Developing Portlets for SAS® Information Delivery Portal 4.4

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2015. *Developing Portlets for SAS® Information Delivery Portal 4.4*. Cary, NC: SAS Institute Inc.

Developing Portlets for SAS® Information Delivery Portal 4.4

Copyright © 2015, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a) and DFAR 227.7202-4 and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513-2414.

July 2015

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Contents

What's New in Custom Portlets for SAS Information Delivery Portal 4.4 v

Chapter 1 • Developing Custom Portlets **1**
Introduction to Portlet Development 1
Requirements for Developing Portlets 2
Options for Implementing Portlets 3
Development Environments 3
Development Steps 3
Implementing Portlet Help 12
Creating a PAR File for Deploying the Portlet in an Application 13

Chapter 2 • Hints and Tips for Creating Custom Portlets **15**
Overview 15
Avoiding Namespace Problems 16
Bundling Multiple Portlets into a Single PAR File 16
Testing Portlets 16
Obtaining a User and Session Context 16
Obtaining the User's Name 17
Obtaining the User's Locale 17

Chapter 3 • Using the Portlet API **19**
Using the Portlet API 19

Chapter 4 • Sample Portlets **21**
Overview of Sample Portlets 22
Creating Portlets Using the Testportlet Scripting Facility 22
SampleWelcome: Localized Display Portlet 24
SampleForm: Interactive Form Portlet 32
SampleDisplayURL: Editable Portlet 40
SampleRemote: Remote Portlet 58

What's New in Custom Portlets for SAS Information Delivery Portal 4.4

Changes to Remote Portlets

In the third maintenance release of SAS Information Delivery Portal 4.4, the following enhancements have been made to remote portlets:

- Remote portlets now use local user and session contexts.
- A new RemotePortletContextData class contains the context for the remote portlet.
- A new RemotePortletContextDecoder class creates the context for the remote portlet.
- The testportlet scripting facility for remote portlets includes new **`configure`** and **`startMidTierServers`** parameters for deploying remote portlets.

Chapter 1

Developing Custom Portlets

Introduction to Portlet Development **1**

Requirements for Developing Portlets **2**

Options for Implementing Portlets **3**

Development Environments **3**

Development Steps **3**
Overview: Steps for Developing a Custom Portlet 3
Creating a Portlet Deployment Descriptor 4
Creating Display Resources Files 6
Creating the Presentation JSP Page 7
Creating Action Classes 8

Implementing Portlet Help **12**

Creating a PAR File for Deploying the Portlet in an Application **13**

Introduction to Portlet Development

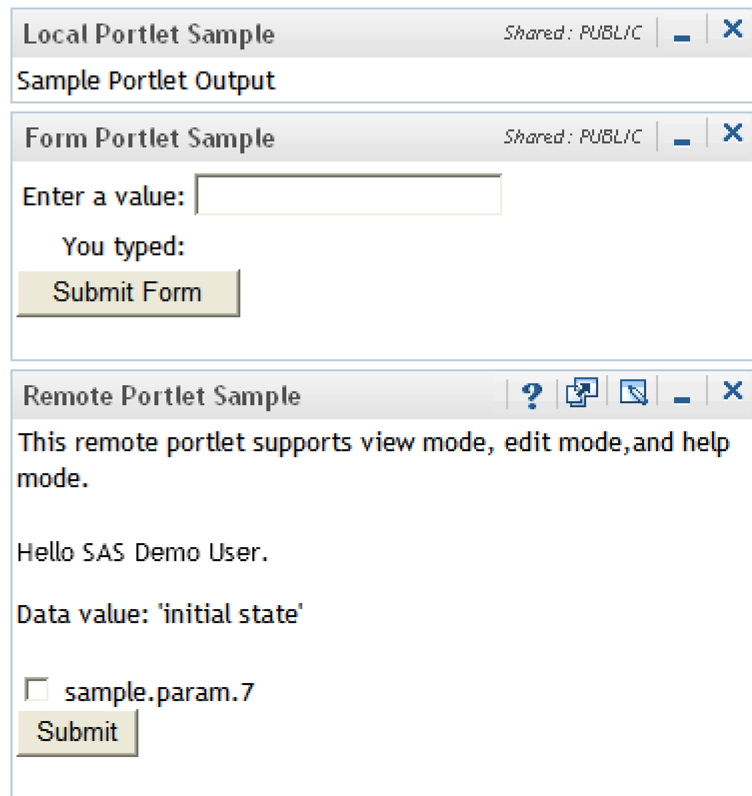
Portlets are the information display components of the SAS Information Delivery Portal. A portlet can process requests from the user and generate dynamic content such as report lists, alerts, workflow notifications, or performance metrics. In addition to a set of standard portlets, the SAS Information Delivery Portal provides a framework that enables you to develop and deploy custom portlets that meet your organization's needs. This framework, which is based on the open-source Struts architecture and conforms to industry-standard Model-View-Controller (Model 2) design patterns, provides the following:

- an execution environment that allows portlets to execute in the portlet container, in the same way that servlets execute in the servlet container. The SAS Information Delivery Portal processes all HTTP requests for portlets, and the session and state information are maintained and shared among portlet actions and across requests.
- support for portlets running remotely in other web technology frameworks, with the option to pass the session and state information to these portlets.
- simplified portlet deployment through the use of the following:
  - a portlet deployment descriptor, which is an XML file that specifies the portlet's actions as well as initialization, path, and access control information.

  - a portlet archive (PAR) file, which includes all of the elements needed to deploy a portlet or series of portlets, including the portlet deployment descriptor, JavaServer Pages (JSP) files, custom Java classes, and associated resources (such as images, resource bundles, and HTML files).

- a set of action and initializer classes, which reduce the need for developing custom programs. These classes perform many commonly used functions, such as displaying the JSP page that is specified in the portlet deployment descriptor.
- access to SAS custom tags and to tags in the Struts development framework to simplify development of JSP pages for your portlets.

Portlets that are created with the framework provided in the SAS Information Delivery Portal have a standard appearance, which includes a title bar that contains icons that link to portlet actions, as shown in the following examples:



Requirements for Developing Portlets

To develop remote portlets, the third maintenance release of SAS Information Delivery 4.4 is required.

Options for Implementing Portlets

The action and initializer classes included in the SAS Information Delivery Portal are designed to handle a portlet's basic function of displaying a single JSP page. However, to meet specialized needs, you can do the following:

- write one or more Java classes that implement the com.sas.portal.portlet.PortletActionInterface. Alternatively, you can extend the provided com.sas.portal.portlet.HTMLPortletAction class to obtain a basic implementation of the interface.
- write Java classes that implement the PortletInitializerInterface, ErrorHandlerInterface, or PostProcessorInterface in the com.sas.portal.portlet package to meet more specialized requirements.

For more information, see Chapter 3, “Using the Portlet API,” on page 19 and Chapter 4, “Sample Portlets,” on page 21.

Development Environments

You can create Java classes, JSP pages, and other supporting files to implement portlets using any Java integrated development environment (IDE). However, SAS AppDev Studio provides templates for creating common types of portlets. For more information about SAS AppDev Studio, see http://support.sas.com/rnd/appdev/.

In addition to interactive development environments, SAS Information Delivery Portal provides a scripting facility for building portlets. To use the scripting facility, you place source files for portlet components in designated directories and execute a configuration script to generate the archive files required to deploy portlets. The use of the scripting facility is illustrated in Chapter 4, “Sample Portlets,” on page 21.

Development Steps

Overview: Steps for Developing a Custom Portlet

To create a custom portlet, follow these steps:

1. Create a portlet deployment descriptor.

   Each portlet that you deploy must be defined in a portlet deployment descriptor. A *portlet deployment descriptor* is an XML file that provides all of the information that the SAS Information Delivery Portal requires to deploy one or more portlets. The file includes information about the portlet's initialization, actions, security settings, and resource paths.

2. Create display resources files.

   The display resources file contains text strings for the portlet's title and description for use in the portlet's metadata. If you create multiple display resources files for different locales, the SAS Information Delivery Portal uses these files to localize the

portlet title and description at the time of deployment, according to the default locale for the SAS Information Delivery Portal.

3. Develop presentation JSP pages.

   Each portlet must have a JSP page to serve as the presentation component.

4. Create action classes.

   You can use the resources of the SAS Information Delivery Portal to develop the following types of action classes for your portlets:

   - initializer classes
   - portlet action classes
   - postprocessing classes
   - error handling classes

5. Implement portlet Help.

   If you want to customize usage instructions for a portlet, you can create an action class with an associated JSP page that contains the Help text. When the user clicks a Help button in the portlet's title bar, the Help appears in a pop-up window.

6. Create a PAR file to deploy in the SAS Information Delivery Portal.

   To automatically deploy a portlet into the SAS Information Delivery Portal, you must provide a PAR file that contains all of the needed files. A PAR file can contain files for one portlet or for multiple related portlets.

For examples of fully developed portlet code, see Chapter 4, “Sample Portlets,” on page 21.

Creating a Portlet Deployment Descriptor

Overview of Portlet Deployment Descriptors

For each PAR file that you create for deployment in the SAS Information Delivery Portal, you must create a portlet deployment descriptor. The portlet deployment descriptor is an XML file that provides all of the information that the SAS Information Delivery Portal needs to deploy the portlets that are contained in the PAR file. The portlet deployment descriptor file must be named portlet.xml.

A PAR file, and its associated portlet deployment descriptor, can contain one portlet or it can contain multiple related portlets; there is no limit to the number of portlets that a PAR file and its associated descriptor can contain.

In addition, a PAR file and its associated portlet deployment descriptor can contain local portlets, remote portlets, or a combination of local and remote portlets.

To create a portlet deployment descriptor, use the element tags that are defined in the portlet deployment descriptor document type definition (DTD). You can view the portlet deployment descriptor DTD at ***`SAS-installation-directory`*** **`\SASInformationDeliveryPortal\4.4\Static\wars\sas.portal\WEB-INF\classes\portlet.dtd`**.

The following examples show portlet deployment descriptors for a local portlet and for a remote portlet. You can use these examples as templates for creating deployment descriptors for your own portlets.

After you create the deployment descriptor file, include it in the PAR file that you create for your portlet or group of portlets. For more information, see "Creating a PAR File for Deploying the Portlet in an Application" on page 13.

Example Deployment Descriptor for a Local Portlet

A *local portlet* is a portlet that meets the following criteria:

- The portlet is deployed within the SAS Information Delivery Portal.
- The portlet executes inside the portlet container.
- The portlet consumes the computing resources (for example, CPU, memory, and disk storage) of the server machine on which the portal container runs.
- The portlet can include resources such as web pages, images, resource bundles, and Java classes that are deployed inside the SAS Information Delivery Portal.

You can use the following example as a template for creating portlet deployment descriptors for your own local portlets:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE portlets SYSTEM "http://www.sas.com/idp/portlet.dtd">
<portlets>

   <local-portlet name="simplejsp" title="SimpleJspPortlet"
         icon="images/ndd.jpg">
      <localized-resources locales="en" />
      <deployment scope="user" autoDeploy="false"
         userCanCreateMore="true">
      </deployment>

      <initializer-type>
         com.sas.portal.portlets.JspPortlet.JspPortletInitializer
      </initializer-type>

      <init-param>
         <param-name>display-page</param-name>
         <param-value>simpleJspTest.jsp</param-value>
      </init-param>

      <portlet-path>/sample/portlets</portlet-path>
      <portlet-actions>
         <portlet-action name="display" default="true">
            <type>com.sas.portal.portlets.JspPortlet.JspPortlet</type>
         </portlet-action>
      </portlet-actions>
   </local-portlet>

</portlets>
```

Example Deployment Descriptor for a Remote Portlet

Remote portlets are portlets that execute outside of the portal container. You can use remote portlets to incorporate data from external applications into the SAS Information Delivery Portal. When a user interacts with a remote portlet, the remote portlet appears to be the same as a local portlet.

Many of the elements in the portlet deployment descriptor DTD relate only to local portlets. Therefore, a portlet deployment descriptor for a remote portlet requires fewer elements than a descriptor for a local portlet.

You can use this example as a template for creating portlet deployment descriptors for your own remote portlets:

```
<!DOCTYPE portlets SYSTEM "http://www.sas.com/idp/portlet.dtd">
<portlets>

   <remote-portlet name="MyRemotePortlet" title="MyRemotePortlet">
      <localized-resources locales="en" />
      <deployment scope="user" autoDeploy="false"
         userCanCreateMore="true">
      </deployment>

      <portlet-path>/sample/portlets/remote</portlet-path>
      <portlet-actions>
         <portlet-action name="display" default="true">
            <url>http://d9999.mycompany.com:8080/test.html</url>
         </portlet-action>
      </portlet-actions>
   </remote-portlet>

</portlets>
```

Creating Display Resources Files

A *display resources file* is a file that contains key=value statements to define text strings for a portlet's title and description. You can create display resources files for the following purposes:

- to specify a description for your portlet. If you do not provide a display resources file, the SAS Information Delivery Portal uses the portlet's name to create a default description.

  Note: The <local-portlet> and <remote-portlet> elements of the portlet deployment descriptor contain a description attribute. However, this description is used only for internal documentation. It is not displayed to users.

- to enable the SAS Information Delivery Portal to localize the portlet title and description at the time of deployment, according to the default locale for the SAS Information Delivery Portal. When the portlet is first deployed, the deployment process determines which locale to use. Based on this locale, the deployment process uses the title and description from the appropriate display resources file to create metadata and register the portlet in the SAS Metadata Repository. The following rules apply:

  - For editable portlets, the portlet name that is displayed in the drop-down list of portlet selections is localized based on the browser locale when the user first logs in to the portal.

    Note: The locale does not apply to new portlet instances created from the template because the user enters an explicit name and description that are stored in metadata for the portlet instance.

  - For portlets with user scope and the autoDeploy="true" attribute in the deployment descriptor, the browser locale when the user first logs in to the portal

determines which display resources file is selected to provide the title and description for the portlet.

- For portlets with group scope and the autoDeploy="true" attribute in the deployment descriptor, the locale used to start the web application server determines which display resources file is selected to provide the title and description for the portlet. Starting the web application server with one locale specified and then starting it again with a different locale specified results in two instances of the portlet.

If your portlet is deployed in only one locale, then the display resources files can be omitted. The portlet name in the default locale is used.

Note: The SAS Metadata Repository cannot store multiple localized values for metadata. Therefore, the portlet title and description are translated only into the default locale for the SAS Information Delivery Portal. They cannot be translated based on the user's locale preference.

If your portlet does not include any display resources files, the portlet deployment mechanism sends a warning message to the server log. The message indicates that no localized title or description can be found.

To create display resources files, follow these steps:

1. Create a separate file for each language (or each country and language combination) that you need to support. In each file, use key=value statements to define text strings for portlet.title and portlet.description, as in the following examples:

