THE POWER TO KNOW.

Developing JSR-168-Compliant Portlets for the SAS® Information Delivery Portal 4.2

SAS® Documentation

The correct bibliographic citation for this manual is as follows: . . . , : .

1st printing,

Contents

Chapter 1

Concepts for Developing JSR-168-Compliant Portlets

Introduction to Developing JSR-168-Compliant Portlets

Portlets are the information display components of a Web portal application such as the SAS Information Delivery Portal. A portlet can process requests from the user and generate dynamic content such as report lists, alerts, workflow notifications, or performance metrics.

In addition to a set of standard portlets, the SAS BI Portlets product provides a framework that enables you to create custom portlets that use the JSR 168 portlet specification.

JSR-168-compliant portlets are different from portlets that are proprietary to SAS in the following ways:

- JSR-168-compliant portlets are implemented by using an industry-standard API. Proprietary SAS portlets are implemented by using the SAS Portlet API.
- JSR-168-compliant portlets can be deployed in either IBM WebSphere Portal or SAS Information Delivery Portal. Proprietary SAS portlets can be deployed in SAS Information Delivery Portal only.

For information about developing portlets using the SAS Portlet API, see *Developing Portlets for the SAS Information Delivery Portal*.

Requirements for Developing JSR-168-Compliant Portlets for SAS

You must meet the following requirements for developing custom JSR-168-compliant portlets:

- SAS Information Delivery Portal and SAS BI Portlets must be installed on the server where you test your custom portlet.
- You must perform your initial portlet development for the SAS Information Delivery Portal. Any compatibility issues with IBM WebSphere Portal should be resolved after the portlet is working in the SAS Information Delivery Portal.

SAS recommends that you develop your portlets on a Windows machine and that you use the JBOSS application server.

Chapter 2

Tools for Creating JSR-168-Compliant Portlets

Using the Portlet API

In general, custom JSR-168-compliant portlets for SAS can be developed by using industry-standard Java classes. The **`com.sas.portal.portlet`** classes in the SAS API are specific to proprietary portlets and are not used with JSR 168 portlets.

However, a class that is specific to SAS is needed to obtain a user context for the portlet:

`com.sas.web.keys.CommonKeys`
defines common prefix strings and other keys used to prevent namespace collisions among various SAS domains.

The **`USER_CONTEXT`** field contains a string that can be used to obtain the user context for a portlet session.

In JSP code, you can obtain a user context directly from the HTTP session by using the following code fragments:

```
<%
import com.sas.services.user.UserContextInterface;
import com.sas.web.keys.CommonKeys;

UserContextInterface userContext =
   (UserContextInterface) session.getAttribute(CommonKeys.USER_CONTEXT);
%>
```

In Java code, you can obtain a user context from the portlet session by using the following code fragments:

```
import javax.portlet.*;
import com.sas.services.user.UserContextInterface;
import com.sas.web.keys.CommonKeys;

UserContextInterface userContext = (UserContextInterface)
   portletRequest.getPortletSession().getAttribute(
      CommonKeys.USER_CONTEXT,
      PortletSession.APPLICATION_SCOPE);
```

The UserContextInterface enables you to access all of the user metadata from the SAS Metadata Repository. For more information, see **http://support.sas.com/rnd/gendoc/bi92/api/Foundation/com/sas/services/user/UserContextInterface.html** in the SAS API documentation.

Using the Testportlet Scripting Facility

Overview of the Testportlet Scripting Facility

The *Testportlet Scripting Facility* is a scripting tool that enables you to initialize your portlet development directories and to compile your portlet source into an EAR file. The files for this facility can be found in the ***SAS-configuration-directory*\Lev1\CustomAppData\testportlet** directory. The scripting facility should be used for portlet development because it provides a process that integrates your custom portlets with the SAS 9.2 versioned JAR repository.

Note: Before you begin developing a custom portlet, ensure that the SAS Metadata Server is running so that metadata can be accessed during the configuration and deployment processes.

Creating a Portlet by Using the Testportlet Scripting Facility

The following steps provide an overview of creating a portlet using the scripting facility. For a detailed example of creating a portlet, see "Sample Portlet: HelloUserJSR168PortletSample" on page 13.

1. Create a source directory for the code associated with the portlet. This directory is referred to in subsequent instructions as the *portlet source directory*.
2. Create a configuration directory for the portlet under the ***SAS-configuration-directory*\Lev1\CustomAppData** directory. Use the portlet name for the configuration directory name.

   The following rules apply to the portlet name and configuration directory structure:

   - Neither portlet names nor their paths can contain spaces.
   - The portlet name must be unique.
3. Copy the files from ***SAS-config-directory*/Levn/CustomAppData/testportlet** to the new directory.
4. Edit the custom.properties file in the portlet configuration directory to specify the portlet name and title and the locations for the configuration and source files.

Note: You must use forward slashes (/) in your directory values. For example, **C:/SAS/EBI/Lev1/CustomAppData/SampleHelloUserJSR168Portlet**.

5. Enter the following command to create the directory structure for your portlet:

```
cfg createJSR168PortletDirectories
    -Dmetadata.connection.passwd="unrestricted-user-password"
```

where *unrestricted-user-password* is the password for the unrestricted user.

Note: You can submit a password as clear text or as an encoded string from the PWENCODE procedure. For more information, see *Encryption in SAS 9.2*.

Check the customconfig.log file in ***SAS-config-directory*/Levn/CustomAppData/SampleHelloUserJSR168Portlet** to ensure that the script completed successfully.

The following directory structure is created:

```
source-directory
   Configurable
      ears
         archive-name
            META-INF
      wars
         archive-name
            WEB-INF
   Picklists
      wars
         archive-name
   Static
      ears
         archive-name
            META-INF
      lib
      wars
         archive-name
            jsp
            source
            WEB-INF
               classes
               spring-config
```

The **\Configurable** and **\Static** directory hierarchies are used to store the files needed to create PAR, EAR, and WAR files for the portlet, in the same directory structure as the PAR, EAR, and WAR files themselves. The **\Configurable** hierarchy is used for files in which values are substituted from the portlet configuration file when the portlet is built. The **\Static** hierarchy is used for files that do not require substitution. The **\Picklists** directory hierarchy is used to store picklist files that specify which of the JAR files from the SAS versioned JAR repository need to be included in the portlet. The **\Static\lib** directory is used to store additional JAR files needed at compile time.

