCO ActiveMatrix Service Grid development tools for Java, Webapp and Spring components.

Customers with only a TIBCO ActiveMatrix Service Bus license are not licensed to use these features, tools or profiles.

The following documents form the documentation set:

- *TIBCO ActiveMatrix Service Grid Concepts*: Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of the platform. The other manuals in the documentation set assume you are familiar with the information in this manual.
- *TIBCO ActiveMatrix Service Grid Development Tutorials*: Read this manual for a step-by-step introduction to the process of creating, packaging, and running composites in TIBCO Business Studio.
- *TIBCO ActiveMatrix Service Grid Composite Development*: Read this manual to learn how to develop and package composites.
- *TIBCO ActiveMatrix Service Grid Java Component Development*: Read this manual to learn how to configure and implement Java components.
- *TIBCO ActiveMatrix Service Grid Mediation Component Development*: Read this manual to learn how to configure and implement Mediation components.
- *TIBCO ActiveMatrix Service Grid Mediation API Reference*: Read this manual to learn how to develop custom Mediation tasks.
- *TIBCO ActiveMatrix Service Grid Spring Component Development*: Read this manual to learn how to configure and implement Spring components.
- *TIBCO ActiveMatrix Service Grid WebApp Component Development*: Read this manual to learn how to configure and implement Web Application components.
- *TIBCO ActiveMatrix Service Grid REST Binding Development*: Read this manual to learn how to configure and implement REST components.
- *TIBCO ActiveMatrix Service Grid Administration Tutorials*: Read this manual for a step-by-step introduction to the process of creating and starting the runtime version of the product, starting TIBCO ActiveMatrix servers, and deploying applications to the runtime.
- *TIBCO ActiveMatrix Service Grid Administration*: Read this manual to learn how to manage the runtime and deploy and manage applications.

- *TIBCO ActiveMatrix Service Grid Hawk ActiveMatrix Plug-in*: Read this manual to learn about the Hawk plug-in and its optional configurations.
- *TIBCO ActiveMatrix Service Grid Policy Director Governance Custom Actions*: Read this manual to learn how you can configure and enforce policies for ActiveMatrix and external services hosted in third party containers, using TIBCO ActiveMatrix Policy Director Governance.
- *TIBCO ActiveMatrix Service Grid Service Performance Manager API Reference*: Read this manual to learn how to use the SPM APIs.
- *TIBCO ActiveMatrix Service Grid Error Codes*: Read this manual to know more about the error messages and how you could use them to troubleshoot a problem.
- *TIBCO ActiveMatrix Service Grid Installation and Configuration*: Read this manual to learn how to install and configure the software.
- *TIBCO ActiveMatrix Service Grid Security Guidelines*: Read this manual to learn more about security guidelines and recommendations for TIBCO ActiveMatrix Service Grid.
- *TIBCO ActiveMatrix Service Grid Release Notes*: Read this manual for a list of new and changed features, steps for migrating from a previous release, and lists of known issues and closed issues for the release.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, 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](https://community.tibco.com). For a free registration, go to <https://community.tibco.com>.

Introduction to Custom Actions

You can configure and enforce policies for ActiveMatrix using TIBCO ActiveMatrix Policy Director Governance. ActiveMatrix nodes and ActiveMatrix 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 Nodes.

Governance Agents intercept service requests, response, 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 you want to enforce the policy for (Service, Reference, and so on.)
- The PEP you want to execute the action in (Message In Flow, Out flow, Fault flow, and so on.)
- The action and configuration that makes up the policy (Authentication, Authorization, and so on.)

Out-of-the-box, ActiveMatrix Policy Director Governance supports many types of Governed Objects, PEPs, and Actions. To configure commonly used combinations of actions (and action configurations), the PEPs for execution, and the Governed Object types they are applicable, ActiveMatrix Policy Director Governance provides many out-of-the-box Policy Templates. The Security and Logging policies in ActiveMatrix 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 execute 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 TIBCO ActiveMatrix Policy Director Governance.

For more details on the Governance concepts, refer to the *TIBCO ActiveMatrix Service Grid Concepts* guide.