```
portlet.title=Welcome Portlet
portlet.description=Welcome Portlet
```

```
portlet.title=Portlet de bienvenida
portlet.description=Portlet de bienvenida
```

2. Name the files as follows:
   - Use the base name portletDisplayResources.properties.
   - If you are creating files for multiple locales, append each file's name with the appropriate locale identifier (for example, portletDisplayResources_en.properties for English, portletDisplayResources_fr.properties for French, and so on). The file for the default locale does not need to have a locale identifier.
3. Place the files in the `/`***`portlet-name`***`/classes` directory of the PAR file.
4. Add the locale identifier for each supported locale to the locales attribute value of the <localized-resources> element in the portlet deployment descriptor (portlet.xml) file.

Note: The properties files must have ISO-8859–1 encoding. Any characters that are not part of ISO-8859–1 must be expressed as Unicode character codes using the format **`\u`*`xxxx`***, where *xxxx* is the four-digit Unicode identifier.

Creating the Presentation JSP Page

JavaServer Page (JSP) pages are the presentation components of local portlets. Because you can define a local portlet's initialization, actions, security settings, and resource paths in the portlet deployment descriptor, the JSP page does not need to contain this information.

In developing the JSP page, you can use the following tags:

- tags from the JSP Standard Tag Libraries (JSTL).

- tags from the Struts tag libraries.
- SAS custom tags, which are available if your site has licensed SAS AppDev Studio in addition to the SAS Information Delivery Portal. For information about these tags, see the SAS Custom Tags Reference page on the SAS AppDev Studio Developer's Site at http://support.sas.com/rnd/appdev.

When you create a JSP page for a portlet, the only requirements are the following:

- The JSP page must be an HTML fragment:
  - The page must not contain starting and ending <HTML>, <HEAD>, or <BODY> tags.
  - The page must be able to be displayed inside a table cell in an HTML document.
- If the JSP page includes custom tags from a tag library, you must include a taglib directive before the first use of a tag from that library. For the JSTL format tag library, use the following taglib directive:

  ```
  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
  ```

- You must include a UTF-8 directive if you want the JSP page to provide full support for internationalization. This directive encodes all user input in the 8-bit Unicode Transformation Format, which supports all of the world's languages, including those that use non-Latin1 characters.

  Note: The SAS Information Delivery Portal supplies this directive when it displays portlets on a portal page. However, you must supply the directive to ensure correct internationalization when your portlet is displayed from the Search Results panel. Consider making your portlet actions extend the HTMLPortletAction class, because that class supplies the directive.

  The syntax for the UTF-8 directive is:

  ```
  <%@ page contentType= "text/html; charset=UTF-8"%>
  ```

- The JSP cannot import Java classes from the portlet's PAR file.

Creating Action Classes

Overview of Creating Action Classes

You can use the resources of the SAS Information Delivery Portal to develop the following types of action classes for your local portlets:

- Initializer classes
- Portlet action classes
- Postprocessing classes
- Error handling classes

The Portlet API includes classes that you can use to create your own action classes for custom portlets. For a summary of these classes, see Chapter 3, “Using the Portlet API,” on page 19. For more information, see SAS Information Delivery Portal API documentation at http://support.sas.com/94m3api.

Any action classes that you develop must be defined in the portlet's deployment descriptor file and included in the portlet's PAR file. These classes cannot be accessed by the portlet's JSP pages.

Note: You can also develop classes other than action classes for your portlet and include them in the portlet's PAR file. These classes do not need to be defined in the portlet deployment descriptor file. However, the additional classes can be accessed only by the action classes. They cannot be accessed from the portlet's JSP pages.

Thread Safety

Portlet actions, like Struts actions, are multithreaded. There is only a single instance of your PortletAction subclass, and you must make your actions thread-safe, as follows:

- You cannot use class properties to share values between member methods.
- If you use member methods, be sure to pass all values through the method's signature. The signature passes all values through the thread-safe stack.

Creating an Initializer Action Class

When you develop a local portlet, you can implement an initializer class that runs before the portlet is displayed for the first time on a SAS Information Delivery Portal page. The initializer does not execute again if the user interacts with your portlet or with other portlets on the same page. It also does not execute again if the user navigates to another page and then back again. However, the initializer does run again if the user logs off, logs on again, and displays the page that contains the portlet.

Uses for an initializer might include reading initial parameters that are specified in your portlet's deployment descriptor file (portlet.xml) or connecting to an external resource such as a database.

The SAS Information Delivery Portal is delivered with a default initializer class named JspPortletInitializer, which requires a parameter named display-page. The initializer places the value of this parameter in the PortletContext object so that it can be used by the portlet's action class. To pass additional parameters, you need to create your own initializer class.

When you create an initializer class, ensure that the following steps have been taken:

- The class must be specified in the <initializer-type> element of the portlet's deployment descriptor file (portlet.xml).
- The class must implement com.sas.portal.portlet.PortletInitializerInterface.

The com.sas.portal.portlet.PortletInitializerInterface class includes one method named initialize(). The following objects are passed to the initialize() method:

java.util.Properties
: contains all of the initial parameters that are specified in your portlet's deployment descriptor. If your portlet's action class or JSP page requires access to these parameters, you should place them in the portlet context object using its setAttribute() method.

com.sas.portal.portlet.PortletContext
: provides a getter method for the HttpSession object so that you can access or set session attributes.

The following example shows an initialize() method that places initial parameters into the portlet context:

```
/**
 * Puts initial properties into the PortletContext object.
 * These come from the portlet.xml.
 * @param initProperties a Properties object
 * @param context the PortletContext for this portlet
 */
```

```
public void initialize(Properties initProperties,
      PortletContext context) {
   context.setAttribute("display-page",
      initProperties.getProperty("display-page"));
   context.setAttribute("image-location",
      initProperties.getProperty("image-location"));
}
```

Creating a Portlet Action Class

When developing a local portlet, you can implement one or more action classes for the portlet. If you use an action class, then the following requirements must be met:

- You must specify the class in your portlet deployment descriptor file (portlet.xml).
- The class must implement com.sas.portal.portlet.PortletActionInterface.
- The class can extend DefaultPortletAction or HTMLPortletAction in com.sas.portal.portlet. The DefaultPortletAction and HTMLPortletAction contain two simple methods for setting and getting an instance of com.sas.portal.portlet.PortletActionInfoInterface, as shown in the following example:

```
public void setInfo(PortletActionInfoInterface pai) {
   _actionInfo = pai;
}public PortletActionInfoInterface getInfo() {
   return _actionInfo;
}
```

The primary method, named service(), runs every time the action is executed. For the portlet's display action, this occurs before the portlet is displayed and every time the portlet is redisplayed. For example, it runs after a user interacts with the portlet or with a different portlet on the same page.

The service() method is provided with the HttpServletRequest, HttpServletResponse, and PortletContext objects. From the PortletContext object, you can obtain the HttpSession object, which provides access to many important servlet objects.

Your service() method must return a string representing a valid URL for the portlet. Typically, the URL is the name of the portlet's JSP page. If your initializer places the display-page property of the portlet deployment descriptor file into the PortletContext object, then you can obtain the URL, as in the following example:

```
String url = (String) context.getAttribute("display-page");
```

If user interaction with your portlet requires a different URL string, then you can return that URL instead.

The service() method can handle any type of exception subclass that is thrown by code within your action. If your portlet action needs to throw an exception, then you can use the portlet error handler. For more information, see "Creating an Error Handling Action" on page 11.

Creating a Postprocessing Action Class

The com.sas.portal.portlet.PostProcessorInterface is available for implementing activity that should occur when a local portlet is no longer on display. Like other parts of the portlet architecture, it must be defined in your deployment descriptor file. You can use the post-processor phase to free resources that you attached to in the portlet initializer. You can also remove HttpSession attributes that were set in the initializer or action. This is especially important to consider because multiple copies of your portlet could exist on other SAS Information Delivery Portal pages or even on the same page.

Creating an Error Handling Action

The com.sas.portal.portlet.ErrorHandlerInterface is available for handling any errors that your local portlets encounter. This interface has one method, service(). The service() method has the same arguments as the service() method of the PortletActionInterface, plus an additional object named Exception.

If you specify an error handler in your portlet deployment descriptor file (portlet.xml), the error handler is called if the portlet action throws an exception. You can direct your error handler to send messages to the server log and to return a URL string representing an error page for the user to view.

If your portlet initializer encounters an exception, the error handler is not called. If you want to ensure that the error handler executes, you can store the exception object in the portlet context. Then, in your action class's service() method, you can get the exception object out of the context and re-throw it. In the following example, this code is put into a method that should be called at the start of the action's service() method:

```
/**
 * Check the PortletContext for an exception object. If
 * present, throw it so that the error handler is executed.
 * @param context the PortletContext
 */
private static void errorCheck(PortletContext context)
      throws Exception {
   Exception e = (Exception) context.getAttribute("PORTLET_EXCEPTION");
   if (e != null) {
      throw e;
   }
}
```

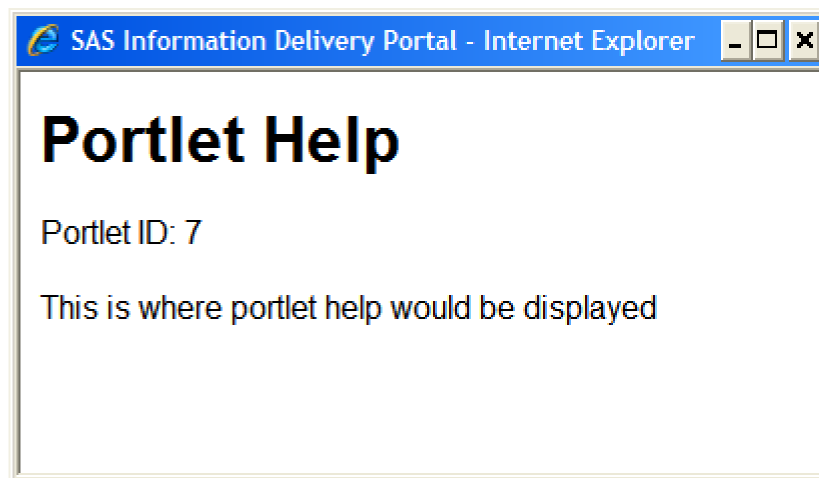
The following example shows a simple error handler that logs the exception and calls a static error page. The error page supplies a general error message from the portlet's localized resource bundles.

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sas.portal.Logger;
import com.sas.portal.portlet.PortletContext;
import com.sas.portal.portlet.ErrorHandlerInterface;
import com.sas.portal.portlet.NavigationUtil;

/**
 * Error handler for some portlets.
 * It logs the exception and returns ErrorPage.jsp
 * for the portlet to display.
 */
public class MyErrorHandler implements ErrorHandlerInterface {
   private final String _loggingContext = this.getClass().getName();

   /**
   * Returns the URL for the portlet controller to call. This is the
   * name of the error page JSP.
   * @param request the HttpServletRequest
   * @param response the HttpServeltResponse
   * @param context the PortletContext
   * @param exception the exception thrown by a portlet action
   * @return the URL to call
   */
```

```
    public String service(HttpServletRequest request, HttpServletResponse
          response, PortletContext context, Exception thrownException) {

       // Prepare the localized resources for use by the jsp.
       try {
          NavigationUtil.prepareLocalizedResources(
             "com.mycompany.portlets.Resources", request, context);
       }
       catch (java.io.IOException ioe) {
          Logger.error(ioe.getMessage(), _loggingContext, ioe);
       }
       Logger.error(thrownException.getMessage(), _loggingContext,
          thrownException);
       return "ErrorPage.jsp";
    }
 }
```

Implementing Portlet Help

You can easily implement Help for a custom portlet. If you implement Help for a portlet, then a Help icon appears in the portlet's title bar. When a user clicks the icon, the portlet Help appears in a resizable, scrollable window that is by default 400 pixels wide and 200 pixels high, as shown in the following example:



To implement portlet Help, use these steps:

1. Create an action class to display the JSP page for the Help (or, if you want, you can use an instance of com.sas.portal.portlet.JspPortlet). The following example shows the code for an example of a custom action class to display portlet Help:

```
package com.sas.portal.portlets.welcome;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sas.portal.portlet.HTMLPortletAction;
import com.sas.portal.portlet.PortletContext;

public class HelpAction extends HTMLPortletAction {
```

```
        public String service(HttpServletRequest request,
              HttpServletResponse response, PortletContext context)
              throws Exception {
           return "help.jsp";
        }
     }
```

2. Create a JSP page that contains the Help text. The JSP page must have the following characteristics:
   - The JSP page must be an HTML fragment. That is, it must not contain starting and ending <HTML>, <HEAD>, or <BODY> tags.
   - The JSP page must have the filename help.jsp. The example JSP page for the Help window that was shown previously consists of the following lines:

     ```
     <h1>Portlet Help</h1>
     <p>Portlet ID: <%= pid %></p>
     <p>This is where portlet Help would be displayed</p>
     ```

3. In the portlet's deployment descriptor (portlet.xml) file, add a <portlet-action> element for the action class. Set the element's Help attribute to true.

The following example of a <portlet-action> element for a portlet uses a custom action to display its Help:

```
<portlet-action name="help" help="true">
   <type>com.sas.portal.portlets.welcome.HelpAction</type>
</portlet-action>
```

The following example of a <portlet-action> element for a portlet uses an instance of com.sas.portal.portlet.JspPortlet to display its Help:

```
<portlet-action name="help" help="true">
   <type>com.sas.portal.portlet.JspPortlet</type>
</portlet-action>
```

Creating a PAR File for Deploying the Portlet in an Application

A PAR file contains all of the files needed to deploy a portlet or a group of portlets into the SAS Information Delivery Portal. A single PAR file can contain files for multiple portlets. A PAR file can contain local portlets, remote portlets, or a combination of local and remote portlets.

For correct deployment, the portlet files must be archived in a PAR file using the following directory structure:

/ (root)
: contains the portlet deployment descriptor file, which must be named portlet.xml.