6. Create source files for your portlet in the source directory structure.

7. Copy the SAS BI Portlets picklist file to tell the portlet which of the JAR files from the SAS versioned JAR repository need to be included in the portlet.

Note: After a SAS maintenance release is applied at your site, you must copy the updated picklist and rebuild and redeploy the PAR and EAR files for custom portlets.

8. Add any additional JAR files to the **\Static\Lib** directory.
9. Enter the following command to compile the portlet:

   ```
   cfg compileJSR168Portlet -Dmetadata.connection.passwd="unrestricted-user-password"
   ```

   where *unrestricted-user-password* is the password for the unrestricted user.

   Note: You can submit a password as clear text or as an encoded string from the PWENCODE procedure. For more information, see *Encryption in SAS 9.2*.

   Check the customconfig.log file to ensure that the script completed successfully.
10. Enter the following command to build the EAR file:

    ```
    cfg buildJSR168Webapps -Dmetadata.connection.passwd="unrestricted-user-password"
    ```

    where *unrestricted-user-password* is the password for the unrestricted user.

    Note: You can submit a password as clear text or as an encoded string from the PWENCODE procedure. For more information, see *Encryption in SAS 9.2*.

    Check the customconfig.log file to ensure that the script completed successfully.
11. Use the Portlet Deployment Tool to deploy the portlet EAR file into the BI Portlets Web application.
12. Stop the Web application server on which the SAS Information Delivery Portal and the BI Portlets Web application are running.
13. Rebuild the SAS Information Delivery Portal application by using the SAS Deployment Manager. For more information, see "Rebuilding the SAS Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.
14. Redeploy the EAR files for the SAS Information Delivery Portal and the BI Portlets Web application.

    The SAS Information Delivery Portal file is located at ***SAS-config-directory*/Levn/Web/Staging/sas.portal4.2.ear**.

    The BI Portlets file is located at ***SAS-config-directory*/Levn/Web/Temp/PDT/staging/sas.biportlets4.2.ear**.

    For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.
15. Start the Web application server on which the SAS Information Delivery Portal is deployed. The custom portlet should now be available to the portal.

Using the Portlet Deployment Tool

Overview of the Portlet Deployment Tool

The Portlet Deployment Tool (PDT) is a command-line script that enables you to add JSR-168-compliant portlets to the BI Portlets Web application.

The basic process to deploy a portlet with the PDT is as follows:

1. Modify the build.properties file to specify the parameters for the PDT.
2. Execute the PDT by submitting an Ant command in the directory where the build.properties file is located.
3. Redeploy the BI Portlets Web application by using the modified sas.biportlets.ear file that is created by the PDT. This file is located in the **/staging** subdirectory of the working directory that you specify in the build.properties file.

Specify Parameters in the build.properties File

The Portlet Deployment Tool is an Ant script that obtains its parameters from a file called build.properties. This file is located in ***SAS-config-directory*/Levn/Web/Applications/SASBIPortlets4.2/PortletDeploymentTool/src/**.

Before you can execute the PDT, you must specify the following parameters in the build.properties file:

servlet-context-name
specifies the context name for the portlet. The value of this parameter must match the value of the <context-root> element in the application.xml file in the portlet source.

Note: If you use the Testportlet Scripting Facility, then the <context-root> value in application.xml is obtained from the webapp.testportlet.contextroot parameter in the custom.properties file.

web-app-name
specifies the name of the WAR file for the portlet, without the .war extension. For example, if the WAR file for your portlet is named sample.hellouser.jsr168.war, then specify the value sample.hellouser.jsr168.

web-app-server
specifies the type of Java application server where the BI Portlets Web application is deployed. Specify either jboss, weblogic, or websphere.

portlet-ear-file-name
specifies the name of the EAR file for the portlet.

portlet-ear-file-path
specifies the full path to the EAR file for the portlet, including the filename. You must use the forward slash (/) character to delimit the directories in your path. For example, **C:/SAS/EntBIServer/Lev1/Web/Staging/myportlet.ear**.

work-dir
specifies the working directory where the PDT places the modified files for the BI Portlets Web application.

If the working directory contains BI Portlets files from a previous execution of PDT, then the PDT adds the new portlet to the BI Portlets files in the working directory. For more information, see "Add Additional Portlets to the BI Portlets Web Application" on page 8.

Execute the PDT Script

To execute the PDT, follow these steps:

1. In a command shell, navigate to the ***SAS-config-directory*/Levn/Web/Applications/SASBIPortlets4.2/PortletDeploymentTool/src** directory.
2. Call the level_env.bat script to set environment variables.

   On Windows, enter the following command:

```
..\..\..\..\..\level_env.bat
```

   On UNIX, enter the following command:

```
../../../../../level_env.sh
```

3. Call the launchant.bat script to execute the Portlet Deployment Tool.

   On Windows, enter the following command:

```
..\..\..\..\..\..\Utilities\launchant.bat
```

   On UNIX, enter the following command:

```
../../../../../../Utilities/launchant.sh
```

Add Additional Portlets to the BI Portlets Web Application

You can add only one portlet to the BI Portlets Web application each time you run PDT. However, you can add additional portlets by running the PDT again for each additional portlet.

The *work-dir* parameter in build.properties specifies the working directory where the PDT places the modified files for the BI Portlets Web application. If the working directory is empty, then PDT adds the new portlet to a copy of the original BI Portlets EAR file. If the working directory contains BI Portlets files from a previous execution of PDT, then the PDT adds the new portlet to the files that are in the working directory.

For example, to add two portlets:

1. Stop the Web application server on which the SAS Information Delivery Portal and the BI Portlets Web application are running.
2. Specify the parameters for the first portlet in build.properties. Specify an empty directory for the *work-dir* parameter.
3. Run the PDT. The portlet is added to a copy of the original BI Portlets Web application. The modified copy is placed in the working directory that you specified.
4. Modify build.properties again to specify the properties for the second portlet. Specify the same working directory that you did in step 1.
5. Run the PDT again. The second portlet is added to the modified BI Portlets Web application in the working directory. The BI Portlets Web application in the working directory now contains both new portlets.
6. Redeploy the BI Portlets Web application by using the sas.biportlets.ear file in the **/staging** subdirectory of the working directory. For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.
7. Restart the Web application server on which the SAS Information Delivery Portal is deployed.