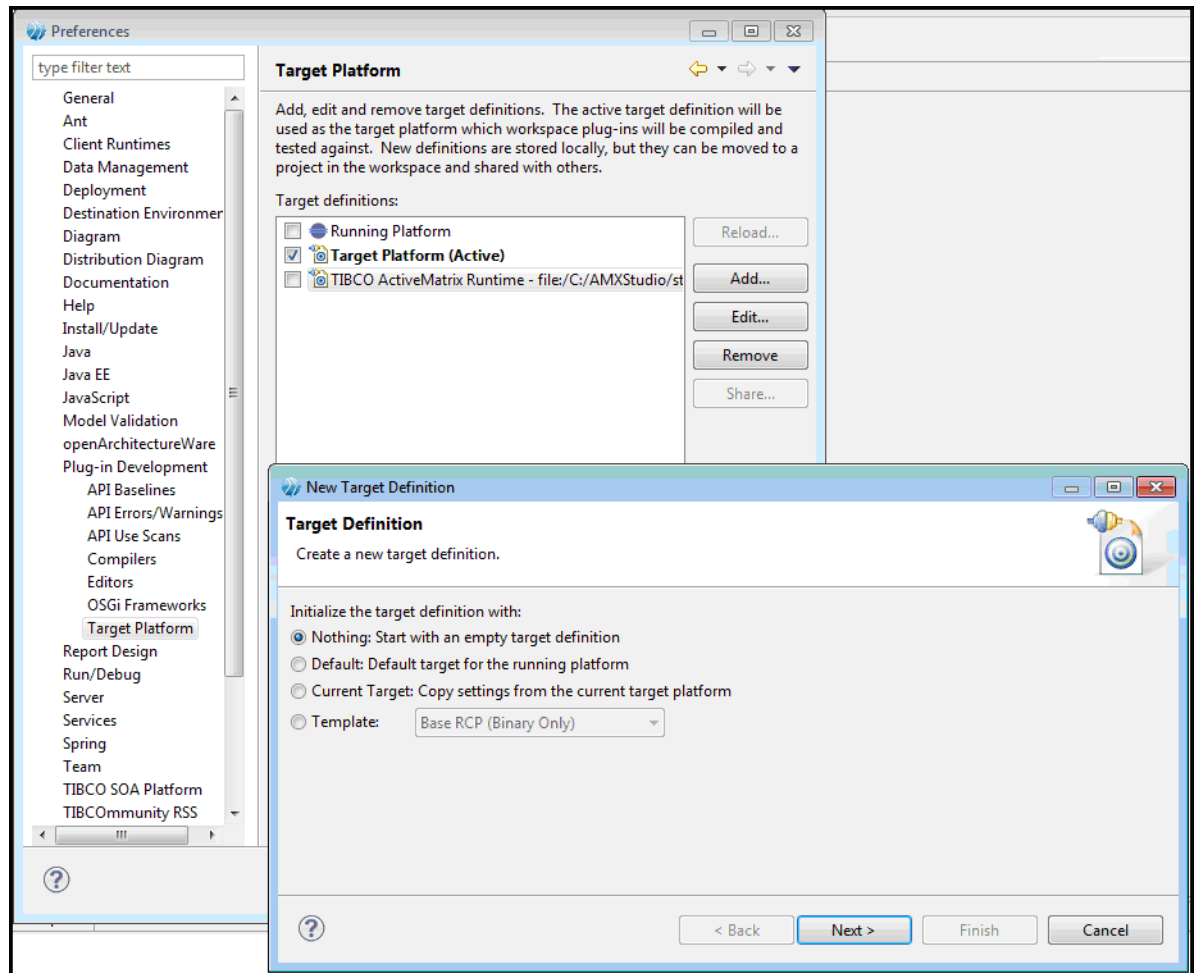
Setting up the Eclipse Environment for Plug-in Development

Prerequisites

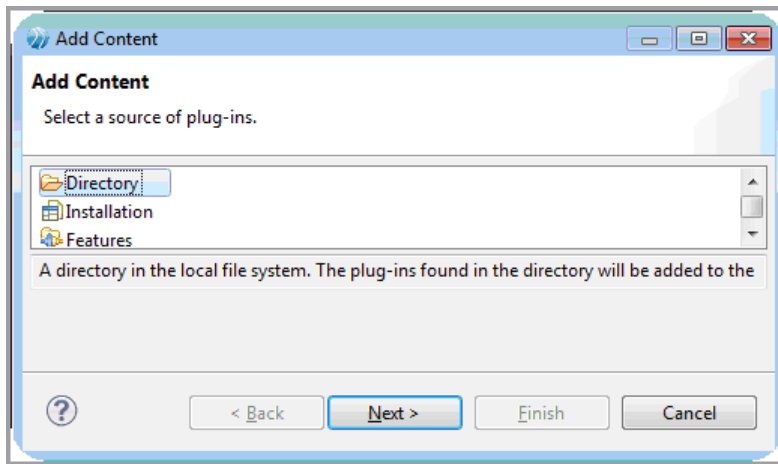
Set up Eclipse for plug-in development. For this tutorial, ActiveMatrix Studio is used as the editor but any other editor can be used for developing the policy action.

Procedure

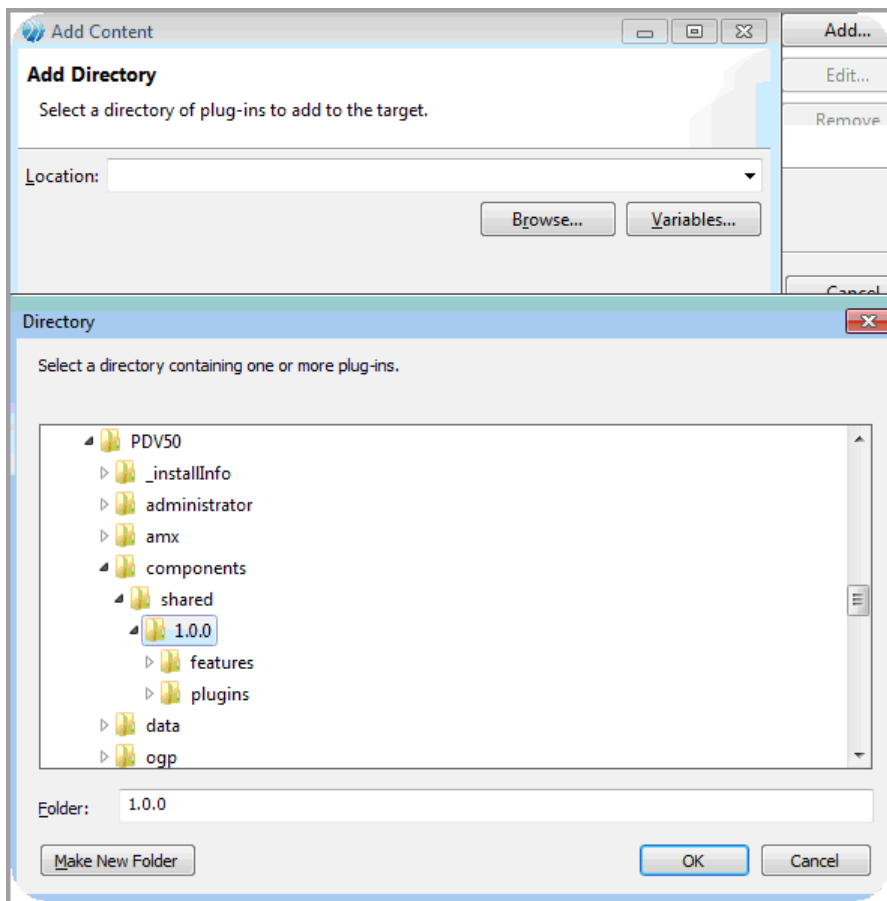
1. Navigate to **Window > Preferences > Plug-in Development > Target Platform** and create a new target with an empty target definition. Click **Next**.