/*portlet-name*
: contains associated subdirectories for each portlet that is defined in portlet.xml. The directory name must match the name of the portlet as specified in the name attribute of the <local-portlet> or <remote-portlet> element in portlet.xml.

/*portlet-name*/classes

/*portlet-name*/content

/portlet-name

Chapter 2
Hints and Tips for Creating Custom Portlets

Overview **15**
Avoiding Namespace Problems **16**
Bundling Multiple Portlets into a Single PAR File **16**
Testing Portlets **16**
Obtaining a User and Session Context **16**
Overview 16
Obtaining a Local User and Session Context 17
Obtaining the User’s Name **17**
Obtaining the User’s Locale **17**

Overview

This chapter provides tips and best practices for developing portlets, including the following:

- avoiding namespace problems
- bundling multiple portlets into a single PAR file
- testing portlets
- obtaining a user or session context
- obtaining the user's name
- obtaining the user's locale

Note: The code samples in this chapter require that you include the appropriate import statements at the top of your program. Here are some examples:

```
import com.sas.preferences.SASProfileInterface;
import com.sas.services.information.metadata.PersonInterface;
import com.sas.services.session.SessionContextInterface;
import com.sas.services.user.UserContextInterface;
import com.sas.services.user.UserServiceInterface;
```

Avoiding Namespace Problems

The portlet namespace consists of the path (with underscores in place of slashes) and the portlet's name. For example, a portlet with the name simpleJSP and a path of **`/mycompany/portlets`** would be deployed as **`_mycompany_portlets_simpleJSP`**.

To avoid namespace problems, do the following:

- Use a standard naming convention for portlet paths.
- Avoid using the _SAS namespace.

Bundling Multiple Portlets into a Single PAR File

When you need to deploy multiple portlets, define the portlets in a single portlet deployment descriptor (portlet.xml) file and bundle the portlets into a single portlet archive (PAR) file if possible. This practice improves performance because only one PAR file needs to be opened and only one portlet descriptor file needs to be read.

Testing Portlets

To test and debug a local portlet that you have developed, deploy it into a staging area (that is, a test installation of the SAS Information Delivery Portal). After the portlet has been verified and tested, deploy it into the production environment.

For remote portlets, test and debug the web application that is called by the portlet by using the application's direct URL. After the application has been verified and tested, deploy the remote portlet into the SAS Information Delivery Portal's production environment.

Obtaining a User and Session Context

Overview

The SAS Foundation Services user context and session context provide a way to manage information while a user is authenticated to the SAS Intelligence Platform middle tier. For more information about using SAS Foundation Services, see the class documentation at http://support.sas.com/94m3api.

A local portlet has access to the SAS Information Delivery Portal's local user and session contexts. By using the local user and session contexts that are shared with the portal, all Java objects created when accessing SAS Foundation Services are local objects that are shared by the portal and the portlets.

A remote portlet is implemented as a separate web application and does not have access to the SAS Information Delivery Portal's local user and session contexts. Instead, the web application receives its own local user and session contexts for the same user through single sign-on.

The preferred ways to obtain the user and session contexts are from the HttpSession. The easiest way to obtain the HttpSession in portlet code is from the request, as in the following example:

```
HttpSession session = request.getSession();
```

If the request is not available, the HttpSession can also be obtained from the portlet context, as in the following example:

```
HttpSession session = portletContext.getHttpSession();
```

Obtaining a Local User and Session Context

The following example obtains both the local user and session contexts from the HttpSession:

```
UserContextInterface userContext =
   (UserContextInterface)session.getAttribute(CommonKeys.USER_CONTEXT);
SessionContextInterface sessionContext =
   (SessionContextInterface)session.getAttribute(CommonKeys.SESSION_CONTEXT);
```

Obtaining the User's Name

In a portlet initializer or action class, you can obtain the display name of the user that is logged on from the user context. For information about obtaining the user context, see "Obtaining a User and Session Context" on page 16.

```
IdentityInterface aPerson = userContext.getPerson();
String name = aPerson.getDisplayName();
```

Obtaining the User's Locale

In a portlet initializer or action class, you can obtain the user's locale from the user context. For information about obtaining the user context, see "Obtaining a User and Session Context" on page 16.

The following code obtains the SAS profile from the user context and then obtains the locale from the instance of the SASProfileInterface:

```
ProfileInterface profile = userContext.getProfile();
SASProfileInterface sasProfile =
  (com.sas.preferences.SASProfileInterface)profile.getProfile("SAS");
Locale locale = sasProfile.getLocale();
```

Note: The locale is null if no value was available from the SAS profile. In this case, use the HttpServletRequest.getLocale() method instead.

Chapter 3
Using the Portlet API

Using the Portlet API 19

Using the Portlet API

The Portlet API provides access to classes that provide the SAS Information Delivery Portal's navigation and request processing functions. For more information about the API, see the class documentation at http://support.sas.com/94m3api.

The following classes are especially useful in creating custom portlets:

com.sas.portal.portlet.DefaultPortletAction
: Extend this class to create your own portlet actions. For more information, see "Creating Action Classes" on page 8.

com.sas.portal.portlet.ErrorHandlerInterface
: Use this interface to handle errors that your portlet encounters. For more information, see "Creating an Error Handling Action" on page 11.

com.sas.portal.portlet.HTMLPortletAction
: Extend this class to create your own portlet actions. Possible uses include the following:

- correctly displaying non-Latin1 character sets when a portlet is displayed in preview mode. For an example of this use, see "Step 4: Create the Action Class" on page 28.
- preparing URLs for actions within an interactive JSP form and populating a JavaBean with parameters from a JSP form. For an example of these uses, see "Step 4: Create the Action Class" on page 36.

com.sas.portal.portlet.PortletContext
: Use this interface to obtain a local session context, which you can use to pass information from one local portlet to another. For examples, see "Obtaining a User and Session Context" on page 16.

com.sas.portal.portlet.NavigationUtil
: Use this class to do the following:

- create URLs for buttons on your JSP pages (for example, **OK** and **Cancel**).
- obtain a portlet's resource bundles to create a localization context for your portlet's JSP page.

com.sas.portal.portlet.PortletActionInterface
: Use this interface to develop an action class for your portlet. For more information, see “Creating Action Classes” on page 8.

com.sas.portal.portlet.PortletInitializerInterface
: Use this interface to develop an initializer class, which runs before your portlet is displayed for the first time on a portal page. Possible uses include

- reading initial parameters that are specified in your portlet's deployment descriptor file (portlet.xml).
- connecting to an external resource such as a database. For more information, see “Creating an Initializer Action Class” on page 9.

com.sas.portal.portlet.PostProcessorInterface
: Use this interface to develop a postprocessor class that runs when your portlet is no longer displayed. Possible uses include the following:

- freeing resources that were used in the portlet initializer.
- removing HttpSession attributes that were set in the portlet initializer or portlet action. This is especially important to consider because multiple copies of your portlet could exist on other portal pages or even on the same page.

For more information, see “Creating a Postprocessing Action Class” on page 10.

Chapter 4

Sample Portlets

Overview of Sample Portlets **22**

Creating Portlets Using the Testportlet Scripting Facility **22**

SampleWelcome: Localized Display Portlet **24**
Overview: Steps for Creating the SampleWelcome Portlet 24
Step 1: Create the Portlet Configuration and Source Directories 25
Step 2: Create the Portlet Deployment Descriptor 26
Step 3: Create the Display Page 28
Step 4: Create the Action Class 28
Step 5: Create the Resource Bundles 29
Step 6: Create Translated Titles and Descriptions 30
Step 7: Compile the Portlet Code 30
Step 8: Create the PAR File and Deploy and Test the Portlet 31

SampleForm: Interactive Form Portlet **32**
Overview: Steps for Creating the SampleForm Portlet 32
Step 1: Create the Portlet Configuration and Source Directories 32
Step 2: Create the Portlet Deployment Descriptor 34
Step 3: Create the Display Page 35
Step 4: Create the Action Class 36
Step 5: Create a JavaBean to Return User Input 38
Step 6: Create a Title and Description for the Portlet 38
Step 7: Compile the Portlet Code 39
Step 8: Create the PAR File and Deploy and Test the Portlet 39

SampleDisplayURL: Editable Portlet **40**
Overview: Steps for Creating the SampleDisplayURL Portlet 40
Step 1: Create the Portlet Configuration and Source Directories 41
Step 2: Create the Portlet Deployment Descriptor 42
Step 3: Create the Display Pages for the Portlet and the Editor 44
Step 4: Create the Action Classes 46
Step 5: Create the Resource Bundle 56
Step 6: Create a Title and Description for the Portlet 56
Step 7: Compile the Portlet Code 57
Step 8: Create the PAR File and Deploy and Test the Portlet 58

SampleRemote: Remote Portlet **58**
Overview: Steps for Creating the SampleRemote Portlet 58
Step 1: Create the Portlet Configuration and Source Directories 59
Step 2: Create the Context Descriptor and the Enterprise
Application Deployment Descriptor 60
Step 3: Create the Web Application Deployment Descriptor 61

Step 4: Create the Spring Framework Configuration Files 62
Step 5: Create the Display Pages for the Web Application 65
Step 6: Create the Controller Servlet Class 68
Step 7: Create the Portlet Deployment Descriptor 71
Step 8: Create a Title and Description for the Portlet 72
Step 9: Create the Application Metadata for the Portlet 72
Step 10: Compile the Remote Portlet 73
Step 11: Create the EAR and PAR Files and Deploy and Test the Portlet 74

Overview of Sample Portlets

This chapter includes complete code for portlet deployment descriptors, JSP pages, resource files, and action classes as applicable for the following sample portlets:

SampleWelcome
: is a simple display portlet that has no interactive capabilities. Because it is internationalized, it displays text in the user's locale (language and country) preference.

SampleForm
: is an interactive form portlet that accepts free-form input and displays it back to the user.

SampleDisplayURL
: is a portlet from which users can create their own portlet instances that display HTML content from any URL. It is a simplified version of the standard DisplayURL portlet template that is delivered with the SAS Information Delivery Portal. This sample provides an Edit mode that enables users to specify the URL to display.

SampleRemote
: is a remote portlet that executes a web application that displays the name of the user who is logged on to the SAS Information Delivery Portal. The portlet also displays text that can be edited by the user.

For more information about a specific portlet development task, see “Development Steps” on page 3.

Creating Portlets Using the Testportlet Scripting Facility

The examples in this chapter use the portlet development scripting facility provided with the SAS Information Delivery Portal. 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 9.4 Versioned JAR Repository.

The following steps provide an overview of creating a portlet using the scripting facility. The instructions for creating the sample portlets in this chapter include detailed examples of using the scripting facility.

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

1. 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 match the name of the portlet that is specified in the name attribute of the **<local-portlet>** or **<remote-portlet>** element in the portlet deployment descriptor (portlet.xml) file.
   - The portlet name must be unique.
2. Copy the contents of the **testportlet** directory to the new portlet configuration directory.
3. Create a source directory for the code associated with the portlet. This directory is referred to in subsequent instructions as the *portlet source 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.
5. Run the configuration script, cfg, to create the source directory structure for building the portlet.

   Note: After running the script, review the customconfig.log file for errors. This file is located in the directory where the script was run.

   For local portlets, the following source directory structure is created:

   ```
   portlet-source-directory/Configurable/pars/archive-name
   portlet-source-directory/Picklists/pars/archive-name
   portlet-source-directory/Static/lib
   portlet-source-directory/Static/pars/archive-name/context-root/content
   portlet-source-directory/Static/pars/archive-name/context-root/source
   portlet-source-directory/Static/pars/archive-name/context-root/classes
   ```

   For remote portlets, the following source directory structure is created:

   ```
   portlet-source-directory/Config/Deployment/Metadata
   portlet-source-directory/Configurable/ears/archive-name/META-INF
   portlet-source-directory/Configurable/pars/archive-name
   portlet-source-directory/Configurable/wars/archive-name/META-INF
   portlet-source-directory/Configurable/wars/archive-name/WEB-INF
   portlet-source-directory/Picklists/wars/archive-name
   portlet-source-directory/Static/lib
   portlet-source-directory/Static/ears/archive-name/META-INF
   portlet-source-directory/Static/pars/archive-name/context-root/classes
   portlet-source-directory/Static/wars/archive-name/jsp
   portlet-source-directory/Static/wars/archive-name/source
   portlet-source-directory/Static/wars/archive-name/WEB-INF/classes
   portlet-source-directory/Static/wars/archive-name/WEB-INF/spring-config
   ```

   The **/Configurable** and **/Static** directory hierarchies 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 stores 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 stores additional JAR files needed at compile time.

6. Load source files for portlet components into the source directory structure.
7. Create a 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 other JAR files to the project.
9. Run the configuration script to compile Java classes.

   Note: After running the script, review the customconfig.log file for errors. This file is located in the directory where the script was run.
10. Stop the application server on which the SAS Information Delivery Portal is deployed so that development of the new portlet will not affect the running system.
11. Run the configuration script to build the appropriate archive files for the portlet. This step creates a PAR file in the SAS Information Delivery Portal's **`Exploded`** and **`Deployed`** portlet directories. For remote portlets, it also creates the remote portlet web application.
12. For remote portlets only: Run the configuration script to deploy the web application EAR file to the web application server.

    Note: After running the script, review the customconfig.log file for errors. This file is located in the directory where the script was run.
13. For remote portlets only: Run the configuration script to update and restart the web application server.

    Note: After running the script, review the customconfig.log file for errors. This file is located in the directory where the script was run.
14. Start the application server where the SAS Information Delivery Portal is deployed. The custom portlet should now be available to the portal.

Note: In order to use the scripting facility, you must have the unrestricted user password for your SAS installation.

SampleWelcome: Localized Display Portlet

Overview: Steps for Creating the SampleWelcome Portlet

The SampleWelcome portlet displays localized text using the user's locale (language and country) preference and is not interactive. The SampleWelcome portlet is a local portlet that runs inside the portlet container.

To create the SampleWelcome portlet, follow these steps:

1. Create the portlet configuration and source directories.
2. Create the portlet deployment descriptor (portlet.xml).
3. Create the display page (Welcome.jsp).
4. Create the actions class (WelcomeAction.java).
5. Create resource bundles to support different locales.

6. Create translated portlet titles and descriptions to support different locales.
7. Compile the portlet code.
8. Create the PAR file and deploy and test the portlet.