Retaining Custom Portlets after Rebuilding the BI Portlets Web Application

When a hot fix is applied or maintenance is installed, the SAS Deployment Manager is used to rebuild the BI Portlets Web application. The SAS Deployment Manager can also be run manually. In any of these cases, the BI Portlets Web application is rebuilt from scratch; it does not contain any custom portlets.

To add your custom portlets back to the BI Portlets Web application, follow these steps after the SAS Deployment Manager process has completed:

1. Stop the web application server on which the SAS Information Delivery Portal and the BI Portlets Web application are running.
2. The PDT can add only one portlet to the BI Portlets Web application at a time (for more information, see "Add Additional Portlets to the BI Portlets Web Application" on page 8). Start by specifying the parameters for the first portlet in build.properties. Specify an empty directory for the *work-dir* parameter.
3. Run the PDT. The portlet is added to a copy of the original BI Portlets Web application. The modified copy is placed in the working directory that you specified.
4. Modify build.properties again to specify the properties for the next portlet. Specify the same working directory that you did in step 2. Repeat steps 3 and 4 for the remaining portlets. The portlets are added to the modified BI Portlets Web application in the working directory. The BI Portlets Web application in the working directory contains all of the portlets when finished.
5. Redeploy the BI Portlets Web application by using the sas.biportlets.ear file in the **/staging** subdirectory of the working directory. For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.
6. Restart the Web application server on which the SAS Information Delivery Portal is deployed.

Start Over with the BI Portlets Web Application

If you want to add a portlet to a copy of the original BI Portlets Web application and remove any portlets that you added previously, follow these steps:

1. Stop the Web application server on which the SAS Information Delivery Portal and the BI Portlets Web application are running.
2. Remove any unused portlet PAR files and exploded portlet directories that were generated by the Testportlet Scripting Facility. The PAR files are located in the ***SAS-configuration-directory*\Levn\Web\Applications\SASPortlets4.2\Deployed** directory. The exploded portlet directories are located in the ***SAS-configuration-directory*\Levn\Web\Applications\SASPortlets4.2\Exploded** directory.
3. Remove any unused portlet EAR files and exploded application directories that were generated by the Testportlet Scripting Facility. The EAR files are located in the ***SAS-configuration-directory*\Levn\Web\Staging** directory. The exploded application directories are located in the ***SAS-configuration-directory*\Levn\Web\Staging\exploded** directory.

4. Specify the parameters for the new portlet in build.properties. Specify an empty directory for the *work-dir* parameter.
5. Run the PDT. Modified files for the BI Portlets Web application are placed in the working directory that you specified.
6. Rebuild the EAR file for the SAS Information Delivery Portal. This EAR file contains files that are associated with each portlet. To add the new portlet, you must rebuild the SAS Information Delivery Portal by using the SAS Deployment Manager. For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.
7. Redeploy the SAS Information Delivery Portal and the BI Portlets Web applications. The SAS Information Delivery Portal EAR file is located at ***SAS-config-directory*/Levn/Web/Staging/sas.portal4.2.ear**. The BI Portlets EAR file is located in the **/staging** subdirectory of the working directory. For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.
8. Restart the Web application server on which the SAS Information Delivery Portal is deployed.

Remove Custom Portlets from the BI Portlets Web Application

To remove any custom JSR-168-compliant portlets from the BI Portlets Web application, redeploy BI Portlets by using the original file in ***SAS-config-directory*/Levn/Web/Staging**. Follow these steps to remove the custom portlets:

1. Stop the Web application server on which the SAS Information Delivery Portal and the BI Portlets Web application are running.
2. Remove any unused portlet PAR files and exploded portlet directories that were generated by the Testportlet Scripting Facility. The PAR files are located in the ***SAS-configuration-directory*\Levn\Web\Applications\SASPortlets4.2\Deployed** directory. The exploded portlet directories are located in the ***SAS-configuration-directory*\Levn\Web\Applications\SASPortlets4.2\Exploded** directory.
3. Remove any unused portlet EAR files and exploded application directories that were generated by the Testportlet Scripting Facility. The EAR files are located in the ***SAS-configuration-directory*\Levn\Web\Staging** directory. The exploded application directories are located in the ***SAS-configuration-directory*\Levn\Web\Staging\exploded** directory.
4. Rebuild the EAR file for the SAS Information Delivery Portal by using the SAS Deployment Manager.
5. Redeploy the SAS Information Delivery Portal and the original BI Portlets Web applications. The SAS Information Delivery Portal EAR file is located at ***SAS-config-directory*/Levn/Web/Staging/sas.portal4.2.ear**. The original BI Portlets EAR file is located at ***SAS-config-directory*/Levn/Web/Staging/sas.biportlets4.2.ear**. For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.

6. Restart the Web application server on which the SAS Information Delivery Portal is deployed.

Chapter 3

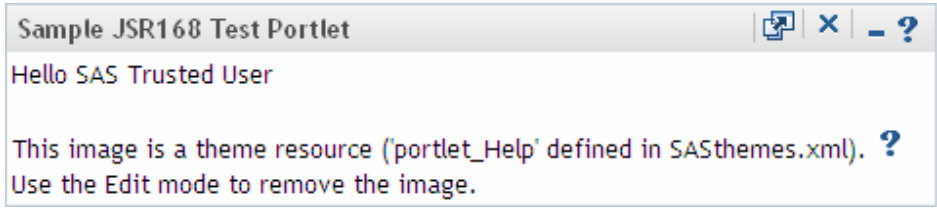
Sample Portlet: HelloUserJSR168PortletSample

Sample Portlet: HelloUserJSR168PortletSample

Overview of HelloUserJSR168PortletSample

The HelloUserJSR168PortletSample portlet is a JSR 168 portlet that greets the user by using the name of the user who is logged on. The portlet also displays images that are packaged with the portlet and retrieves resources from the theme service. The portlet features VIEW, EDIT, and HELP modes.

Figure 3.1 *The HelloUserJSR168PortletSample Portlet*



The portlet calls a Web application named sample.hellouser.jsr168.war, which uses SAS Foundation Services to access session information from the Portal Web application and

obtain the user name of the current user. The sample.hellouser.jsr168.war application generates an HTML fragment that is displayed within the portlet.