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:

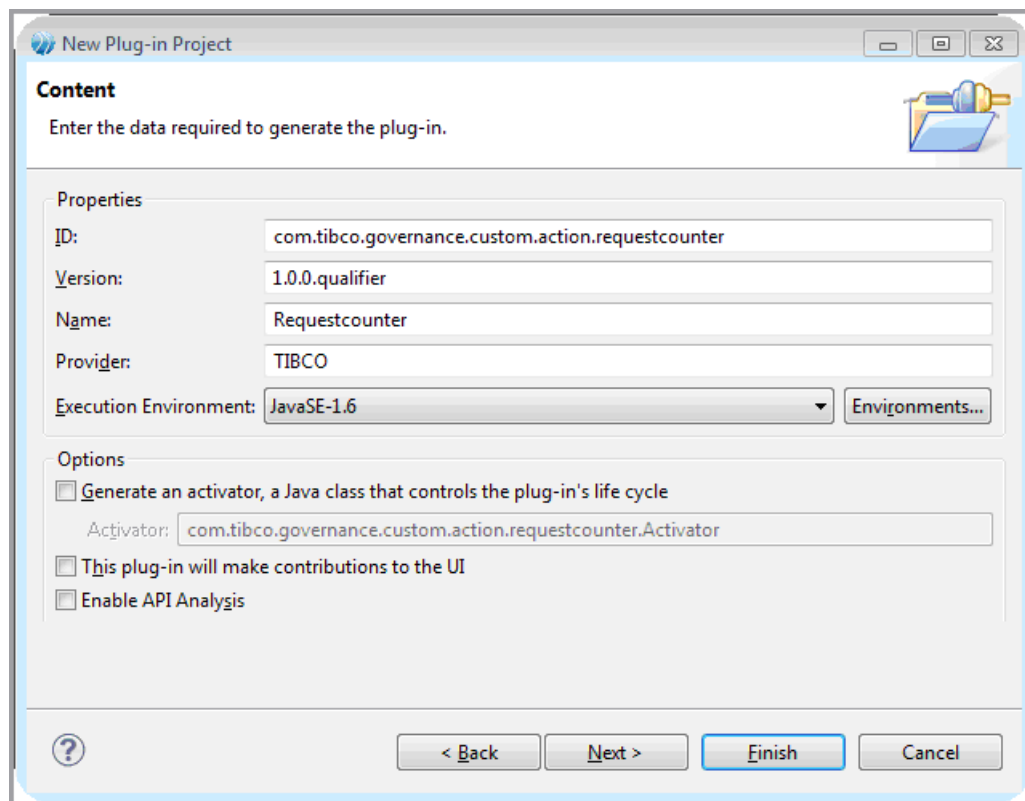
- `com.tibco.governance.agent.action.Action`
- `com.tibco.governance.agent.action.ActionConfigurationProcessor`

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 an `execute` method. This `execute` method is called each time the policy is invoked. This method may implement the complete execution logic.

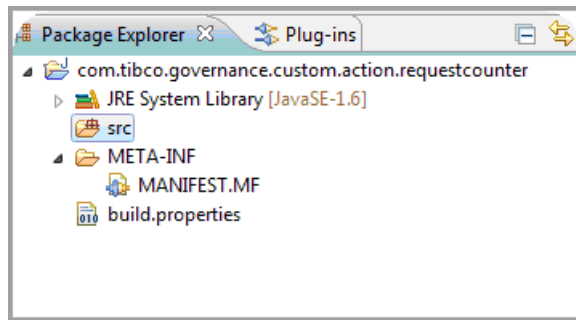
To create a custom action, create a new **Java-plugin** project first.

Procedure

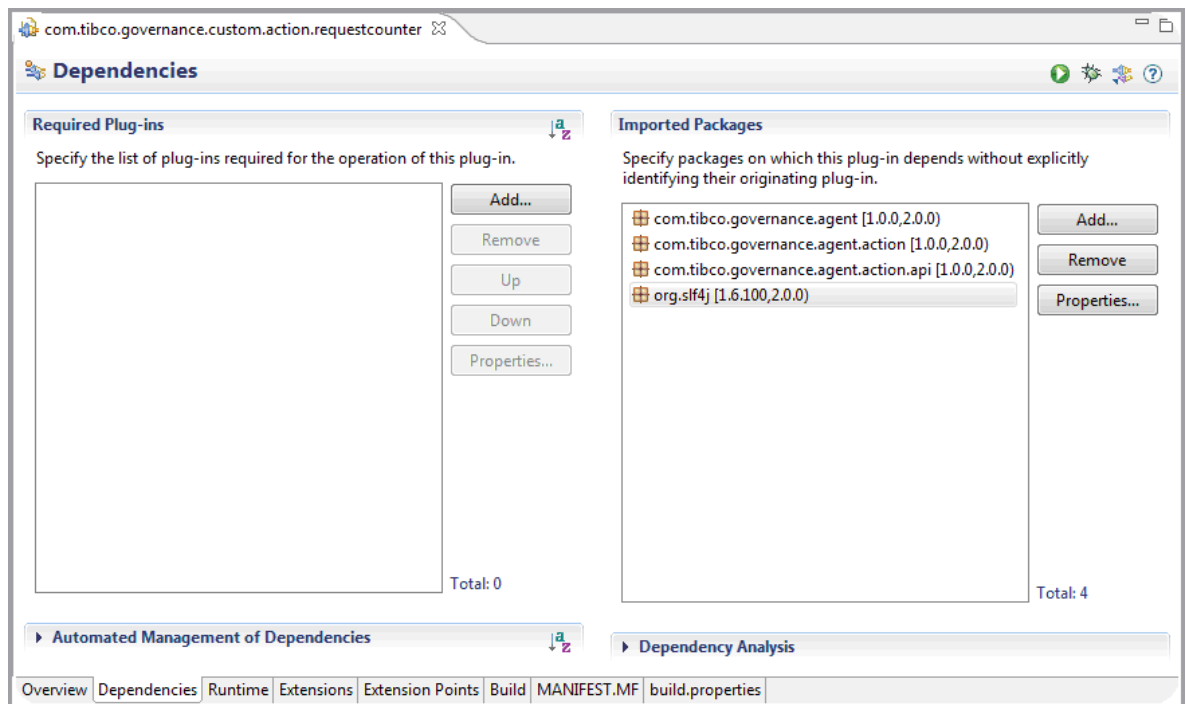
1. Clear the **Generate an activator, a Java class that controls the plug-in's life cycle** and **This plug-in will make contributions to the UI** check boxes. Click **Finish**.