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

Step 1: Create the Portlet Configuration and Source Directories

Follow these steps to create a configuration and source directory structure for building the portlet:

1. Create a configuration directory for the portlet named **SampleWelcome** under the ***SAS-configuration-directory*/Lev1/CustomAppData** directory. This directory is referred to as *portlet-configuration-directory* in the code and descriptions for this portlet.
2. Copy the contents of the **testportlet** directory into this new **SampleWelcome** directory.
3. Create a source code directory for the portlet named **Source** under the ***SAS-configuration-directory* /Lev1/CustomAppData/SampleWelcome** directory. This directory is referred to as *portlet-source-directory* in the code and descriptions for this portlet.
4. Edit the custom.properties file in the **SampleWelcome** directory as follows.

   Note: Be sure to substitute the full pathnames from the steps here in the **install.currprod.config.dir=** and **testportlet.install.dir=** argument values.

```
# 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=Welcome Portlet Sample

# 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 par, war, and ear
# file.  The name might be changed if you change the value of
# config.currprod.12byte above.
webapp.testportlet.archive.name=sample.welcome

# 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=SampleWelcome

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

5. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleWelcome** directory, run the configuration script with the following arguments to create the source directory structure for building the portlet:

   ```
   cfg createLocalPortletDirectories -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see “The PWENCODE Procedure” in *Encryption in SAS*.

6. Review the customconfig.log file that was created in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleWelcome** directory to determine whether any errors occurred.

Step 2: Create the Portlet Deployment Descriptor

The portlet deployment descriptor is an XML file that provides all of the information that the SAS Information Delivery Portal needs to deploy one or more portlets. The following example shows the contents of the portlet deployment descriptor file for the SampleWelcome portlet. For more information about portlet deployment descriptor files, see “Creating a Portlet Deployment Descriptor” on page 4.

```
1<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE portlets SYSTEM "http://www.sas.com/idp/portlet.dtd">
<portlets>

2<local-portlet name="@webapp.testportlet.contextroot@"
   title="Welcome Portlet Sample">

3<localized-resources locales="de,en" />

4<deployment scope="user" autoDeploy="false" userCanCreateMore="true" />

5<initializer-type>
   com.sas.portal.portlets.JspPortlet.JspPortletInitializer
</initializer-type>

<init-param>
   <param-name>display-page</param-name>
   <param-value>Welcome.jsp</param-value>
</init-param>

6<portlet-path>/sample/portlets</portlet-path>

7<portlet-actions>
   <portlet-action name="display" default="true">
```

```
      <type>sample.welcome.WelcomeAction</type>
   </portlet-action>
</portlet-actions>

   </local-portlet>
</portlets>
```

1 The DOCTYPE statement must be present in the descriptor file for the portlet to run. However, the document type definition (DTD) does not need to be accessible at the URL that the statement specifies.

   If you want to look at the portlet.dtd file, you can find it in the portal installation directory in the path **portal/WEB-INF**. For example, on a Windows system, the DTD is located at the following path: ***SAS-installation-directory/*** **SASInformationDeliveryPortal/4.4/Static/wars/sas.portal/WEB-INF/classes/portlet.dtd**.

2 The **<local-portlet>** element assigns the name to the portlet. The name cannot contain spaces. The portlet identifier, which consists of the portlet path (defined in the **<portlet-path>** element) together with the portlet name, must be unique within the SAS Information Delivery Portal. This example uses the testportlet scripting facility to build the portlet, which replaces **@webapp.testportlet.contextroot@** with the literal name **SampleWelcome**.

3 The **<localized-resources>** element lists the locales that the portlet supports. Display resource files must be provided for each of these locales.

4 The **<deployment>** element specifies that all users can create new instances of this portlet.

5 Because the SampleWelcome portlet does not need its own initializer class, the default portlet initializer (JspPortletInitializer) is specified. This class requires a parameter named **display-page**. The initializer places the value of this parameter in the PortletContext object so that it can be used by the portlet's action class. The value of the parameter is the name of the SampleWelcome portlet's JSP page, named Welcome.jsp.

   Note: The default initializer passes only the **display-page** parameter. To pass additional parameters, you need to create your own initializer class. For more information, see "Creating an Initializer Action Class" on page 9.

6 The **<portlet-path>** element specifies the directory location in which the portlet is deployed. The portlet identifier, which consists of the portlet path together with the portlet name (defined in the **<local-portlet>** element), must be unique within the SAS Information Delivery Portal. For example, Orion Star Sports & Outdoors could have two SampleWelcome portlets if different paths are specified for each (as in **/OrionStar/Sales/SampleWelcome** and **/OrionStar/Purchasing/SampleWelcome**).

7 To provide for internationalization of the text that appears inside the portlet border, the SampleWelcome portlet has its own action class, named WelcomeAction. The name of the class is specified in the **<type>** subelement of the **portlet-action** element.

Store this portlet deployment descriptor source text in a file named portlet.xml.orig in the ***portlet-source-directory*/Configurable/pars/sample.welcome** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the portlet.xml file.

Step 3: Create the Display Page

JSP pages are the presentation components of portlets. The following example shows the source code for the SampleWelcome portlet's JSP page. For more information about portlet display pages, see "Creating the Presentation JSP Page" on page 7.

```
<%-- Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513 --%>
1<%@ page language="java" contentType= "text/html; charset=UTF-8" %>
2<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
3<fmt:message key="welcome.msg1.txt"/>
<br>
<fmt:message key="welcome.msg2.txt"/>
```

1 This line contains the UTF-8 directive, which is required for internationalization. This directive encodes all user input in the 8-bit Unicode Transformation Format, which supports all of the world's languages including those that use character sets other than Latin1.

2 This line contains the taglib directive for the JSP Standard Tag Library (JSTL) formatting tags. The directive must appear before the first use of these tags.

3 These lines use JSTL formatting tags to display text. The key attribute is used to obtain the appropriate text from the resource bundle that most closely matches the user's locale preference. The SAS Information Delivery Portal makes the user's locale available to these tags.

Store this JSP code in a file named Welcome.jsp in the ***portlet-source-directory*/Static/pars/sample.welcome/SampleWelcome/content** directory.

Step 4: Create the Action Class

The SampleWelcome portlet has its own action class, WelcomeAction, which provides support for localizing messages. This class extends com.sas.portal.portlet.HTMLPortletAction, which contains code to correctly display character sets other than Latin1 when the SAS Information Delivery Portal displays the portlet in preview mode.

The following example shows the source code for the WelcomeAction class. For more information about action classes, see "Creating Action Classes" on page 8.

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.welcome;

import com.sas.portal.portlet.HTMLPortletAction;
import com.sas.portal.portlet.NavigationUtil;
import com.sas.portal.portlet.PortletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Action for the SampleWelcome Portlet. This prepares the localized resource
 * bundles for use by the JSTL tags within the portlet's JSP.
 * @version 1
 */
```

```
public final class WelcomeAction extends HTMLPortletAction {

   public WelcomeAction() {
   }

   /**
    * Configure the JSTL localization context for use in the SampleWelcome
    * portlet. Returns the value of "display-page" from the portlet's
    * XML descriptor.
    *
    * @param request The HttpServletRequest associated with the method
    *                invocation
    * @param response HttpServletResponse associated with the method
    *                 invocation
    * @param context PortletContext mapped to the request path
    *
    * @return java.lang.String - representing a valid URL.
    */
    public String service(HttpServletRequest request,
         HttpServletResponse response,
         PortletContext context)
            throws Exception {
1       super.service(request, response, context);

      NavigationUtil.prepareLocalizedResources(
         "sample.welcome.res.Resources", request, context);

      // This comes from the portlet.xml.
      String url = (String) context.getAttribute("display-page");
      return url;
   }

   private static final long serialVersionUID = 1L;
}
```

1 The NavigationUtil method uses the portlet's classloader to obtain the portlet's resource bundle. Using this bundle and the locale of the current user, it creates a new JSTL localization context. The localization context is made available to the portlet's JSP page with request scope.

Create a directory named **`sample`** under the ***`portlet-source-directory/`*** **`Static/pars/sample.welcome/SampleWelcome/source`** directory, and then create a directory named **`welcome`** under the ***`portlet-source-directory/`*** **`Static/pars/sample.welcome/SampleWelcome/source/sample`** directory. Store the class source code in a file named WelcomeAction.java in the ***`portlet-source-directory`*** **`/Static/pars/sample.welcome/SampleWelcome/source/sample/welcome`** directory.

Step 5: Create the Resource Bundles

Resource bundles provide translated text that is displayed inside the SampleWelcome portlet. The portlet's WelcomeAction.class calls the NavigationUtil.prepareLocalizedResources() method to create a JSTL localization context based on the user's locale preference. This context enables the JSTL tags in Welcome.jsp to use the appropriate resource bundle to display the text.

Note: For information about localizing the portlet's title and description, see "Step 6: Create Translated Titles and Descriptions" on page 30.

Create the following resource files for the SampleWelcome portlet:

Example Code 4.1 *Resources_de.properties (for German)*

```
# This is where you put key/value pairs for message strings that need to
# be localized.
## These are the messages for the Welcome portlet
welcome.msg1.txt=Willkommen bei SAS Information Delivery Portal.
welcome.msg2.txt=Schauen Sie sich um!
```

Example Code 4.2 *Resources_en.properties (for English)*

```
# This is where you put key/value pairs for message strings that need to
# be localized.
## These are the messages for the SampleWelcome portlet
welcome.msg1.txt=Welcome to the SAS Information Delivery Portal.
welcome.msg2.txt=Take a look around!
```

Create a directory named **res** under the ***portlet-source-directory*/Static/pars/sample.welcome/SampleWelcome/source/sample/welcome** directory. Store these resource bundles in files named Resources_de.properties and Resources_en.properties in the ***portlet-source-directory*/Static/pars/sample.welcome/SampleWelcome/source/sample/welcome/res** directory.

Step 6: Create Translated Titles and Descriptions

Translated titles and descriptions for the SampleWelcome portlet are stored in display resource files. See "Creating Display Resources Files" on page 6 for more information about display resource files.

The following display resource files are required for the SampleWelcome portlet:

Example Code 4.3 *portletDisplayResources_de.properties (for German)*

```
portlet.title=Begrüßungs-Portlet Muster
portlet.description=Begrüßungs-Portlet Muster
```

Example Code 4.4 *portletDisplayResources_en.properties (for English)*

```
portlet.title=Welcome Portlet Sample
portlet.description=Welcome Portlet Sample
```

Store these files with the specified names in the ***portlet-source-directory*/Static/pars/sample.welcome/SampleWelcome/classes** directory.

Step 7: Compile the Portlet Code

The action class that was defined in Step 4 must be compiled before the portlet can be used. SAS 9.4 uses a Versioned JAR Repository to manage the JAR files that are shipped with SAS products. The testportlet scripting facility integrates with the Versioned JAR Repository by requiring a picklist to define which JAR files are used for compiling the portlet and building the WAR file. If your portlet requires additional JAR files, they must also be added to the picklist.

Follow these steps to compile the SampleWelcome portlet:

1. Create a picklist for this sample portlet. As a starting point, copy the SAS Information Delivery Portal picklist file from the ***SAS-installation-directory*/SASInformationDeliveryPortal/4.4/Picklists/wars/**

sas.portal directory into the ***portlet-source-directory*/Picklist/pars/sample.welcome** directory.

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

2. Copy the file named servlet-api.jar that ships with the application server into the ***portlet-source-directory*/Static/lib** directory.

   Note: The ***portlet-source-directory*/Static/lib** directory is where you store any custom or third-party JAR files that are not defined in the SAS picklist but that are needed to compile the custom portlet.

3. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleWelcome** directory, run the configuration script with the following arguments to compile the Java class:

   ```
   cfg compileLocalPortlet -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

4. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleWelcome** directory to determine whether any errors occurred.

Step 8: Create the PAR File and Deploy and Test the Portlet

The last step in developing the SampleWelcome portlet is to archive its files into a PAR file and deploy the new portlet. The PAR file includes all of the portlet's supporting files, including the files created in Steps 2 through 7.

To create the PAR file and deploy the portlet, follow these steps:

1. Stop the web application server on which the SAS Information Delivery Portal is deployed so that development of the new portlet will not affect the running system.

2. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleWelcome** directory, run the configuration script with the following arguments:

   ```
   cfg buildPortletArchive -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

   The portlet archive file is created in the ***SAS-configuration-directory*/Lev1/Web/Applications/SASPortlets4.4/Deployed** directory with the name sample.welcome.par.

3. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleWelcome** directory to determine whether any errors occurred.

4. Rebuild the sas.portal4.4.ear file using the SAS Deployment Manager. This step is required because the sas.portal4.4.ear file contains files associated with each portlet.
5. Manually redeploy the sas.portal4.4.ear file into the web application server.
6. Start the web application server on which the SAS Information Delivery Portal is deployed. The SampleWelcome portlet should now be available to the portal.

It is a good practice to deploy new portlets into a staging area (that is, a test installation of the SAS Information Delivery Portal) for verification and testing before deploying them into a production environment.

SampleForm: Interactive Form Portlet

Overview: Steps for Creating the SampleForm Portlet

The SampleForm portlet is an interactive portlet that accepts free-form input from the user. When the user clicks **Submit Form**, the portlet displays the entered text back to the user.

SampleForm is a local portlet that runs inside the portlet container. It was developed using a JavaBean, which places values in the PortletContext object so that the values are available to the JSP page. The HttpServletRequestor HttpSession object could be used for this purpose. However, the PortletContext object is unique to the portlet and is not shared with other processes. Therefore, using it avoids possibly overwriting attribute values.

To create the SampleForm portlet, follow these steps:

1. Create the portlet configuration and source directories.
2. Create the portlet deployment descriptor (portlet.xml).
3. Create the display page (SampleForm.jsp).
4. Create the action class (DisplayAction.java).
5. Create a JavaBean to return user input (ExampleBean.java).
6. Create a title and description for the portlet.
7. Compile the portlet code.
8. Create the PAR file and deploy and test the portlet.

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

Step 1: Create the Portlet Configuration and Source Directories

Follow these steps to create a configuration and source directory structure for building the portlet:

1. Create a configuration directory for the portlet named **`SampleForm`** under the ***`SAS-configuration-directory`*****`/Lev1/CustomAppData`** directory. This directory is referred to as *portlet-configuration-directory* in the code and in descriptions for this portlet.

2. Copy the contents of the **testportlet** directory into this new **SampleForm** directory.
3. Create a source code directory for the portlet named **Source** under the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleForm** directory. This directory is referred to as *portlet-source-directory* in the code and in descriptions for this portlet.
4. Edit the custom.properties file in the **SampleForm** directory as follows:

   Note: Be sure to substitute the full pathnames from the steps here in the **install.currprod.config.dir=** and **testportlet.install.dir=** argument values.