To create the HelloUserJSR168PortletSample sample, follow these steps:

1. Create the base directory for the portlet source.
2. Configure the scripting facility.
3. Use the scripting facility to create the subdirectories for the source.
4. Copy the picklist of required JAR files.
5. Configure files for the Spring Framework.
6. Create the configurable source files.
7. Create the static source files.
8. Copy the images for the portlet.
9. Prepare your development environment.
10. Compile the source code and build the portlet EAR file.
11. Load the portlet into the BI Portlets Web application.
12. Rebuild the SAS Information Delivery Portal.
13. Redeploy the Web applications.
14. Restart the Web application server.

Step 1: Create the Base Directory for the Portlet Source

Create the base directory. All of the portlet source files are placed in subdirectories of this path. Throughout this example, the base directory is referred to as the ***source-directory***.

Step 2: Configure the Scripting Facility

This sample uses the Testportlet Scripting Facility that is delivered with the SAS Information Delivery Portal. To configure the scripting facility for the sample, follow these steps:

1. Create a new directory within ***SAS-config-directory*/Levn/CustomAppData** named **SampleHelloUserJSR168Portlet**. Use the portlet name for the configuration directory name.

   The following rules apply to the portlet name and configuration directory structure:

   - Neither portlet names nor their paths can contain spaces.
   - The portlet name must be unique.

2. Copy the files from ***SAS-config-directory*/Levn/CustomAppData/testportlet** to the new directory.
3. Edit the custom.properties file in your **SampleHelloUserJSR168Portlet** directory. The code that is highlighted must be changed or added to the file:

```
# If you change the value "testportlet", make sure to rename in all properties
# here as well as in the custom_config.xml.
config.currprod.12byte=testportlet
```

```
# Change the value of this property to be the name of your web application.
config.currprod.legalname=Hello User JSR168 Portlet Sample

# The value of this property should be the location where the configuration
# files are placed.  Make sure to change the level directory based on your
# installation and make sure to rename testportlet if the value of
# config.currprod.12byte changes above.
install.currprod.config.dir=SAS-config-directory/Levn/CustomAppData/SampleHelloUserJSR168Portlet

# Do not change the value of this property.  The name might be changed if you
# change the value of config.currprod.12byte above.
webappsrv.testportlet.server=server

# Change the value of this property to be the location of your portlet's source
# code and configuration files.  The name might be changed if you change the
# value of config.currprod.12byte above.
testportlet.install.dir=portlet-source-directory

# Change the value of this property to be the name of your war and ear
# file.  The name might be changed if you change the value of
# config.currprod.12byte above.
webapp.testportlet.archive.name=sample.hellouser.jsr168

# Change the value of this property to be the context root of your web
# application and the name of the portlet.  The name might be changed if you
# change the value of config.currprod.12byte above.
webapp.testportlet.contextroot=SampleHelloUserJSR168

# Change the value of this property to be the versioned name of your web
# application.  This property is only used for JSR168 portlets.  The name might
# be changed if you change the value of config.currprod.12byte above.
webapp.testportlet.display.name=Hello User JSR168 Portlet Sample

# Change the value of this property to be the name of the local services
# to use for the connection information.
webapp.testportlet.localservices.name=SASBIPortlets4.2

# Add any other configuration properties that your JSR168 portlet and/or web
# application may need.
```

Note: You must use forward slashes (/) in your directory values. For example, **`C:/SAS/EBI/Lev1/CustomAppData/SampleHelloUserJSR168Portlet`**.

Step 3: Create the Source Subdirectories

Perform the following steps to create the source directories that are needed for the portlet source:

1. Ensure that the SAS Metadata Server is running.
2. In a command shell, navigate to the ***`SAS-config-directory`*`/Levn/CustomAppData/SampleHelloUserJSR168Portlet`** directory.
3. Enter the following command:

```
cfg createJSR168PortletDirectories
    -Dmetadata.connection.passwd="unrestricted-user-password"
```

where *unrestricted-user-password* is the password for the unrestricted user.

Note: You can submit a password as clear text or as an encoded string from the PWENCODE procedure. For more information, see *Encryption in SAS 9.2*.

Check the customconfig.log file in ***SAS-config-directory*/Levn/CustomAppData/SampleHelloUserJSR168Portlet** to ensure that the script completed successfully.

4. Manually create these additional directories for this sample:

source-directory/Static/wars/sample.hellouser.jsr168/images

source-directory/Static/wars/sample.hellouser.jsr168/source/sample

source-directory/Static/wars/sample.hellouser.jsr168/source/sample/res

Your source directory should contain the following structure:

```
source-directory
   Configurable
      ears
         sample.hellouser.jsr168
            META-INF
      wars
         sample.hellouser.jsr168
            WEB-INF
   Picklists
      wars
         sample.hellouser.jsr168
   Static
      ears
         sample.hellouser.jsr168
            META-INF
      lib
      wars
         sample.hellouser.jsr168
            images
            jsp
            source
               sample
                  res
            WEB-INF
               classes
               spring-config
```

Step 4: Copy the Picklist of Required JAR Files

The portlet sample uses a picklist file to identify the JAR files from the SAS versioned JAR repository that are needed.

Copy the picklist from ***SAS-Home-Directory*/SASBIPortlets/4.2/Picklists/wars/sas.biportlets** to ***source-directory*/Picklists/wars/sample.hellouser.jsr168**.

Note: If you update your SAS software, then you must update the picklist and repeat steps 9 through 15.

Step 5: Configure Files for the Spring Framework

The sample portlet uses the Spring Framework to integrate with the Web Infrastructure Platform. To configure the Spring Framework for the portlet, perform the following steps:

1. Copy the Spring configuration files from ***SAS-Home-Directory*/SASBIPortlets/4.2/Static/wars/sas.biportlets/WEB-INF/spring-config** to ***source-directory*/Static/wars/sample.hellouser.jsr168/WEB-INF/spring-config**.
2. Modify the pc-config.xml file to remove the following lines:

```
<bean id="viewerFactory"
    class="com.sas.portal.plugins.viewers.components.omr.impl.SASPortalViewerImpl">
 <constructor-arg ref="baseUrlLocator" />
</bean>

<bean id="omrSearchInfoFactory"
       class="com.sas.portal.plugins.search.components.omr.impl.OMRSearchInfoFactoryImpl">
 <constructor-arg ref="localSecuredUser" />
<constructor-arg ref="logger" />
</bean>

<bean id="searchEngine"
       class="com.sas.portal.plugins.search.components.engine.impl.SearchEngineImpl">
<constructor-arg ref="localInformationService" />
<constructor-arg ref="omrSearchInfoFactory" />
<constructor-arg ref="logger" />
</bean>
```

Step 6: Create the Configurable Source Files

Overview of the Configurable Source Files

The *configurable source files* are files that contain parameter substitutions. These files are placed in the ***source-directory*/Configurable** directory structure. These files use values from the custom.properties file that you modified in step 2.

application.xml.orig

This file uses parameters from the custom.properties file to describe the contents of the portlet Web application.