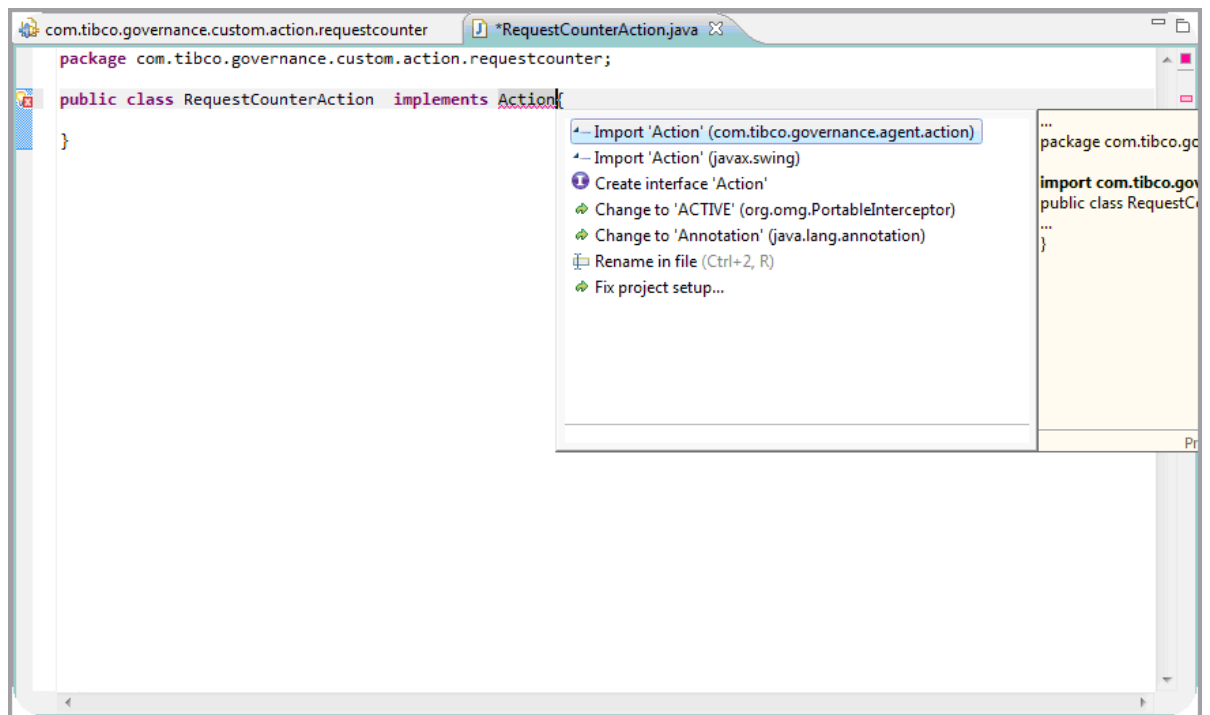
An empty plug-in project is created. **MANIFEST.MF** is created in the **META-INF** folder.



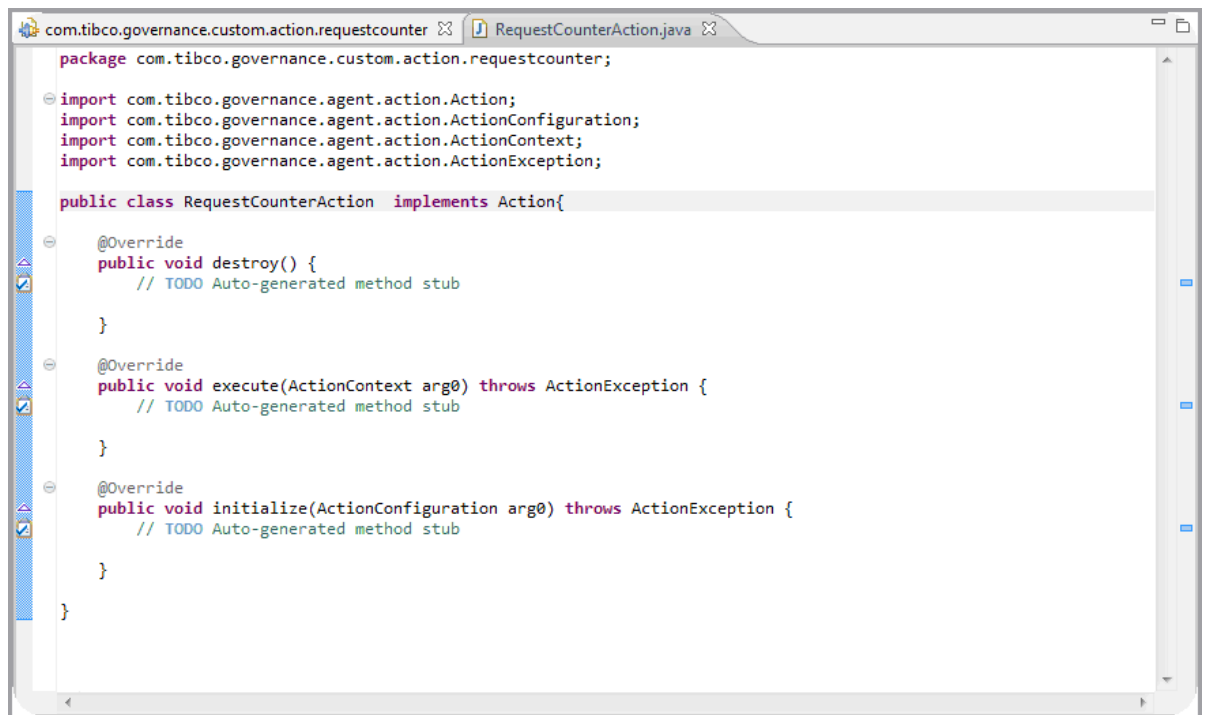
2. Open the **MANIFEST.MF** file. Under **Imported Packages**, click **Add**, and add the `com.tibco.governance.agent`, `com.tibco.governance.agent.action`, and `com.tibco.governance.agent.action.api` packages. **Save** the file.