```
# 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=Form Portlet Sample

# 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 par, war, and ear
# file.  The name might be changed if you change the value of
# config.currprod.12byte above.
webapp.testportlet.archive.name=sample.form

# 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=SampleForm

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

5. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleForm** directory, run the configuration script with the following arguments to create the source directory structure for building the portlet:

```
cfg createLocalPortletDirectories -Dmetadata.connection.passwd="password"
```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see “The PWENCODE Procedure” in *Encryption in SAS*.

6. Review the customconfig.log file that was created in the *SAS-configuration-directory*/**Lev1/CustomAppData/SampleForm** directory to determine whether any errors occurred.

Step 2: Create the Portlet Deployment Descriptor

The portlet deployment descriptor is an XML file that provides all of the information that the SAS Information Delivery Portal needs to deploy one or more portlets. The following example shows the contents of the portlet deployment descriptor file for the SampleForm portlet. For more information about portlet deployment descriptor files, see “Creating a Portlet Deployment Descriptor” on page 4.

```
1<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE portlets SYSTEM "http://www.sas.com/idp/portlet.dtd">
<portlets>

2<local-portlet name="@webapp.testportlet.contextroot@"
   title="Form Portlet Sample">
<localized-resources locales="de,en" />

3<deployment scope="user" autoDeploy="false" userCanCreateMore="true" />

4<init-param>
   <param-name>display-page</param-name>
   <param-value>SampleForm.jsp</param-value>
</init-param>

5<portlet-path>/sample/portlets</portlet-path>

6<portlet-actions>
   <portlet-action name="display" default="true">
      <type>sample.form.DisplayAction</type>
   </portlet-action>
</portlet-actions>

   </local-portlet>
</portlets>
```

1 The DOCTYPE statement must be present in the descriptor file in order for the portlet to run. However, the document type definition (DTD) does not need to be accessible at the URL that the statement specifies.

   If you want to look at the portlet.dtd file, you can find it in the portal installation directory in the path **portal/WEB-INF**. For example, on a Windows system, the DTD is located at the following path: ***SAS-installation-directory/*** **SASInformationDeliveryPortal/4.4/Static/wars/sas.portal/WEB-INF/classes/portlet.dtd**.

2 The **<local-portlet>** element assigns the name to the portlet. The name cannot contain spaces. The portlet identifier, which consists of the portlet path (defined in the **<portlet-path>** element) together with the portlet name, must be unique within the SAS Information Delivery Portal. This example uses the testportlet scripting facility to build the portlet, which replaces **@webapp.testportlet.contextroot@** with the literal name **SampleForm**.

3 The `<deployment>` element specifies that all users can create new instances of this portlet.

4 Because no initializer class is specified, the SampleForm portlet uses the default initializer, JspPortletInitializer. This initializer requires a page name as a parameter. The SampleForm portlet has its own JSP page named SampleForm.jsp.

5 The `<portlet-path>` element specifies the directory location in which the portlet is deployed. The portlet identifier, which consists of the portlet path together with the portlet name (defined in the `<local-portlet>` element), must be unique within the SAS Information Delivery Portal. For example, Orion Star Sports & Outdoors could have two SampleForm portlets if different paths are specified for each (as in **/OrionStar/Sales/SampleForm** and **/OrionStar/Purchasing/SampleForm**).

6 To provide its interactive functionality, the SampleForm portlet has its own action class, named DisplayAction. The name of the class is specified in the `<type>` subelement of the **portlet-action** element.

Store this portlet deployment descriptor source text in a file named portlet.xml.orig in the ***portlet-source-directory*/Configurable/pars/sample.form** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the portlet.xml file.

Step 3: Create the Display Page

JSP pages are the presentation components of portlets. The following example shows the code for the SampleForm portlet's display page. This JSP page uses SAS custom tags, which require a license for SAS AppDev Studio. For more information, see "Creating the Presentation JSP Page" on page 7. If SAS AppDev Studio software is not licensed at your site, you must substitute equivalent HTML or JSP coding instead.

```
<%-- Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513 --%>
<%@ page language="java" import="com.sas.portal.portlet.PortletContext,
      com.sas.portal.common.PortletConstants"
    contentType="text/html;charset=UTF-8" %>

<%@taglib uri="http://www.sas.com/taglib/sas" prefix="sas"%>

<%-- This portlet echoes user input back to the portlet display. --%>

<% PortletContext context = (PortletContext)
   request.getAttribute(PortletConstants.CURRENT_PORTLET_CONTEXT );
%>

<sas:Form id="form" name="form" method="POST"
   action="<%= (String) context.getAttribute(\"formExample_baseURL\")%>">

<table border="0">

<tr>
<td align="right">Enter a value:</td>

<td><sas:TextEntry id="userInput" /></td>
</tr>

<tr>
<td align="right">You typed:</td>
```

```
<td><%=(String) context.getAttribute("formExample_userInput") %></td>
</tr>

</table>

<sas:PushButton id="submit"
 text="Submit Form" type="submit" />

</sas:Form>
```

Store this JSP code in a file named SampleForm.jsp in the ***portlet-source-directory*/Static/pars/sample.form/SampleForm/content** directory.

Step 4: Create the Action Class

The SampleForm portlet has its own action class, DisplayAction. The following example shows the source code for the DisplayAction class. For more information, see “Creating Action Classes” on page 8.

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.form;

import java.util.Enumeration;
import java.util.HashMap;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.beanutils.BeanUtils;

import com.sas.portal.portlet.HTMLPortletAction;
import com.sas.portal.portlet.NavigationUtil;
import com.sas.portal.portlet.PortletContext;

/**
 * Action for the Form Example Portlet. This prepares the URL
 * that is assigned to the form's action within the portlet's JSP.
 * It also populates a bean with the parameters from the JSP form.
 *
 */
public final class DisplayAction extends HTMLPortletAction {

   public DisplayAction() {
   }

   /**
   * Prepare the URL for the form used in the portlet.
   * Returns the value of "SampleForm.jsp".
   *
   * @param request The HtppServletRequest associated with the
   * method invocation
   * @param response HttpServletResponse associated with the
   * method invocation
```

```
    * @param context PortletContext mapped to the request path
    *
    * @return java.lang.String - representing a valid
    * URL.
    */
    public String service(HttpServletRequest request,
          HttpServletResponse response, PortletContext context)
          throws Exception{

       super.service(request, response, context);

       // Prepare the base URL for setting on the form in the JSP.
       // The "display" is the value used in portlet.xml for this
       // action.
       String baseURL = NavigationUtil.buildBaseURL(context, request,
          "display");
       context.setAttribute("formExample_baseURL", baseURL);

       // Make a new ExampleBean. Alternatively, this could be made
       // once in the portlet initializer class, then you manage
       // its properties in the action.
       ExampleBean bean = new ExampleBean();

       // The BeanUtils class populates any bean with all the
       // parameters from the form.
       HashMap map = new HashMap();
       Enumeration names = request.getParameterNames();
       while (names.hasMoreElements()) {

          String name = (String) names.nextElement();

          map.put(name, request.getParameterValues(name));
       }
       BeanUtils.populate(bean, map);

       // Put the userInput into the portlet context so we can get it out
       // in the JSP.
       context.setAttribute("formExample_userInput", bean.getUserInput());

       return "SampleForm.jsp";
    }

     private static final long serialVersionUID = 1L;

}
```

Create a directory named sample under the ***portlet-source-directory*/Static/pars/sample.form/SampleForm/source** directory, and then create a directory named form under the ***portlet-source-directory*/Static/pars/sample.form/SampleForm/source/sample** directory. Store the class source code in a file named DisplayAction.java in the ***portlet-source-directory*/Static/pars/sample.form/SampleForm/source/sample/form** directory.

Step 5: Create a JavaBean to Return User Input

The SampleForm portlet uses a JavaBean to place values in the PortletContext object so that the values are available to the JSP page.

The HttpServletRequestor HttpSession object could be used for this purpose. However, the PortletContext object is unique to the portlet and is not shared with other processes. Therefore, using it avoids possibly overwriting attribute values.

The following example shows the source code for the JavaBean:

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
* All Rights Reserved.
*/
package sample.form;

public final class ExampleBean {

   /**
   * Sets the user's input to this property. If input
   * is null, it is changed to "" so that getUserInput()
   * never returns null.
   *
   * @param input
   */
   public void setUserInput(String input) {
      if (input == null) {
         input = "";
      }
      this.input = input;
   }

   /**
   * Returns the user's input or "".
   *
   * @return String
   */
   public String getUserInput() {
      return input;
   }

   private String input = "";
}
```

Store the source code for the JavaBean in a file named ExampleBean.java in the ***portlet-source-directory*/Static/pars/sample.form/SampleForm/source/sample/form** directory.

Step 6: Create a Title and Description for the Portlet

The title and description for the SampleForm portlet are stored in a display resource file. See “Creating Display Resources Files” on page 6 for more information about display resource files.

Create the following display resource file for the SampleForm portlet:

```
portlet.title=Form Portlet Sample
```

```
portlet.description=Form Portlet Sample
```

Store this text in a file named portletDisplayResources.properties in the ***portlet-source-directory*/Static/pars/sample.form/SampleForm/classes** directory.

Step 7: Compile the Portlet Code

The action class that was defined in Step 4 must be compiled before the portlet can be used. SAS 9.4 uses a Versioned JAR Repository to manage the JAR files that are shipped with SAS products. The testportlet scripting facility integrates with the Versioned JAR Repository by requiring a picklist to define which JAR files are used for compiling the portlet and building the WAR file. If your portlet requires additional JAR files, they must also be added to the picklist.

Follow these steps to compile the SampleForm portlet:

1. Create a picklist for this sample portlet. As a starting point, copy the SAS Information Delivery Portal picklist file from the ***SAS-installation-directory*/SASInformationDeliveryPortal/4.4/Picklists/wars/sas.portal** directory into the ***portlet-source-directory*/Picklist/pars/sample.form** directory.

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

2. Copy the file named servlet-api.jar that ships with the application server into the ***portlet-source-directory*/Static/lib** directory.

   Note: The ***portlet-source-directory*/Static/lib** directory is where you store any custom or third-party JAR files that are not defined in the SAS picklist but that are needed to compile the custom portlet.

3. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleForm** directory, run the configuration script with the following arguments to compile the Java class:

   ```
   cfg compileLocalPortlet -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

4. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleForm** directory to determine whether any errors occurred.

Step 8: Create the PAR File and Deploy and Test the Portlet

The last step in developing the SampleForm portlet is to archive its files into a PAR file and deploy the new portlet. The PAR file includes all of the portlet's supporting files, including the files created in Steps 2 through 7.

To create the PAR file and deploy the portlet, follow these steps:

1. Stop the web application server on which the SAS Information Delivery Portal is deployed so that development of the new portlet will not affect the running system.
2. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleForm** directory, run the configuration script with the following arguments:

   ```
   cfg buildPortletArchive -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

   The portlet archive file is created in the ***SAS-configuration-directory*/Lev1/Web/Applications/SASPortlets4.4/Deployed** directory with the name sample.form.par.
3. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleForm** directory to determine whether any errors occurred.
4. Rebuild the sas.portal4.4.ear file using the SAS Deployment Manager. This step is required because the sas.portal4.4.ear file contains files associated with each portlet.
5. Manually redeploy the sas.portal4.4.ear file into the web application server.
6. Start the web application server on which the SAS Information Delivery Portal is deployed. The SampleForm portlet should now be available to the portal.

It is a good practice to deploy new portlets into a staging area (that is, a test installation of the SAS Information Delivery Portal) for verification and testing before deploying them into a production environment.

SampleDisplayURL: Editable Portlet

Overview: Steps for Creating the SampleDisplayURL Portlet

The SampleDisplayURL portlet is an example of an editable portlet from which users can create their own portlet instances. The Sample DisplayURL portlet includes classes that enable the user to edit the new portlet instance to point to any URL that returns an HTML fragment.

To create the SampleDisplayURL portlet, follow these steps:

1. Create the portlet configuration and source directories.
2. Create the portlet deployment descriptor (portlet.xml).
3. Create the display pages for the portlet and the editor (Viewer.jsp, Editor.jsp, and Error.jsp).
4. Create the action classes (Initializer.java, BaseAction.java, DisplayAction.java, EditorAction.java, OKAction.java, CancelAction.java, and ErrorHandler.java).
5. Create resource bundles to supply user interface text.
6. Create a title and description for the portlet.

7. Compile the portlet code.
8. Create the PAR file and deploy and test the portlet.

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

Step 1: Create the Portlet Configuration and Source Directories

Follow these steps to create a source directory structure for building the portlet:

1. Create a configuration directory for the portlet named SampleDisplayURL under the ***SAS-configuration-directory*/Lev1/CustomAppData** directory. This directory is referred to as *portlet-configuration-directory* in the code and in descriptions for this portlet.
2. Copy the contents of the **testportlet** directory to the **SampleDisplayURL** directory.
3. Create a source code directory for the portlet named **Source** under the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleDisplayURL** directory. This directory is referred to as *portlet-source-directory* in the code and in descriptions for this portlet.
4. Edit the custom.properties file in the **SampleDisplayURL** directory as follows:

   Note: Be sure to substitute the full pathnames from the steps here in the **install.currprod.config.dir=** and **testportlet.install.dir=** argument values.

```
# 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=URL Display Portlet Sample

# 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 par, war, and ear
# file.  The name might be changed if you change the value of
# config.currprod.12byte above.
webapp.testportlet.archive.name=sample.displayurl

# 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=SampleDisplayURL

# Change the value of this property to be the versioned name of your web
```

```
# application.  This property is only used for remote portlets.  The name might
# be changed if you change the value of config.currprod.12byte above.
webapp.testportlet.display.name=URL Display Portlet Sample
```

5. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleDisplayURL** directory, run the following configuration script to create the source directory structure for building the portlet:

   ```
   cfg createLocalPortletDirectories -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

6. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleDisplayURL** directory to determine whether any errors occurred.

Step 2: Create the Portlet Deployment Descriptor

The portlet deployment descriptor is an XML file that provides all of the information that the SAS Information Delivery Portal needs to deploy one or more portlets. The following example shows the portlet deployment descriptor file for the SampleDisplayURL portlet. For more information about portlet deployment descriptor files, see "Creating a Portlet Deployment Descriptor" on page 4.