Example Code 3.1 */Configurable/ears/sample.hellouser.jsr168/META-INF/application.xml.orig*

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                                 http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
             version="1.4">
    <display-name>@webapp.testportlet.display.name@</display-name>
    <description>@webapp.testportlet.display.name@</description>
    <module>
        <web>
            <web-uri>@webapp.testportlet.archive.name@.war</web-uri>
```

```
            <context-root>@webapp.testportlet.contextroot@</context-root>
        </web>
    </module>
</application>
```

web.xml.orig

This file uses parameters from the custom.properties file to define settings for the portlet Web application.

Example Code 3.2 */Configurable/wars/sample.hellouser.jsr168/WEB-INF/web.xml.orig*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="@webapp.testportlet.contextroot@" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>@webapp.testportlet.display.name@</display-name>

<!-- BEGIN Spring Integration -->
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>
        /WEB-INF/spring-config/infrastructure-config.xml,
        /WEB-INF/spring-config/jps-config-local-remote.xml,
        /WEB-INF/spring-config/pc-config.xml,
        /WEB-INF/spring-config/presentation-config.xml,
        /WEB-INF/spring-config/ppm-config.xml
      </param-value>
    </context-param>
    <!-- beanRefContext.xml is currently provided as part of the sas.svcs.cluster.jar. -->
    <context-param>
       <param-name>locatorFactorySelector</param-name>
       <param-value>classpath:beanRefContext.xml</param-value>
    </context-param>
    <context-param>
       <param-name>parentContextKey</param-name>
       <param-value>config.context</param-value>
    </context-param>
    <!--
      This corresponds to the name of your SoftwareComponent
      object in the SAS Metadata Server.
    -->
    <context-param>
       <param-name>application-name</param-name>
       <param-value>BI Portlets 4.2</param-value>
    </context-param>
    <!-- END Spring Integration -->

    <!-- Cross Site Scripting Sanitizer -->
    <filter>
        <filter-name>SanitizingRequestFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        <init-param>
            <param-name>targetBeanName</param-name>
```

```
            <param-value>sanitizingRequestFilter</param-value>
        </init-param>
        <init-param>
            <param-name>targetFilterLifecycle</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <!-- Cross Site Scripting Filter -->
    <filter-mapping>
        <filter-name>SanitizingRequestFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- logging context separation listener (this should be the FIRST listener) -->
    <listener>
        <listener-class>com.sas.svcs.logging.LoggingContextListener</listener-class>
    </listener>

    <!-- Spring Bootstrap -->
    <listener>
       <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <listener>
       <listener-class>com.sas.portal.plugins.servlet.impl.PortletServletScopeInitializer</listener-class>
    </listener>

    <session-config>
       <session-timeout>30</session-timeout>
    </session-config>

    <!-- only comes into play if container = WebSphere -->
    <jsp-config>
        <taglib>
            <taglib-uri>http://www.sas.com/portlet</taglib-uri>
            <taglib-location>/WEB-INF/taglib/portlet-container-was-taglib.tld</taglib-location>
        </taglib>
    </jsp-config>

</web-app>
```

portlet.xml.orig

This file uses parameters from the custom.properties file to define settings for the portlet.

Example Code 3.3 */Configurable/wars/sample.hellouser.jsr168/WEB-INF/portlet.xml.orig*

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:portlet="http://java.sun.com/xml/ns/portlet"
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
                                 /opt/SUNWps/dtd/portlet.xsd" version="1.0">

    <!-- Sample Hello User JSR168 Portlet -->
    <portlet>
        <description>A sample JSR168 SAS-enabled portlet.</description>
        <!-- *** Name used as PAR file name and displayed to user. *** -->
```

```
        <portlet-name>@webapp.testportlet.contextroot@</portlet-name>
        <display-name xml:lang="en">@webapp.testportlet.display.name@</display-name>
        <portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
        <init-param>
            <name>contextConfigLocation</name>
            <value>/WEB-INF/spring-config/hellouserjsr168portlet.xml</value>
        </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
            <portlet-mode>EDIT</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <supported-locale>en</supported-locale>

        <!-- Portlet Resource Bundle  -->
        <resource-bundle>sample.res.Resources</resource-bundle>

    </portlet>

</portlet-app>
```

jboss-web.xml.orig

This file uses parameters from the custom.properties file to define settings that the portlet uses in the JBOSS server.

Example Code 3.4 */Configurable/wars/sample.hellouser.jsr168/WEB-INF/jboss-web.xml.orig*

```
<?xml version="1.0" encoding="UTF-8" ?>
<jboss-web>
    <security-domain>java:/jaas/PFS</security-domain>
    <context-root>@webapp.testportlet.contextroot@</context-root>
</jboss-web>
```

weblogic.xml.orig

This file uses parameters from the custom.properties file to define settings that the portlet uses in the Oracle WebLogic server.

Example Code 3.5 */Configurable/wars/sample.hellouser.jsr168/WEB-INF/weblogic.xml.orig*

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <jsp-descriptor>
        <jsp-param>
            <param-name>page-check-seconds</param-name>
            <param-value>-1</param-value>
        </jsp-param>
    </jsp-descriptor>
    <container-descriptor>
        <servlet-reload-check-secs>-1</servlet-reload-check-secs>
        <prefer-web-inf-classes>true</prefer-web-inf-classes>
    </container-descriptor>
    <context-root>@webapp.testportlet.contextroot@</context-root>
</weblogic-web-app>
```

Step 7: Create the Static Source Files

Overview of the Static Source Files

The *static source files* are files that do not contain parameter substitutions. These files are placed in the ***source-directory*/Static** directory structure.

edit.jsp

This file creates the page that is displayed for the EDIT mode of the portlet.