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 unimplemented 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.

```

package com.tibco.governance.custom.action.requestcounter;

import javax.xml.namespace.QName;

import com.tibco.governance.agent.action.Action;
import com.tibco.governance.agent.action.ActionConfiguration;
import com.tibco.governance.agent.action.ActionConfigurationProcessor;
import com.tibco.governance.agent.action.ActionContext;
import com.tibco.governance.agent.action.ActionException;

public class RequestCounterActionConfigurationProcessor implements
    ActionConfigurationProcessor {

    public void addAction(String arg0, Action arg1) throws ActionException {
        // TODO Auto-generated method stub
    }

    @Override
    public Action createAction(ActionConfiguration arg0) throws ActionException {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

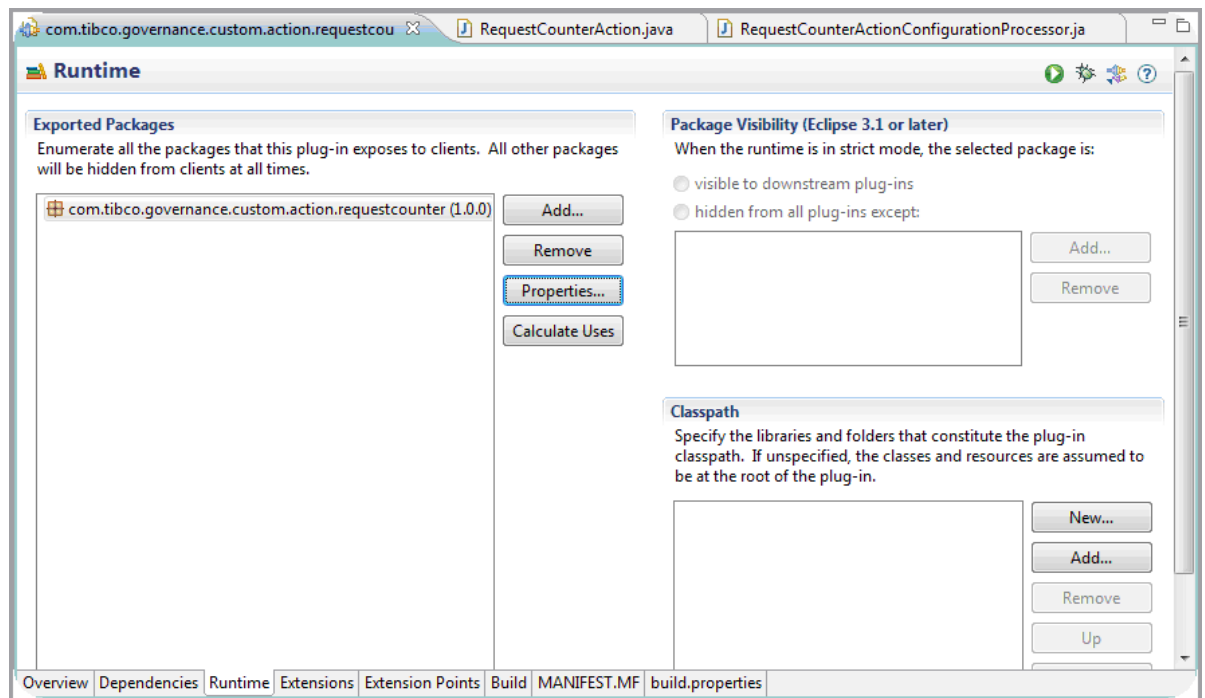
    @Override
    public void executeAction(String arg0, ActionContext arg1)
        throws ActionException {
        // TODO Auto-generated method stub
    }

    @Override
    public QName getName() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void initialize() {
        // TODO Auto-generated method stub
    }
}

```

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](#) on page 28.
6. After the business logic has been added, ensure that the custom action package is exported to **MANIFEST.MF**.



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

Packaging the Action for ActiveMatrix

To deploy the action on ActiveMatrix, it must be wrapped as an ActiveMatrix component. A DAA is generated from the ActiveMatrix composite; which is then deployed on ActiveMatrix.