```
<?xml version="1.0" encoding="UTF-8" ?>

1<!DOCTYPE portlets SYSTEM "http://www.sas.com/idp/portlet.dtd">
<portlets>
2    <local-portlet name="@webapp.testportlet.contextroot@"
      title="URL Display Portlet Sample"
      editorType="portlet">

      <localized-resources locales="en" />
3       <deployment scope="user" autoDeploy="false" userCanCreateMore="true" />

      <initializer-type>
        sample.displayurl.Initializer
      </initializer-type>

      <init-param>
         <param-name>error-page</param-name>
         <param-value>Error.jsp</param-value>
      </init-param>

      <init-param>
         <param-name>display-page</param-name>
         <param-value>Viewer.jsp</param-value>
      </init-param>

      <init-param>
         <param-name>edit-page</param-name>
         <param-value>Editor.jsp</param-value>
```

```
        </init-param>

        <error-handler>
           <type>sample.displayurl.ErrorHandler</type>
        </error-handler>

        <portlet-path>/sample/portlets</portlet-path>
4         <portlet-actions>
           <portlet-action name="display" default="true">
              <type>sample.displayurl.DisplayAction</type>
           </portlet-action>
5             <portlet-action name="editor" editor="true">
              <type>sample.displayurl.EditorAction</type>
           </portlet-action>
6             <portlet-action name="ok" default="false">
              <type>sample.displayurl.OKAction</type>
           </portlet-action>

           <portlet-action name="cancel" default="false">
              <type>sample.displayurl.CancelAction</type>
           </portlet-action>

        </portlet-actions>
     </local-portlet>
  </portlets>
```

1 The DOCTYPE statement must be present in the descriptor file in order for the portlet to run. However, the DTD does not need to be accessible at the URL that the statement specifies.

2 In the **<local-portlet>** element, the value of the title attribute specifies the new portlet type that is displayed to users in the Create a New Portlet dialog box.

The attribute value **editorType="portlet"** indicates that portlets created from the template are editable. When this attribute value is specified, a portlet action with the attribute value editor="true" must also be specified. Otherwise, the portlet deployer sends a warning to the server log and does not deploy the portlet.

Note: Add the word Sample to the name and title to distinguish this portlet from the URL Display portlet that is delivered with the portal.

3 The **<deployment>** element specifies that all portal users can create new instances of the portlet.

4 This **<portlet-action>** element specifies the action class DisplayAction. The attribute value default="true" indicates that this is the default action class, which means that the class is invoked before the portlet's JSP renders.

5 This **<portlet-action>** element specifies the action class EditorAction. The attribute value editor="true" indicates that this action is invoked when a user clicks the portlet's Edit icon.

6 These **<portlet-action>** elements specify the action classes that are invoked when the user clicks **OK** and **Cancel** in the editor display page.

Store this portlet deployment descriptor source text in a file named portlet.xml.orig in the ***portlet-source-directory*****/Configurable/pars/sample.displayurl** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the portlet.xml file.

Step 3: Create the Display Pages for the Portlet and the Editor

Overview of the Pages in the SampleDisplayURL Portlet

The SampleDisplayURL portlet has the following JSP pages:

Viewer.jsp
: is the presentation component of the portlet.

Editor.jsp
: is the presentation component of the editor action.

Error.jsp
: displays messages for errors that occur during the editing process.

Create the Viewer.jsp Page

The following example shows the code for the Viewer JSP page, which is the presentation component of the SampleDisplayURL portlet:

```
<!-- Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513 -->
<%@ page language="java" contentType= "text/html; charset=UTF-8" %>
<%@ page import="com.sas.portal.portlet.PortletContext" %>
<%@ page import="com.sas.portal.common.PortletConstants" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c_rt" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt_rt" prefix="fmt_rt" %>

<%
   PortletContext context = (PortletContext) request.getAttribute(
   PortletConstants.CURRENT_PORTLET_CONTEXT );
   String url = (String) context.getAttribute("SampleDisplayURL_DisplayURL");

   if ((url == null) || (url.length() == 0)) {
%>
<p style="text-align: center;"><fmt:message key="viewer.nourl.txt"/></p>
<%
   }
   else {
      try {
%>
<c_rt:import charEncoding="UTF-8" url="<%= url %>" />
<%
      }
      catch (Exception ex) {
%>
<p style="text-align: center;">
<fmt_rt:message key="viewer.badurl.fmt">
<fmt_rt:param value="<%= url %>"/>
<fmt_rt:param value="<%= ex.getMessage() %>"/>
</fmt_rt:message>
</p>
<%
      }
   }
%>
```

Store this JSP code in a file named Viewer.jsp in the ***portlet-source-directory/*** **Static/pars/sample.displayurl/SampleDisplayURL/content** directory.

Create the Editor.jsp Page

The following example shows the code for the Editor JSP page, which is the presentation component of the editor for the SampleDisplayURL portlet:

```
<!-- Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513 -->
<%@ page language="java" contentType= "text/html; charset=UTF-8" %>
<%@ page import="com.sas.portal.portlet.PortletContext" %>
<%@ page import="com.sas.portal.common.PortletConstants" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>

<% PortletContext context = (PortletContext) request.getAttribute(
   PortletConstants.CURRENT_PORTLET_CONTEXT ); %>

<table border="0" cellpadding="2" cellspacing="0" align="center"
 width="100%">
   <tr>
   <td colspan="3"> </td>
   </tr>

   <tr>
   <td> </td>
   <td nowrap align="center"><fmt:message key="editor.task.txt"/></td>
   <td> </td>
   </tr>

   <tr>
   <td colspan="3"> </td>
   </tr>

   <tr>
   <td> </td>
   <td> <table border="0" cellpadding="0" cellspacing="0" align="center">
   <td class="celljustifyright" nowrap>
      <fmt:message key="editor.url.txt"/>
   </td>
   <td> </td>
   <td class="celljustifyleft" nowrap>
      <form method="post" action="<%= context.getAttribute(
         "SampleDisplayURL_EditOkURL") %>">
      <input type="text" name="SampleDisplayURL_DisplayURL"
      value="<%= context.getAttribute("SampleDisplayURL_DisplayURL") %>"
      size="60">
   </td>
   </tr>

   <tr>
   <td colspan="3"> </td>
   </tr>

   <tr>
   <td class="celljustifyright">
      <input class="button" type="submit"
      value="<fmt:message key="editor.action.ok.txt"/>"
```

```
         name="submit">
      </form>
      </td>
      <td> </td>
      <td class="celljustifyleft">
         <form method="post" action="<%= context.getAttribute(
         "SampleDisplayURL_EditCancelURL") %>">
         <input class="button" type="submit"
         value="<fmt:message key="editor.action.cancel.txt"/>"
         name="cancel">
         </form>
      </td>
      </tr>

   </table>
```

Store this JSP code in a file named Editor.jsp in the ***portlet-source-directory/*** **Static/pars/sample.displayurl/SampleDisplayURL/content** directory.

Create the Error.jsp Page

The following example shows the code for the Error JSP page, which displays messages for any errors that occur during the editing process:

```
<%-- Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513 --%>
<%@ page language="java" contentType= "text/html; charset=UTF-8" %>
<%@ page import="com.sas.portal.portlet.PortletContext" %>
<%@ page import="com.sas.portal.common.PortletConstants" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<% PortletContext context = (PortletContext)request.getAttribute(
   PortletConstants.CURRENT_PORTLET_CONTEXT ); %>

<fmt:message key="error.msg1.txt"/>
<br />
<%= context.getAttribute("Exception_message") %>
```

Store this JSP code in a file named Error.jsp in the ***portlet-source-directory/*** **Static/pars/sample.displayurl/SampleDisplayURL/content** directory.

Step 4: Create the Action Classes

Overview of the Action Classes in the SampleDisplayURL Portlet

The SampleDisplayURL portlet has the following action classes:

- Initializer
- BaseAction
- DisplayAction
- EditorAction
- OKAction and CancelAction
- ErrorHandler

Create the Initializer Action Class

The SampleDisplayURL portlet's Initializer action class initializes properties that are used by the other action classes and puts the properties into a PortletContext object. The following example shows the source code for the Initializer class:

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.displayurl;

import java.rmi.RemoteException;
import java.util.Properties;

import com.sas.portal.Logger;
import com.sas.portal.portlet.PortletContext;
import com.sas.portal.portlet.PortletInitializerInterface;
import com.sas.portal.portlet.configuration.Attribute;
import com.sas.portal.portlet.configuration.Configuration;
import com.sas.portal.portlet.configuration.ConfigurationFactory;

/**
 * This initializes common properties by putting them into a
 * PortletContext object.
 */
public class Initializer implements PortletInitializerInterface {

   public Initializer() {
   }

   /* Key for the URL String in the PortletContext.*/
   public static final String DISPLAY_URL_KEY =
      "SampleDisplayURL_DisplayURL";

   /* PortletContext key for the edit screen Ok button URL */
   public static final String EDIT_OK_URL_KEY =
      "SampleDisplayURL_EditOkURL";

   /* PortletContext key for the edit screen Cancel button URL */
   public static final String EDIT_CANCEL_URL_KEY =
      "SampleDisplayURL_EditCancelURL";

   /* Key for the PortletException object in the PortletContext  */
   public static final String PORTLET_EXCEPTION_KEY =
      "sasPortletException";

   /**
    * Puts initial properties into the PortletContext object. These
    * come from the portlet.xml.
    * @param initProperties a Properties object
    * @param context the PortletContext for this portlet
    */
   public void initialize(Properties initProperties,
      PortletContext context) {

      try {
         // Get the initial URL from the portlet configuration object
```

```
         Configuration config = ConfigurationFactory.getConfiguration(
            context);
         Attribute attr = config.getAttribute(
            Initializer.DISPLAY_URL_KEY);
         String url = (attr == null) ? "" : attr.getValue();

         context.setAttribute("error-page",
            initProperties.getProperty("error-page"));
         context.setAttribute("display-page",
            initProperties.getProperty("display-page"));
         context.setAttribute("edit-page",
            initProperties.getProperty("edit-page"));
         context.setAttribute(Initializer.DISPLAY_URL_KEY, url);

         if (Logger.isDebugEnabled(_loggingContext)){
            Logger.debug("Display portlet URL: " +
            url, _loggingContext);
         }
      }

      catch (RemoteException e) {
         context.setAttribute(Initializer.PORTLET_EXCEPTION_KEY, e);
      }
   }

   private static final long serialVersionUID = 1L;
   private final String _loggingContext = this.getClass().getName();

}
```

Create a directory named **sample** under the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source** directory, and then create a directory named **displayurl** under the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample** directory. Store the class source code in a file named Initializer.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Create the Base Action Class

The SampleDisplayURL portlet's BaseAction class is a superclass that is extended by the DisplayAction, EditorAction, OkAction, and CancelAction classes. The source code is shown here:

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.displayurl;

import com.sas.portal.Logger;
import com.sas.portal.container.deployment.PortletActionInfoInterface;
import com.sas.portal.portlet.NavigationUtil;
import com.sas.portal.portlet.PortletActionInterface;
import com.sas.portal.portlet.PortletContext;
import sample.displayurl.Initializer;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

public abstract class BaseAction implements PortletActionInterface {

   public BaseAction() {
      _actionInfo = null;
   }

   private final String _loggingContext = this.getClass().getName();

   /**
   * This method must be overridden in subclasses.
   * The subclasses must call super and supply a return value.
   * In this class, the method returns null.
   *
   * @see com.sas.portal.portlet.PortletActionInterface#service(
   * HttpServletRequest, HttpServletResponse, PortletContext)
   */
   public String service(HttpServletRequest request,
      HttpServletResponse response,
      PortletContext context) throws Exception {

      Logger.debug("started..", _loggingContext);
      response.setContentType("text/html;charset=UTF-8");

      // Prepare the localized resources for use by the jsp.
      try {
         NavigationUtil.prepareLocalizedResources(
            "sample.displayurl.res.Resources",
            request, context);
      }

      catch (java.io.IOException ioe) {
         Logger.error(ioe.getMessage(), _loggingContext, ioe);
      }

      return null;
   }

   /**
   * @see com.sas.portal.portlet.PortletActionInterface#setInfo(
   * PortletActionInfoInterface)
   */
   public final void setInfo(PortletActionInfoInterface info) {
      _actionInfo = info;
   }

   /**
   * @see com.sas.portal.portlet.PortletActionInterface#getInfo()
   */
   public final PortletActionInfoInterface getInfo()
   {
      return _actionInfo;
   }

   /**
```

```
    * Check the PortletContext for an exception object. If present,
    * throw it to invoke the error handler.
    * @param context the PortletContext
    */
   protected static final void errorCheck(PortletContext context)
      throws Exception {
      Exception e = (Exception)context.getAttribute(
         Initializer.PORTLET_EXCEPTION_KEY);
      if (e != null)
         throw e;
      else
         return;
   }

   private static final long serialVersionUID = 1L;
   private PortletActionInfoInterface _actionInfo;

}
```

Store the class source code in a file named BaseAction.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Create the Display Action Class

The DisplayAction class is the default action class for the SampleDisplayURL portlet. This means that the class is invoked before the portlet's JSP page renders.

The following example shows the source code for the DisplayAction class:

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.displayurl;

import com.sas.portal.portlet.PortletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Action class that presents the display page. It sets up the display
 * model then instructs the portlet container to present the display
 * page.
 */
public final class DisplayAction extends BaseAction {

   public DisplayAction() {
   }

   /**
    * Service the portlet request.
    *
    * @param request the HttpServletRequest
    * @param response the HttpServeltResponse
    * @param context the PortletContext
    * @return the URL to call
    */
```

```
   public String service(HttpServletRequest request,
      HttpServletResponse response,
      PortletContext context) throws Exception {

      super.service(request, response, context);

      // Check whether an initialization error occurred
      errorCheck(context);
      return (String)context.getAttribute("display-page");

   }

   private static final long serialVersionUID = 1L;

}
```

Store the class source code in a file named DisplayAction.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Create the Editor Action Class

The SampleDisplayURL portlet's EditorAction class is invoked when a user clicks the portlet's Edit icon.