Example Code 3.6 */Static/wars/sample.hellouser.jsr168/jsp/edit.jsp*

```
<%@ page import="javax.portlet.*"%>
<%@ page import="java.util.*"%>
<%@ page import="sample.HelloUserJSR168PortletSample"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<portlet:defineObjects/>
<%
    PortletURL actionUrl = renderResponse.createActionURL();
%>

<fmt:setLocale value="${sas_portlet_locale}" />
<fmt:setBundle basename="sample.res.Resources" />

<form name="<portlet:namespace />_form1" method="post" action="<%=actionUrl.toString()%>">
        <div class="portlet-font" >
            <fmt:message key="edit1.txt"/>
            <p>
            <fmt:message key="edit2.txt"/>
            <p>
            <input type="hidden"
                name="<%=HelloUserJSR168PortletSample.FORM_ACTION%>"
                value="<%=HelloUserJSR168PortletSample.ACTION_UPDATE%>" />
            <input class="button" type="submit"
                value='<fmt:message key="ok.txt"/>'
                alt='<fmt:message key="ok.txt"/>'
                />
        </div>

</form>

<c:if test="${PARAM_THEME_IMAGE_PATH != null}">
    <form name="<portlet:namespace />_form2" method="post" action="<%=actionUrl.toString()%>">
            <div class="portlet-font" >
                <fmt:message key="edit3.txt"/>
                <p>
                <input type="hidden"
                    name="<%=HelloUserJSR168PortletSample.FORM_ACTION%>"
                    value="<%=HelloUserJSR168PortletSample.ACTION_REMOVE%>" />
                <input class="button" type="submit"
                    value='<fmt:message key="remove.txt"/>'
                    alt='<fmt:message key="remove.txt"/>'
```

```
                    />
            </div>
    </form>
</c:if>
```

help.jsp

This file creates the page that is displayed for the HELP mode of the portlet.

Example Code 3.7 */Static/wars/sample.hellouser.jsr168/jsp/help.jsp*

```
<%@ page import="java.util.*"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>

<portlet:defineObjects/>

<%
    String filename = renderResponse.encodeURL(renderRequest.getContextPath())+"/images/Note.gif";
    ResourceBundle rb = portletConfig.getResourceBundle(renderRequest.getLocale());
%>

<div class="portlet-font" >
     <%= rb.getString("help1.txt") %>
     <br/>
     <%= rb.getString("help2.txt") %>
</div>

<img src="<%=filename%>" alt="Note.gif Image" title="Note GIF Image" />
```

view.jsp

This file creates the page that is displayed for the VIEW mode of the portlet.

Example Code 3.8 */Static/wars/sample.hellouser.jsr168/jsp/view.jsp*

```
<%@ page import="java.util.*"%>
<%@ page import="sample.HelloUserJSR168PortletSample"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<portlet:defineObjects/>

<%
    String themeImagePath =
      (String)renderRequest.getAttribute(HelloUserJSR168PortletSample.PARAM_THEME_IMAGE_PATH);
%>

<fmt:setLocale value="${sas_portlet_locale}" />
<fmt:setBundle basename="sample.res.Resources" />

<div class="portlet-font" >
    <fmt:message key="view1.txt"/>
    <c:out value="${PARAM_USERNAME}" />
</div>

<c:if test="${PARAM_THEME_IMAGE_PATH != null}">
    <br/>
```

```
    <fmt:message key="view2.txt"/>
    <img src="<%=themeImagePath%>" />
    <br/>
    <fmt:message key="view3.txt"/>
</c:if>

<br/>
```

HelloUserJSR168PortletSample.java

This file creates the main Java source for the portlet.

Note: The filename and file location must match the 'portletClass' property value that is specified in the hellouserjsr168portlet.xml Spring configuration file.