For an ActiveMatrix composite to identify that it is an ActiveMatrix extension:

1. Open the composite file using a Text Editor.
2. Add the following tag: **scaext: extension** 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"/>
```

3. Modify **sca:composite** to include `xmlns:scaext="http://xsd.tns.tibco.com/amf/models/sca/extensions"`.

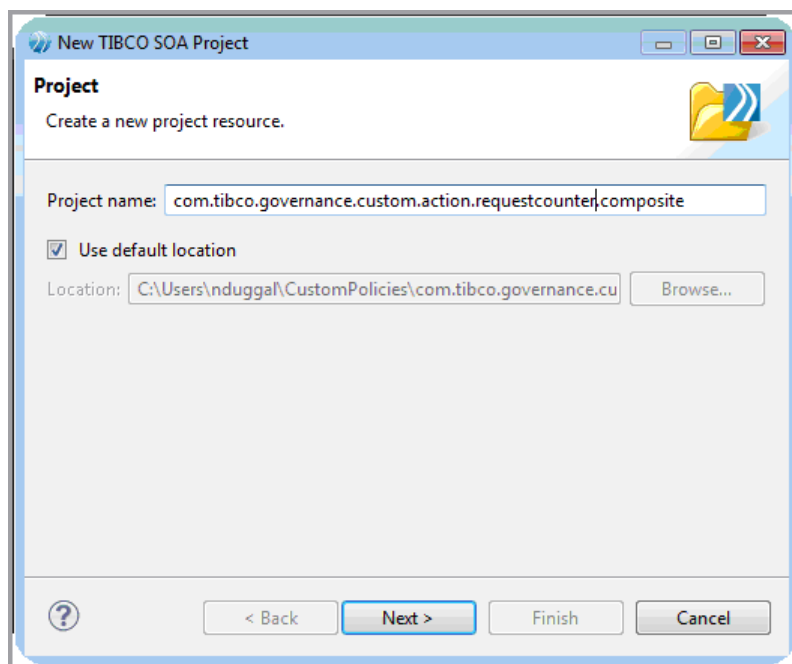
If the **scaext:extension** tag is specified, the ActiveMatrix component only creates an instance of the Action Configuration Processor when deployed (if it does not already exist).

For creating an ActiveMatrix composite wrapper, see [Creating an ActiveMatrix Composite Wrapper for a Custom Action](#) on page 14.

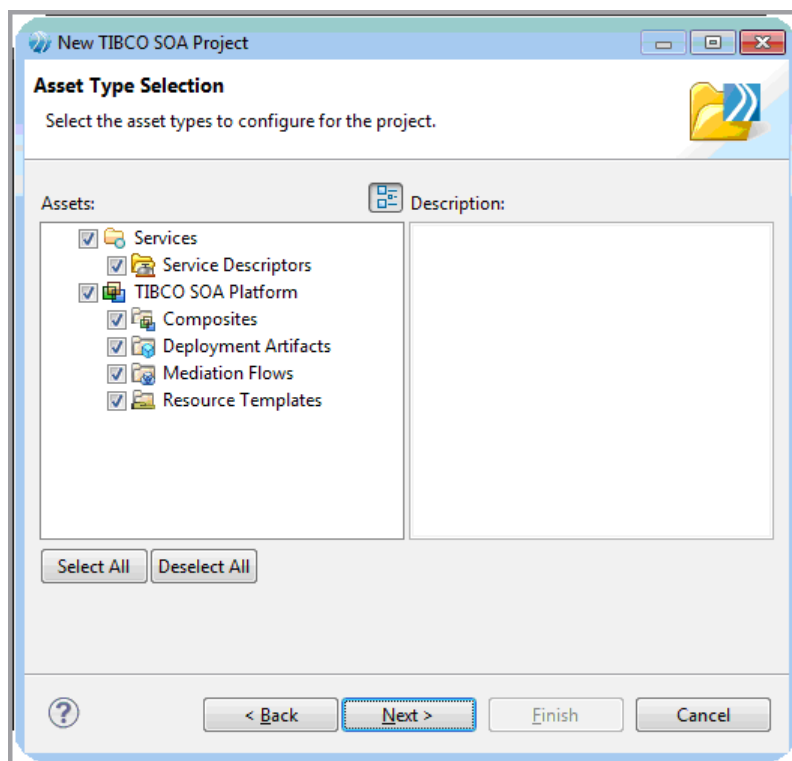
Creating an ActiveMatrix Composite Wrapper for a Custom Action

Procedure

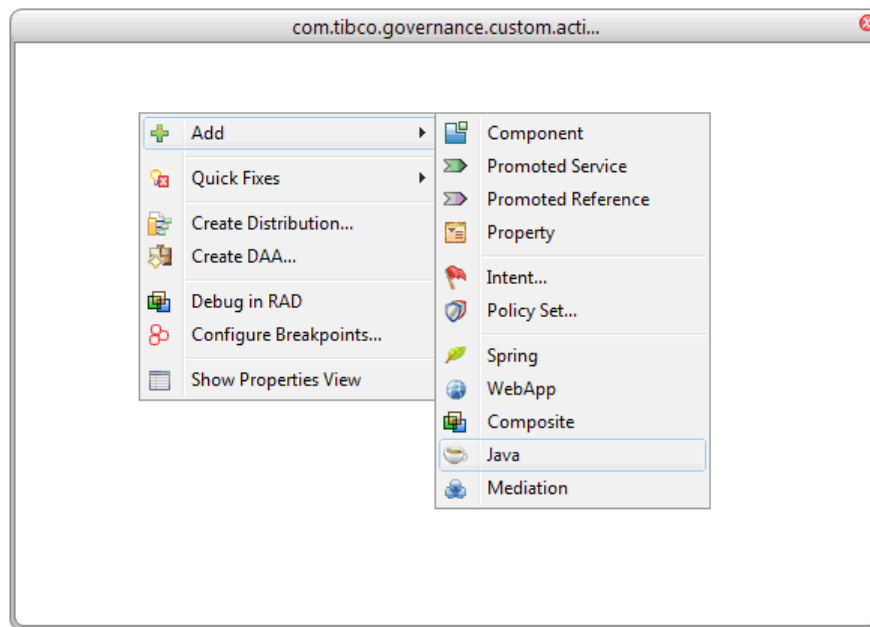
1. Create an SOA project using Business Studio.



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.



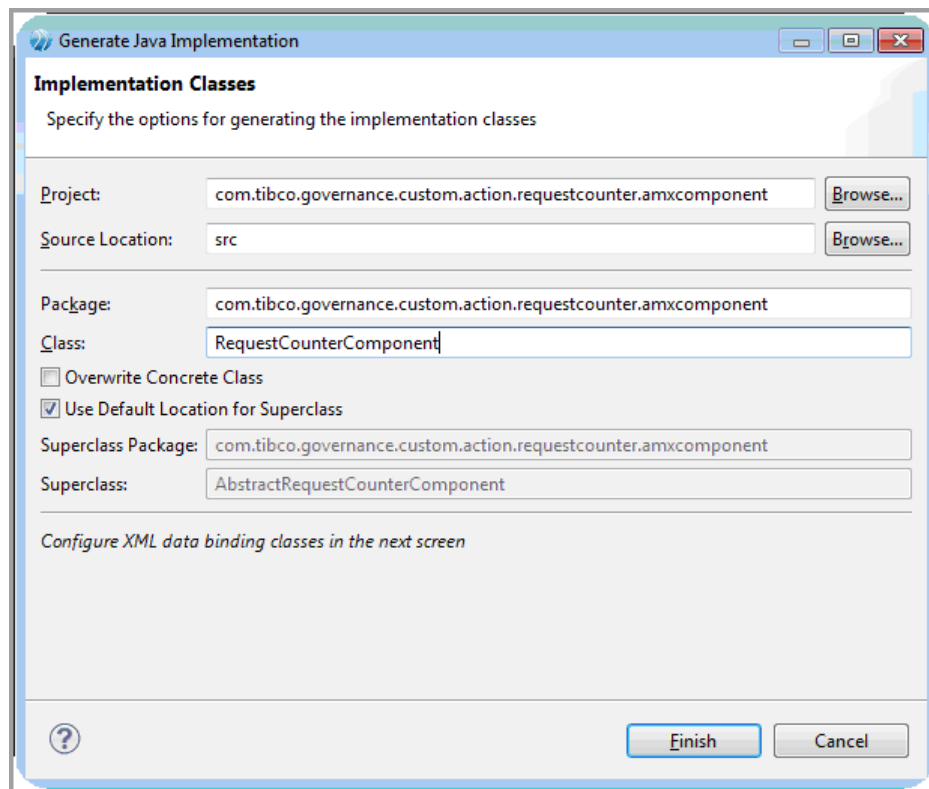
4. Open the **composite file** in a text editor and add the given tags after substituting the customA action name in to the tags:

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

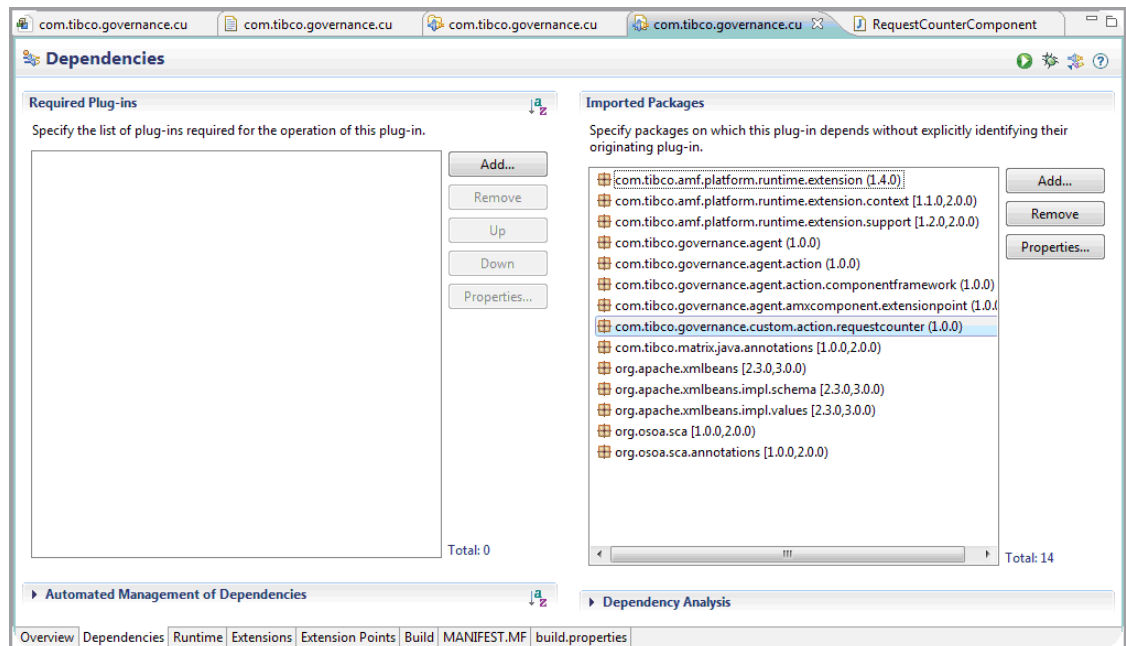
Also edit the **sca:composite** tab to define the scaext namespace. For example:

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

5. 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.



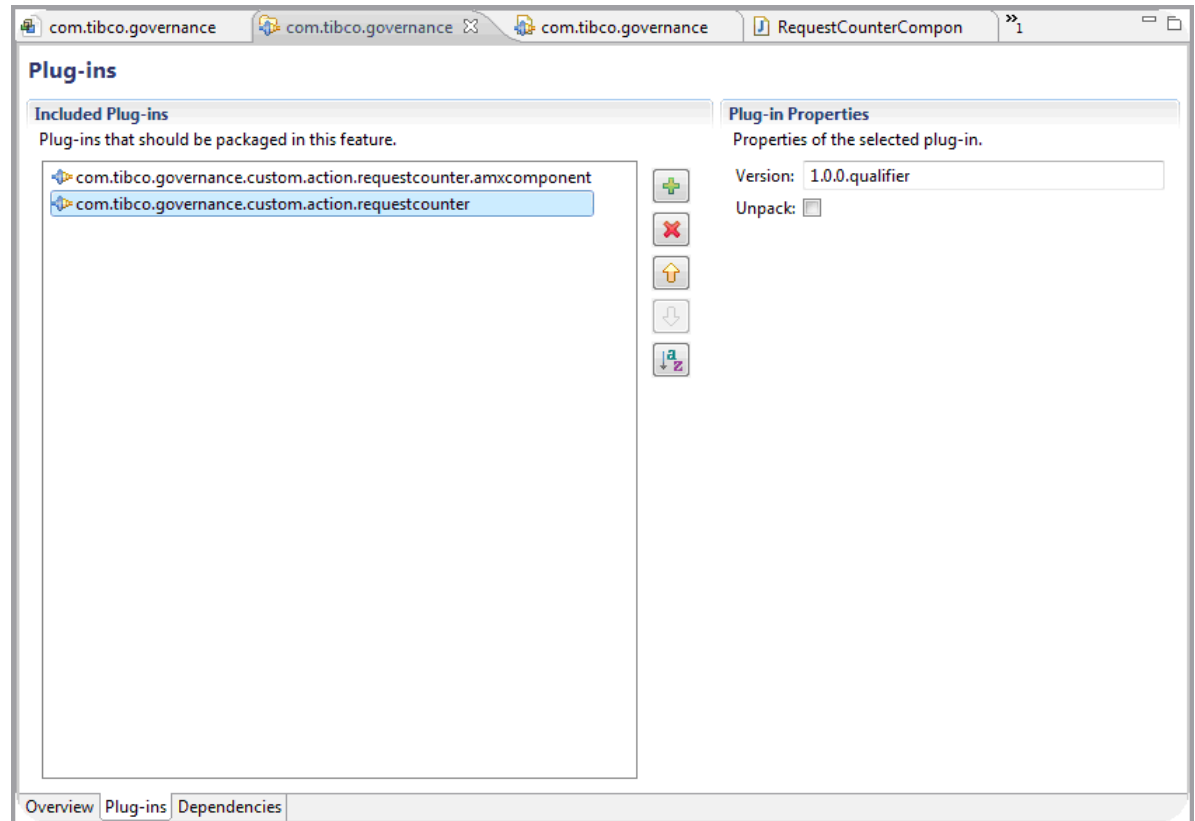
6. Navigate to **MANIFEST.MF** of the generated implementation and ensure the packages are added. If not, manually add the packages.



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 **amxcomponent** plug-in 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. For a completely implemented ActiveMatrix component, see Request Counter in [Sample Action Source Code](#) on page 28.

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. Refer to [Deploying Policy Templates](#) on page 25 for more information on where these templates and `Action.xsd` files are to be copied.

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 parameters and the allowed tags, see [Parameters and Action Templates](#) on page 29.

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 the **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>