The following example shows the source code for the EditorAction class:

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.displayurl;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sas.portal.portlet.PortletContext;
import com.sas.portal.portlet.NavigationUtil;

/**
 * Action class that presents the edit page. It sets up the edit model
 * then instructs the portlet container to present the edit page.
 */
public final class EditorAction extends BaseAction {

   public EditorAction() {
   }

   /**
    * Service the portlet request.
    *
    * @param request the HttpServletRequest
    * @param response the HttpServeltResponse
    * @param context the PortletContext
    * @return the URL to call
    */
   public String service(HttpServletRequest request,
      HttpServletResponse response,
      PortletContext context) throws Exception {
```

```
        super.service(request, response, context);

        // Create the URLs for the OK and Cancel buttons.
        String url;

        url = NavigationUtil.buildBaseURL(context, request,
           "ok");
        context.setAttribute(Initializer.EDIT_OK_URL_KEY, url);

        url = NavigationUtil.buildBaseURL(context, request,
           "cancel");
        context.setAttribute(Initializer.EDIT_CANCEL_URL_KEY, url);

        context.resetMode();

        return (String) context.getAttribute("edit-page");
     }

     private static final long serialVersionUID = 1L;

  }
```

Store the class source code in a file named EditorAction.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Create the OK Action Class

The SampleDisplayURL portlet's OkAction class is invoked when a user clicks **OK** in the editor display page.

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.displayurl;

import com.sas.portal.Logger;
import com.sas.portal.portlet.configuration.ConfigurationFactory;
import com.sas.portal.portlet.configuration.Configuration;
import com.sas.portal.portlet.PortletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Action class that processes the Ok action from the editor. It
 * persists the user-specified URL, sets up the display model, then
 * instructs the portlet container to present the display page.
 */
public final class OKAction extends BaseAction {

   public OKAction() {
   }

   /**
    * Service the portlet request.
    *
    * @param request the HttpServletRequest
```

```
     * @param response the HttpServeltResponse
     * @param context the PortletContext
     * @return the URL to call
     */
    public String service (HttpServletRequest request,
       HttpServletResponse response,
       PortletContext context) throws Exception {

       super.service(request, response, context);

       String url = request.getParameter(Initializer.DISPLAY_URL_KEY);
       context.setAttribute(Initializer.DISPLAY_URL_KEY, url);

       // Save the URL parameter
       Configuration config = ConfigurationFactory.getConfiguration(
          context);
       config.setAttribute(Initializer.DISPLAY_URL_KEY, url);
       ConfigurationFactory.storeConfiguration(context, config);

       if (Logger.isDebugEnabled(_loggingContext)){
          Logger.debug("Display portlet URL: " + url, _loggingContext);
       }

       // Back to the default, display, mode
       // context.resetMode();

       return (String)context.getAttribute("display-page");
    }

    private static final long serialVersionUID = 1L;
    private final String _loggingContext = this.getClass().getName();

 }
```

Store the class source code in a file named OKAction.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Create the Cancel Action Class

The CancelAction class is invoked when a user clicks **Cancel** in the editor display page.

```
/** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
 * All Rights Reserved.
 */
package sample.displayurl;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sas.portal.portlet.PortletContext;

/**
 * Action class that processes the Cancel action from the editor. It
 * sets up the display model then instructs the portlet container to
 * present the display page.
 */
public final class CancelAction extends BaseAction {
```

```
      public CancelAction() {
      }


      /**
       * Service the portlet request.
       *
       * @param request the HttpServletRequest
       * @param response the HttpServeltResponse
       * @param context the PortletContext
       * @return the URL to call
       */
      public String service(HttpServletRequest request,
         HttpServletResponse response,
         PortletContext context) throws Exception {

         super.service(request, response, context);

         // Back to the default, display, mode
         // context.resetMode();
         return (String)context.getAttribute("display-page");

      }
   }
```

Store the class source code in a file named CancelAction.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Create the Error Handler Action Class

The following example shows the source code for the ErrorHandler class:

```
   /** Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513.
    * All Rights Reserved.
    */
   package sample.displayurl;

   import com.sas.apps.portal.PortalException;
   import com.sas.portal.Logger;
   import com.sas.portal.portlet.ErrorHandlerInterface;
   import com.sas.portal.portlet.NavigationUtil;
   import com.sas.portal.portlet.PortletContext;
   import java.io.IOException;
   import javax.servlet.http.HttpServletRequest;
   import javax.servlet.http.HttpServletResponse;


   /**
    * Error handler for this portlet. It logs the exception and
    * returns ErrorPage.jsp for the portlet to display.
    */
   public class ErrorHandler implements ErrorHandlerInterface {

      public ErrorHandler() {
      }
```

```
private final String _loggingContext = this.getClass().getName();

/**
 * Returns the URL for the portlet controller to call.
 * This is the name of the error page JSP.
 * @param request the HttpServletRequest
 * @param response the HttpServeltResponse
 * @param context the PortletContext
 * @param exception the exception thrown by a portlet action
 * @return the URL to call
 */
public String service(HttpServletRequest request,
      HttpServletResponse response,
      PortletContext context,
      Exception thrownException) {

   // Prepare the localized resources for use by the jsp.
   try {
      NavigationUtil.prepareLocalizedResources(
         "sample.displayurl.res.Resources",
         request, context);
   }

   catch (java.io.IOException ioe) {
      Logger.error(ioe.getMessage(), _loggingContext, ioe);
   }

   // Send error to server log in default locale.
   Logger.error(thrownException.getMessage(), _loggingContext,
      thrownException);

   // Get message in user's locale.
   String msg = null;
   try {
      PortalException ourException = (PortalException)
      thrownException;
      msg = ourException.getMessage(request.getLocale());
   }

   catch (ClassCastException cce){
      msg= "";
   }

   if (msg == null) {
      // Prevent showing the word null in a JSP
      msg = "";
   }

   // Make msg available for display on error jsp.
   context.setAttribute("Exception_message", msg);

   return (String)context.getAttribute("error-page");
}

private static final long serialVersionUID = 1L;
```

```
}
```

Store the class source code in a file named ErrorHandler.java in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory.

Step 5: Create the Resource Bundle

The resource bundles provide translated text that is displayed inside the SampleDisplayURL portlet. The portlet's BaseAction, EditorAction, and ErrorHandler classes call the NavigationUtil.prepareLocalizedResources()method to create a JSTL localization context based on the user's locale preference. This context enables the JSTL tags in the portlet's JSP pages to use the appropriate resource bundle to display text. For information about localizing a portlet's title and description, see "Creating Display Resources Files" on page 6.

The following example shows the resource bundle contents for the SampleDisplayURL portlet:

Note: If you copy and paste this code, then you must remove any line breaks in the message strings for error.msg1.txt and viewer.nourl.txt.

```
# Messages for the SampleDisplayURL portlet

# NOTE: This is the same message text as found in
# com.sas.portal.res.Resources.properties. The localized versions
# from that file can be used here.
error.msg1.txt=A serious error occurred. Contact the Portal administrator.

# {0} is a URL. {1} is an exception message.
viewer.badurl.fmt=Unable to display ''{0}'' because ''{1}''.
viewer.nourl.txt=No URL has been specified. Please edit the portlet to set a URL.

editor.task.txt=Enter the URL of the HTML fragment to display.
editor.url.txt=URL:

# NOTE: These are the same messages as found in
# com.sas.portal.res.Resources.properties. The localized versions
# from that file can be used here.
editor.action.cancel.txt=Cancel
editor.action.ok.txt=OK
```

Create a directory named **res** under the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl** directory. Store the resource bundle text in a file named Resources.properties in the ***portlet-source-directory*/Static/pars/sample.displayurl/SampleDisplayURL/source/sample/displayurl/res** directory.

Step 6: Create a Title and Description for the Portlet

The SampleDisplayURL portlet uses a display resources file to provide a description that is placed in the portlet's metadata for display to users.

You can supply multiple display resources files if you want the SAS Information Delivery Portal to localize the portlet title and description at the time of deployment,

according to the default locale for the SAS Information Delivery Portal. For more information, see "Creating Display Resources Files" on page 6.

For the SampleDisplayURL portlet, create a display resources file with the following contents:

```
portlet.title=URL Display Portlet Sample
portlet.description=Sample portlet that displays the contents of a URL
```

Store this text in a file named portletDisplayResources.properties in the ***portlet-source-directory*****/Static/pars/sample.displayurl/SampleDisplayURL/classes** directory.

Step 7: Compile the Portlet Code

The action classes that were defined in Step 4 must be compiled before the portlet can be used. SAS 9.4 uses a Versioned JAR Repository to manage the JAR files that are shipped with SAS products. The testportlet scripting facility integrates with the Versioned JAR Repository by requiring a picklist to define which JAR files are used for compiling the portlet and building the WAR file. If your portlet requires additional JAR files, they must also be added to the picklist.

Follow these steps to compile the SampleDisplayURL portlet:

1. Create a picklist for this sample portlet. As a starting point, copy the SAS Information Delivery Portal picklist file from the ***SAS-installation-directory*****/SASInformationDeliveryPortal/4.4/Picklists/wars/sas.portal** directory into the ***portlet-source-directory*****/Picklist/pars/sample.displayurl** directory.

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

2. Copy the file named servlet-api.jar that ships with the application server into the ***portlet-source-directory*****/Static/lib** directory.

   Note: The ***portlet-source-directory*****/Static/lib** directory is where you store any custom or third-party JAR files that are not defined in the SAS picklist but that are needed to compile the custom portlet.

3. From the ***SAS-configuration-directory*****/Lev1/CustomAppData/SampleDisplayURL** directory, run the configuration script with the following arguments to compile the Java class:

   ```
   cfg compileLocalPortlet -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

4. Review the customconfig.log file in the ***SAS-configuration-directory/*****Lev1/CustomAppData/SampleDisplayURL** directory to determine whether any errors occurred.

Step 8: Create the PAR File and Deploy and Test the Portlet

The last step in developing the SampleDisplayURL portlet is to archive its files into a PAR file and deploy the new portlet. The PAR file includes all of the portlet's supporting files, including the files created in Steps 2 through 7.

To create the PAR file and deploy the portlet, follow these steps:

1. Stop the web application server on which the SAS Information Delivery Portal is deployed so that development of the new portlet will not affect the running system.
2. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleDisplayURL** directory, run the configuration script with the following arguments:

   ```
   cfg buildPortletArchive -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

   The portlet archive file is created in the ***SAS-configuration-directory*/Lev1/Web/Applications/SASPortlets4.4/Deployed** directory with the name sample.displayurl.par.
3. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleDisplayURL** directory to determine whether any errors occurred.
4. Rebuild the sas.portal4.4.ear file using the SAS Deployment Manager. This step is required because the sas.portal4.4.ear file contains files associated with each portlet.
5. Manually redeploy the sas.portal4.4.ear file into the web application server.
6. Start the web application server on which the SAS Information Delivery Portal is deployed. The SampleDisplayURLportlet should now be available to the portal.

It is a good practice to deploy new portlets into a staging area (that is, a test installation of the SAS Information Delivery Portal) for verification and testing before deploying them into a production environment.

SampleRemote: Remote Portlet

Overview: Steps for Creating the SampleRemote Portlet

SampleRemote is a remote portlet that calls a web application. The web application displays the string **Hello *user***, where *user* is the name of the user who is logged on to the SAS Information Delivery Portal. It also displays text that the user can edit.

To create the SampleRemote portlet, follow these steps:

1. Create portlet configuration and source directories.
2. Create the context descriptor and deployment descriptor.
3. Create the web application deployment descriptor (web.xml).

4. Create the Spring framework configuration file (infrastructure-config.xml).
5. Create the display pages for the web application (Viewer.jsp, Editor.jsp, Error.jsp, and Help.jsp).
6. Create the controller servlet class (ControllerServlet.java).
7. Create the portlet deployment descriptor (portlet.xml).
8. Create a title and description for the portlet.
9. Create the application metadata (SampleRemote.appxml).
10. Compile the remote portlet.
11. Create the EAR and PAR files and deploy and test the portlet.

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

Step 1: Create the Portlet Configuration and Source Directories

1. Create a configuration directory for the portlet named **`SampleRemote`** under the ***`SAS-configuration-directory`*`/Lev1/CustomAppData`** directory. This directory is referred to as *portlet-configuration-directory* in the code and in descriptions for this portlet.
2. Copy the contents of the **`testportlet`** directory to the **`SampleRemote`** directory.
3. Create a source code directory for the portlet named **`Source`** under the ***`SAS-configuration-directory`*`/Lev1/CustomAppData/SampleRemote`** directory. This directory is referred to as *portlet-source-directory* in the code and in descriptions for this portlet.
4. Edit the custom.properties file in the **`SampleRemote`** directory as follows:

```
# 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=Remote Portlet Sample
# 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 par, war, and ear
# file.  The name might be changed if you change the value of
# config.currprod.12byte above.
webapp.testportlet.archive.name=sample.remote

# 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=SampleRemote

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

Note: Be sure to substitute the full pathname in the **testportlet.install.dir=** argument value.

5. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory, run the following configuration script to create the source directory structure for building the portlet:

```
cfg createRemotePortletDirectories -Dmetadata.connection.passwd="password"
```

For the *password* value, you must supply the unrestricted user password for your SAS installation.

Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

6. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory to determine whether any errors occurred.

Step 2: Create the Context Descriptor and the Enterprise Application Deployment Descriptor

The following example shows the contents of the context descriptor file for the SampleRemote portlet:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="../sas_webapps/@webapp.testportlet.archive.name@.war"
         path="/@webapp.testportlet.contextroot@">
    <ResourceLink global="sas/jdbc/SharedServices" name="jdbc/SASAPP"
                  type="javax.sql.DataSource"/>
</Context>
```

Store this text in a file named context.xml.orig in the ***portlet-source-directory*/Configurable/wars/sample.remote/META-INF** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the context.xml file.

The EAR file must also contain an enterprise application deployment descriptor file that describes the web projects and other components that comprise an EAR file. The following example shows the contents of the enterprise application deployment descriptor file for the SampleRemote portlet:

```
<?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.contextroot@</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>
```

Store this text in a file named application.xml.orig in the ***portlet-source-directory*/Configurable/ears/sample.remote/META-INF** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the application.xml file.

The EAR file must also contain a manifest file that specifies information about the files packaged in the EAR. The SampleRemote portlet uses the following manifest file:

```
Manifest-Version: 1.0
```

Store this text in a file named MANIFEST.MF in the ***portlet-source-directory*/Static/ears/sample.remote/META-INF** directory.