Example Code 3.9 */Static/wars/sample.hellouser.jsr168/source/sample/HelloUserJSR168PortletSample.java*

```
package sample;

import java.io.IOException;

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletConfig;
import javax.portlet.PortletException;
import javax.portlet.PortletMode;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.PortletSession;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

import org.springframework.context.ApplicationContext;
import org.springframework.web.portlet.context.PortletApplicationContextUtils;

import com.sas.framework.commons.resolvers.ThemeResolverInterface;
import com.sas.framework.themes.client.Image;
import com.sas.framework.themes.client.Theme;
import com.sas.framework.themes.client.ThemeServiceInterface;
import com.sas.framework.webapp.util.WebKey;
import com.sas.portal.portlet.toolkit.session.PortletSessionFacade;
import com.sas.portal.portlet.toolkit.session.impl.PortletSessionFacadeImpl;
import com.sas.services.ServiceException;
import com.sas.services.user.UserContextInterface;
import com.sas.svcs.authentication.client.SecurityContext;
import com.sas.svcs.authentication.client.SecurityContextHolder;

public class HelloUserJSR168PortletSample extends GenericPortlet {

    /** used by the JSP to get request parameters and attribute to display users name **/
    public static String PARAM_USERNAME = "PARAM_USERNAME";
    public static String PARAM_THEME_IMAGE_PATH = "PARAM_THEME_IMAGE_PATH";

    public static String FORM_ACTION = "FORM_ACTION";
    public static String ACTION_UPDATE = "UPDATE";
    public static String ACTION_REMOVE = "REMOVE";

    public static String _viewJsp;
```

```
public static String _editJsp;
public static String _helpJsp;
public static String _errorJsp;

public void init()  {
    PortletConfig portletConfig = getPortletConfig();
    /** these have to match property values
        from WEB-INF/spring-config/hellouserjsr168portlet.xml **/
    _viewJsp = portletConfig.getInitParameter("viewPage");
    _editJsp = portletConfig.getInitParameter("editPage");
    _helpJsp = portletConfig.getInitParameter("helpPage");
    _errorJsp = portletConfig.getInitParameter("errorPage");
}

protected void doView(RenderRequest request, RenderResponse response) throws PortletException,
        IOException {
    String forwardPage = null;

    /** get a UserContext from the PortletSessionFacade **/
    PortletSessionFacade sessionFacade = PortletSessionFacadeImpl.getSessionFacade(request);
    UserContextInterface userContext = sessionFacade.getUserContext();
    try {
        String userName = userContext.getPerson().getDisplayName();
        // set request parameters as attributes for use by JSTL
        request.setAttribute(PARAM_USERNAME, userName);
        // get the PortletSession to determine if the image should be displayed
        PortletSession ps = request.getPortletSession();
        // see if theme name exists as the result of a processAction call
        String themeImagePath = (String) ps.getAttribute(PARAM_THEME_IMAGE_PATH);
        request.setAttribute(PARAM_THEME_IMAGE_PATH, themeImagePath);
        forwardPage = _viewJsp;
    } catch (ServiceException e) {
        forwardPage = _errorJsp;
        e.printStackTrace();
    }
    // forward request to jsp
    PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(forwardPage);
    rd.include(request, response);
}

public void doHelp(RenderRequest request, RenderResponse response) throws
    IOException, PortletException {
    PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(_helpJsp);
    rd.include(request, response);
}

public void doEdit(RenderRequest request, RenderResponse response) throws PortletException, IOException {
    PortletSession ps = request.getPortletSession();
    String themeImagePath = (String) ps.getAttribute(PARAM_THEME_IMAGE_PATH);
    request.setAttribute(PARAM_THEME_IMAGE_PATH, themeImagePath);

    PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(_editJsp);
    rd.include(request, response);
}

public void processAction(ActionRequest request, ActionResponse response)
```

```
        throws PortletException {
        PortletSession ps = request.getPortletSession();
        // retrieved from the form action hidden field name EDIT_ACTION
        String action = request.getParameter(FORM_ACTION);
        // ACTION_UPDATE is the value of the hidden field named EDIT_ACTION
        if (action != null) {
            if (action.equals(ACTION_UPDATE)) {
                /** demonstrates accessing WIP Services to get a theme **/
                // set the SecurityContext as a thread local attribute needed by WIP Services
                SecurityContext sc = (SecurityContext)request.getPortletSession()
                    .getAttribute(WebKey.SECURITY_CONTEXT, PortletSession.APPLICATION_SCOPE);
                SecurityContextHolder.setSecurityContext(sc);
                // get the ApplicationContext and ThemeResolver to access the theme service to get the theme
                // name being used
                ApplicationContext appContext =
                   PortletApplicationContextUtils.getWebApplicationContext(request.getPortletSession().getPortletContext());
                ThemeResolverInterface themeResolver =
                   (ThemeResolverInterface)appContext.getBean("themeResolver");
                String themeName = themeResolver.getCurrentThemeName();
                ThemeServiceInterface themeService = (ThemeServiceInterface)appContext.getBean("themeService");
                Theme theme = themeService.getTheme(themeName);
                // get theme resource to display in View (an image defined in SASthemes.xml)
                Image image = theme.getImage("portlet_Help");
                String imagePath = image.getFile();
                ps.setAttribute(PARAM_THEME_IMAGE_PATH, imagePath);
            } else if (action.equals(ACTION_REMOVE)) {
                ps.removeAttribute(PARAM_THEME_IMAGE_PATH);
            }
        }
        response.setPortletMode(PortletMode.VIEW);
    }
}
```

Resources.properties

This file contains all of the strings that are used in the portlet user interface.

Note: The filename and file location must match the value of the 'setBundle' property that is specified in the edit.jsp file.

Example Code 3.10 */Static/wars/sample.hellouser.jsr168/source/sample/res/Resources.properties*

```
javax.portlet.title=Sample JSR168 Test Portlet
javax.portlet.short-title=JSR168 Test Portlet
javax.portlet.keywords=SAS, UserContext, JSR168, test

# View mode text
view1.txt=Hello
view2.txt=This image is a theme resource ('portlet_Help' defined in SASthemes.xml).
view3.txt=Use the Edit mode to remove the image.

# Edit mode text
edit1.txt=You are in Edit mode.
edit2.txt=The OK button will retrieve a resource (image) from the SAS ThemeService.
edit3.txt=The Remove button will remove the Theme image from the View mode.

# Help mode text
help1.txt=You are in Help mode.
```

```
help2.txt=This shows an image that is included with the portlet.

#buttons
ok.txt=Ok
remove.txt=Remove
```

interceptors.xml

This file creates the Spring Framework interceptors for the portlet. *Interceptors* handle user requests for portlet features.

Note: This file must be located in the same directory as hellouserjsr168portlet.xml.

Example Code 3.11 */Static/wars/sample.hellouser.jsr168/WEB-INF/spring-config/interceptors.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "../../dtd/spring-beans.dtd">
<beans>
  <!-- HandlerIntercepters -->
  <bean id="sasPortletInterceptor"
        class="com.sas.portal.portlet.toolkit.interceptor.SASPortletInterceptor" />
  <bean id="backToPortalInterceptor"
        class="com.sas.portal.portlet.toolkit.interceptor.BackToPortalInterceptor" />
</beans>
```

hellouserjsr168portlet.xml

This file creates the Spring Framework integration parameters for the portlet.

Note: The filename and file location must match the 'contextConfigLocation' property value that is specified in the portlet.xml file.

Example Code 3.12 */Static/wars/sample.hellouser.jsr168/spring-config/hellouserjsr168portlet.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "../../dtd/spring-beans.dtd">
<beans>
  <import resource="interceptors.xml" />
    <bean id="HelloUserJSR168PortletSampleBean"
          class="org.springframework.web.portlet.mvc.PortletWrappingController">
      <property name="portletClass">
        <!-- This is the class name for the sample portlet. -->
        <value>sample.HelloUserJSR168PortletSample</value>
      </property>
      <property name="useSharedPortletConfig">
        <value>false</value>
      </property>
      <property name="initParameters">
        <props>
          <prop key="viewPage">/jsp/view.jsp</prop>
          <prop key="editPage">/jsp/edit.jsp</prop>
          <prop key="helpPage">/jsp/help.jsp</prop>
          <prop key="errorPage">/jsp/error.jsp</prop>
        </props>
      </property>
    </bean>
    <!-- Handler Mappings -->
    <bean id="portletModeHandlerMapping"
          class="org.springframework.web.portlet.handler.PortletModeHandlerMapping">
      <property name="interceptors">
```

```
        <list>
          <ref bean="sasPortletInterceptor"/>
        </list>
      </property>
      <property name="portletModeMap">
        <map>
          <entry key="view"><ref bean="HelloUserJSR168PortletSampleBean"/></entry>
          <entry key="edit"><ref bean="HelloUserJSR168PortletSampleBean"/></entry>
          <entry key="help"><ref bean="HelloUserJSR168PortletSampleBean"/></entry>
        </map>
      </property>
    </bean>
</beans>
```

Step 8: Copy the Images for the Portlet

Copy the file Note.gif from ***SAS-Home-Directory*/SASBIPortlets/4.2/Static/wars/sas.biportlets/images** to ***source-directory*/Static/wars/sample.hellouser.jsr168/images**.