```

- Change '\$PARAMETERGROUPNAME' to the name of the parameter group name.
- Change '\$INPUT1' to the name of the first input parameter.
- 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.
- Change the '\$FIRST INPUT' to enter the description of the parameter that is being defined.
- 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.
- Change '\$TYPE_OF_INPUT' to the type of the input parameter.

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

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

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

NOTE: 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>

```

- Change the **config fragment** as expected by the action. configFragment is interpreted by 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>

```

- 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 [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 located 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 CustomAction.rtp file from [Sample Action Rule Templates and User Interface Files](#) on page 31.

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 throws 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 **\$TYPEOFOBJECT** 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:$TYPEOFOBJECT</ogsei_base:leafObjectType>
  </ogsei_base:targetObjectGroupSelector>
</ogsei:togSelectors>
```

For a list of all valid object types, see [Parameters and Action Templates](#) on page 29.

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](#) on page 31.

User Interface

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



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 `locales` 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.

For more information on the General Interface UI, refer to the General Interface website: <http://generalinterface.org/docs/display/DOC/Learning+Center>

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

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 already 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 already exist.
3. Copy the action schema file to `CONFIG_HOME/admin/amxadmin/shared/repo/governance/actionschema/custom` folder.

Deploying User Interface

Procedure

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

```
CONFIG_HOME/admin/amxadmin/shared/repo/governance/ui/rules/pd/templates/
$YourCustomActionName
```

For Example, for 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

Procedure

- Deploy the DAA that was created on ActiveMatrix Policy Director Governance Administrator server.
For a complete implemented ActiveMatrix component, see Request Counter Sample in [Sample Action Source Code](#) on page 28.

Debugging and Logging

You can add a debug point in the custom action and remote debug the action using Eclipse or ActiveMatrix Business Studio.

Debugging and Logging ActiveMatrix 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 remote debug the action using Eclipse or ActiveMatrix Studio by connecting to the port specified.

Troubleshooting

Issue	Resolution
Policy does not appear in the governance control wizard template even after the templates are 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 component 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.java

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.
- 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 is used as the value of the parameter.
- 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 check boxes 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 actually inform the agent on where the policies are to be enforced. A policy can be applied at various stages and phases.

For more information, refer to the Concepts guide.

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



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 at:

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 at:

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 javascript files located at:

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 at:

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 at:

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 at:

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 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, refer to [Debugging and Logging](#) on page 26.

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);
```