Step 9: Prepare the Development Environment

Before you compile and deploy the sample portlet, prepare your development environment by following these steps:

1. Stop the Web application server on which the SAS Information Delivery Portal is running.
2. Back up your metadata content. For more information, see the *SAS Intelligence Platform: System Administration Guide*.
3. Ensure that the SAS Metadata Server is running. The configuration and deployment scripts require access to your metadata.

Step 10: Compile the Source Code

Use the scripting facility to compile your portlet source and build the portlet EAR file:

1. In a command shell, navigate to the ***SAS-config-directory*/Levn/CustomAppData/SampleHelloUserJSR168Portlet** directory.
2. Enter the following command to compile the portlet:

   ```
   cfg compileJSR168Portlet -Dmetadata.connection.passwd="unrestricted-user-password"
   ```

   where *unrestricted-user-password* is the password for the unrestricted user.

   Note: You can submit a password as clear text or as an encoded string from the PWENCODE procedure. For more information, see *Encryption in SAS 9.2*.
3. Check the customconfig.log file to ensure that the script completed successfully.
4. Enter the following command to build the EAR file:

   ```
   cfg buildJSR168Webapps -Dmetadata.connection.passwd="unrestricted-user-password"
   ```

   where *unrestricted-user-password* is the password for the unrestricted user.

Note: You can submit a password as clear text or as an encoded string from the PWENCODE procedure. For more information, see *Encryption in SAS 9.2*.

5. Check the customconfig.log file to ensure that the script completed successfully.

Step 11: Load the Portlet into the BI Portlets Web Application

Use the Portlet Deployment Tool to load the new portlet into the BI Portlets Web application:

1. Modify the properties file for the Portlet Deployment Tool. Update the file ***SAS-config-directory*/Levn/Web/Applications/SASBIPortlets4.2/PortletDeploymentTool/src/build.properties** with the changes that are highlighted:

```
#################################################################
#
# Properties that will change for each portlet ear processed
#
#################################################################

# the webapps ServletContext name. the value needs to be the same
# as the value of the <context-root> tag in the
# META-INF/application.xml file of the portlet ear.
#
# example: SASBIDashboardJsr168
servlet-context-name=HelloUserJSR168PortletSample

# the name of the war file that will be processed inside the portlet
# ear. Note that this version of the Portlet Deployment Tool can only
# process one war file.
#
# example: sas.bidashboardjsr168
web-app-name=sample.hellouser.jsr168

# the app server to which the portlet ear file will be deployed.
# valid entries are jboss, weblogic, and websphere
#
# example: websphere
web-app-server=server-type

# name of the portlet ear file.
#

# example: sas.bidashboardjsr1684.2.ear
portlet-ear-file-name=sample.hellouser.jsr168.ear

# full path to the portlet ear file.
#
# example: c:/SAS/EntBIServer/Levn/Web/Staging/portlet-ear-file-name
portlet-ear-file-path=SAS-config-directory/Levn/Web/Staging/sample.hellouser.jsr168.ear

# work directory for temporary files.
#
# example: c:/temp/pdt
work-dir=SAS-config-directory/Levn/Web/Temp/PDT
```

2. In a command shell, navigate to the ***SAS-config-directory*/Levn/Web/Applications/SASBIPortlets4.2/PortletDeploymentTool/src** directory.
3. Call the level_env.bat script to set environment variables.

   On Windows, enter the following command:

```
..\..\..\..\..\level_env.bat
```

   On UNIX, enter the following command:

```
../../../../../level_env.sh
```

4. Call the launchant.bat script to execute the Portlet Deployment Tool.

   On Windows, enter the following command:

```
..\..\..\..\..\..\Utilities\launchant.bat
```

   On UNIX, enter the following command:

```
../../../../../../Utilities/launchant.sh
```

Step 12: Rebuild the SAS Information Delivery Portal

The EAR file for the SAS Information Delivery Portal contains files that are associated with each portlet. To add the new portlet, you must rebuild the SAS Information Delivery Portal by using the SAS Deployment Manager. For more information, see "Rebuilding the SAS Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.

Step 13: Redeploy the Web Applications

Redeploy the EAR files for the SAS Information Delivery Portal and the BI Portlets Web applications.

The SAS Information Delivery Portal file is located at ***SAS-config-directory*/Levn/Web/Staging/sas.portal4.2.ear**.

The BI Portlets file is located at ***SAS-config-directory*/Levn/Web/Temp/PDT/staging/sas.biportlets4.2.ear**.

For instructions about redeploying Web applications, see "Redeploying Web Applications" in the "Middle-Tier Administration" chapter of the *SAS Intelligence Platform: Web Application Administration Guide*.

Step 14: Restart the Web Application Server

Start the Web application server on which the SAS Information Delivery Portal is deployed. The custom portlet should now be available to the portal as **SampleHelloUserJSR168Portlet**.

Appendix 1

Tips and Best Practices

The following best practices apply to all SAS JSR-168-compliant portlets:

- The HTML that is displayed in the portlet should be as simple as possible to ensure that the portlet is displayed the same way in each portal. For example, avoid using Dojo in JSR 168 portlets.
- The IBM WebSphere Portal expects all portlets to be compatible with Dojo 1.1. In addition, its JSR 168 implementation does not define a namespace. SAS components use Dojo 0.4.3, which does not support namespacing.
- The SAS Information Delivery Portal expects all portlets to be compatible with Dojo 0.4.3. In addition, its JSR 168 implementation does not define a namespace.
- The JSR 168 portlet container in the SAS Information Delivery Portal performs an additional portlet render after each action. Some types of redirect errors might be hidden by the second render. Also, some actions that are performed during renders can cause unwanted changes of state due to the multiple renders.
- The HTML that is displayed in the portlet can use the CSS and images that are included in the SAS Themes Web application.