Note: This file is placed in the **/Static** directory hierarchy rather than the **/Configurable** directory hierarchy because no substitution is required.

Step 3: Create the Web Application Deployment Descriptor

The web application deployment descriptor is an XML file that describes the web application's initialization parameters, servlets, and other components. The following example shows the contents of the deployment descriptor file for the SampleRemote portlet's web application. For more information about creating web application deployment descriptors, see the documentation for your servlet container.

```
<web-app id="SASSampleRemotePortlet" 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>

    <context-param>
        <param-name>application-name</param-name>
        <param-value>@webapp.testportlet.display.name@</param-value>
    </context-param>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-config/application-context.xml</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <servlet>
        <servlet-name>ControllerServlet</servlet-name>
        <servlet-class>sample.remote.ControllerServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
```

```
        <servlet-name>ControllerServlet</servlet-name>
        <url-pattern>/Controller</url-pattern>
    </servlet-mapping>

  <!-- filter definitions are defined in webapp-config.xml -->
  <filter>
    <filter-name>myFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
  </filter>

  <filter>
    <filter-name>RemotePortletFilter</filter-name>
    <filter-class>com.sas.portal.portlet.remote.RemotePortletFilter
    </filter-class>
    <init-param>
        <param-name>allow-webapp-mode</param-name>
        <param-value>true</param-value>
    </init-param>
  </filter>

  <!-- filter mappings -->
  <filter-mapping>
    <filter-name>myFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>RemotePortletFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!-- servlets -->
  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <resource-ref id="ResourceRef_SASAPP_DataSource">
    <res-ref-name>jdbc/SASAPP</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

</web-app>
```

Store this web application deployment descriptor source text in a file named web.xml.orig in the ***portlet-source-directory*/Configurable/wars/sample.remote/WEB-INF** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the web.xml file.

Step 4: Create the Spring Framework Configuration Files

The web application for the SampleRemote portlet uses the open-source Spring J2EE application development framework. An infrastructure configuration file defines the Spring framework components that are used in the application. The following examples

show the infrastructure configuration files for the SampleRemote portlet. For more information about the Spring framework, see **`http://www.springsource.org`**.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:sas-aop="http://www.sas.com/xml/schema/aop"
    xmlns:sas-env="http://www.sas.com/xml/schema/env"
    xsi:schemaLocation="http://www.sas.com/xml/schema/aop
        http://www.sas.com/xml/schema/aop/sas-aop.xsd
        http://www.sas.com/xml/schema/env
        http://www.sas.com/xml/schema/env/sas-env.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

  <context:annotation-config/>

  <!-- SAS Web Infrastructure Platform imports -->
  <import resource="classpath*:META-INF/xss-config.xml"/>
  <import resource="classpath*:META-INF/aop-cache-config.xml"/>
  <import resource="classpath*:META-INF/wip-services-client-config.xml"/>
  <import resource="classpath*:META-INF/shared-services-client-config.xml"/>
  <import resource="classpath*:META-INF/content-services-client-config.xml"/>

  <sas-aop:client-context-propagation/>
  <sas-env:properties id="environmentProperties"/>
  <sas-env:configurer environment-ref="environmentProperties"/>
  <sas-env:serviceURLDAO id="serviceURLDAO"/>

  <!-- infrastructure beans -->
  <bean id="localJpsProperties" class="org.springframework.beans.factory.config.PropertiesFactoryBean">
    <property name="properties" ref="environmentProperties"/>
  </bean>
  <import resource="classpath:META-INF/local-jps.xml"/>
  <bean id="trustedUserFactory" class="com.sas.svcs.authentication.helper.TrustedUserFactory">
    <constructor-arg index="0" ref="localUserService"/>
  </bean>
  <bean id="passwordRecoveryUtil" class="com.sas.svcs.authentication.helper.PasswordRecoveryUtil">
    <property name="urlGenerator" ref="svcs.urlGenerator"/>
  </bean>
  <bean id="userSessionFactory" class="com.sas.svcs.authentication.helper.UserSessionFactory">
    <property name="localSessionService" ref="localSessionService"/>
    <property name="localUserService" ref="localUserService"/>
    <property name="trustedUserFactory" ref="trustedUserFactory"/>
    <property name="passwordRecoveryUtil" ref="passwordRecoveryUtil"/>
    <property name="domain" value="@{metadata.domain}"/>
  </bean>
  <bean id="remoteToLocalContextConverter"
        class="com.sas.svcs.authentication.helper.RemoteToLocalContextConverter">
    <property name="userSessionFactory" ref="userSessionFactory"/>
```

```
  </bean>
  <!-- presentation -->
  <bean id="configServiceInitialization" autowire="no"
        class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
    <property name="targetClass">
      <value>com.sas.framework.webapp.helpers.FrameworkHelper</value>
    </property>
    <property name="targetMethod">
      <value>setConfigurationService</value>
    </property>
    <property name="arguments">
      <list>
        <ref bean="configurationService"/>
      </list>
    </property>
  </bean>
</beans>
```

Store this text in a file named infrastructure-config.xml in the ***portlet-source-directory*****/Static/wars/sample.remote/WEB-INF/spring-config** directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd">
  <!--
        This file is the root application context read by
        com.sas.svcs.webapp.servlet.springframework.DefaultContextLoaderListener
        specified in web.xml
    -->

  <!-- Servlet Filter Definitions -->
  <import resource="webapp-config.xml"/>
  <!-- SAS Web Infrastructure Platform -->
  <import resource="infrastructure-config.xml"/>

</beans>
```

Store this text in a file named application-context.xml in the ***portlet-source-directory*****/Static/wars/sample.remote/WEB-INF/spring-config** directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:sec="http://www.springframework.org/schema/security"
    xmlns:sas-web="http://www.sas.com/xml/schema/web"
    xmlns:sas-sso="http://www.sas.com/xml/schema/sso"
    xsi:schemaLocation="http://www.sas.com/xml/schema/web
```

```
        http://www.sas.com/xml/schema/web/sas-web.xsd
        http://www.sas.com/xml/schema/sso
        http://www.sas.com/xml/schema/sso/sso.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security.xsd">

  <sas-sso:http stateless="true" />
  <!-- SAS Web Infrastructure Platform filters -->
  <sas-web:locale-filter id="localeFilter"/>
  <sas-web:local-platform-services-filter id="localPlatformServicesFilter"/>
  <sas-web:character-encoding-filter id="characterEncodingFilter"/>
  <!-- filter chain -->
  <bean id="myFilterChain" class="org.springframework.security.web.FilterChainProxy">
    <sec:filter-chain-map request-matcher="ant">
      <sec:filter-chain pattern=".js" filters="none"/>
      <sec:filter-chain pattern=".css" filters="none"/>
      <sec:filter-chain pattern="/**" filters="
          sanitizingRequestFilter,
          characterEncodingFilter,
          springSecurityFilterChain,
          localeFilter,
          localPlatformServicesFilter"/>
      </sec:filter-chain-map>
  </bean>
</beans>
```

Store this text in a file named webapp-config.xml in the ***portlet-source-directory*/Static/wars/sample.remote/WEB-INF/spring-config** directory.

Step 5: Create the Display Pages for the Web Application

Overview of the Display Pages

The web application for the SampleRemote portlet has the following JSP pages:

Viewer.jsp
: is the presentation component of the portlet.

Editor.jsp
: is the presentation component of the editor action.

Error.jsp
: displays messages for errors that occur during the editing process.

Help.jsp
: displays help information for the portlet.

Create the Viewer.jsp Page

The following example shows the code for the Viewer JSP page, which is the presentation component of the SampleRemote portlet:

```
<%-- Copyright (c) 2009 by SAS Institute Inc., Cary, NC 27513 --%>
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ page import="com.sas.portal.portlet.NavigationUtil" %>
<%@ page import="com.sas.framework.webapp.servlet.RemotePortletContextData" %>
<%@ page import="com.sas.portal.common.remote.RemotePortletContextDecoder" %>
<%@ page import="com.sas.services.user.UserContextInterface" %>
<%@ page import="com.sas.web.keys.CommonKeys" %>
<%@ page import="com.sas.webapp.contextsharing.WebappContextParams" %>
<%@ page import="sample.remote.ControllerServlet" %>

<%
try {
   // Get the Remote User Context Information
   UserContextInterface uc =
      (UserContextInterface) session.getAttribute(CommonKeys.USER_CONTEXT);

   // Determine the Display Name of the user (Remote User Context).
   String user = uc.getPerson().getDisplayName();

   RemotePortletContextData pcontext =
      RemotePortletContextDecoder.decode(request);
   String pid = pcontext.getId();
   String param_name = "sample.param." + pid;
   String param_value = (request.getParameter(param_name) == null) ?  "" : "checked";
   String submiturl = NavigationUtil.buildBaseURL(pcontext, request, "formsubmit");
%>

<p>This remote portlet supports view mode, edit mode,and help mode.</p>

<p>Hello <%= user %>.</p>

<p>Data value: <%= session.getAttribute(ControllerServlet.DATA_KEY1 + pid) %></p>

<form method="post" action="<%= submiturl %>">
  <input name="<%= param_name %>" type="checkbox" <%= param_value %> />
  <%= param_name %>
  <br />
  <button type="submit">Submit</button>
</form>

<%
} catch (Throwable thr1) {
    thr1.printStackTrace();
}
%>
```

Store this JSP code in a file named Viewer.jsp in the ***portlet-source-directory/*** **Static/wars/sample.remote/jsp** directory.

Create the Editor.jsp Page

The following example shows the code for the Editor JSP page, which is the presentation component of the editor for the SampleRemote portlet:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ page import="com.sas.framework.webapp.servlet.RemotePortletContextData" %>
<%@ page import="com.sas.portal.common.remote.RemotePortletContextDecoder" %>
<%@ page import="sample.remote.ControllerServlet" %>
```

```
<%
    RemotePortletContextData pcontext =
       RemotePortletContextDecoder.decode(request);
    String pid = pcontext.getId();
%>

<table border="0" cellpadding="2" cellspacing="0" align="center" width="100%">
  <tr>
    <td colspan="3"> </td>
  </tr>
  <tr>
    <td> </td>
    <td nowrap align="center">Edit Page</td>
    <td> </td>
  </tr>
  <tr>
    <td colspan="3"> </td>
  </tr>
  <tr>
    <form method="post"
          action="<%= request.getAttribute(ControllerServlet.EDIT_OK) %>"
          accept-charset="UTF-8">
    <td> </td>
    <td> <table border="0" cellpadding="0" cellspacing="0" align="center">
      <tr>
        <td class="celljustifyright" nowrap>Input:</td>
        <td> </td>
        <td class="celljustifyleft" nowrap>
          <input type="text" name="<%= ControllerServlet.EDIT_FIELD1 %>"
              value="<%= session.getAttribute(ControllerServlet.DATA_KEY1 + pid) %>"
              size="25">
        </td>
      </tr>
      <tr>
        <td colspan="3"> </td>
      </tr>
      <tr>
        <td class="celljustifyright">
          <input class="button" type="submit" value="Ok" name="submit" >
    </form>
        </td>
        <td> </td>
        <td class="celljustifyleft">
          <form method="post"
                action="<%= request.getAttribute(ControllerServlet.EDIT_CANCEL) %>"
                accept-charset="UTF-8">
          <input class="button" type="submit" value="Cancel" name="submit" >
          </form>
        </td>
      </tr>
      </table>
    </td>
  </tr>
</table>
```

Store this JSP code in a file named Editor.jsp in the ***portlet-source-directory/*** **Static/wars/sample.remote/jsp** directory.

Create the Error.jsp Page

The following example shows the code for the Error JSP page, which displays messages for any errors that occur during the editing process:

```
<%@ page language="java" contentType= "text/html; charset=UTF-8" %>
<%@ page import="sample.remote.ControllerServlet" %>

<h1>Error</h1>
<p><%= request.getAttribute(ControllerServlet.ERROR_MESSAGE) %></p>
```

Store this JSP code in a file named Error.jsp in the ***portlet-source-directory/*** **Static/wars/sample.remote/jsp** directory.

Create the Help.jsp Page

The following example shows the code for the Help JSP page, which displays text to help the user understand how to use the portlet:

```
<%@ page language="java" contentType= "text/html; charset=UTF-8" %>


<h1>Portlet Help</h1>
<p>This is where portlet help would be displayed</p>
```

Store this JSP code in a file named Help.jsp in the ***portlet-source-directory/*** **Static/wars/sample.remote/jsp** directory.

Step 6: Create the Controller Servlet Class

The SampleRemote portlet has its own class, ControllerServlet, to support the actions of the JSP pages for the application. The following example shows the source code for the ControllerServlet class:

```
package sample.remote;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.sas.framework.webapp.servlet.RemotePortletContextData;
import com.sas.portal.common.remote.RemotePortletContextDecoder;
import com.sas.portal.portlet.NavigationUtil;
import com.sas.portal.portlet.remote.RemotePortletToolkitUtil;

public class ControllerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /* RemotePortletToolkitUtil: Request attribute key used to pass the portlet id
    of the remote portlet the requested is targeted  */
    private static final String REMOTE_PORTLET_REQUEST_TARGET = "sasportal_target_portletid";
```

```
/* Request attribute key for the error screen error message */
public static final String ERROR_MESSAGE = "sample.remote.errormessage";

/* Request attribute key for the edit screen Ok button URL */
public static final String EDIT_OK = "sample.remote.editok.url";

/* Request attribute key for the edit screen Cancel button URL */
public static final String EDIT_CANCEL = "sample.remote.editcancel.url";

/* Name of field1 input parameter */
public static final String EDIT_FIELD1 = "sample.remote.editfield1.name";

/* Attribute name for persistent storage of string entered during edit mode */
public static final String DATA_KEY1 = "sample.remote.key1.";

/* Request attribute specified in the par file to determine the portlet action  */
public static final String PORTLET_ACTION = "rp.action";

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException
{
    processRequest(request, response);
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException
{
    processRequest(request, response);
}

public boolean processRequest(HttpServletRequest request,
   HttpServletResponse response)
      throws ServletException
{
    String data = null;
    String targetPid = request.getParameter(REMOTE_PORTLET_REQUEST_TARGET);

    try {
        // If there are multiple instances of the same remote portlet in a
        // users portal session, they will use the same HttpSession. This
        // requires session attributes to be namespaced.
        HttpSession session = request.getSession();
        ServletContext servletContext = getServletConfig().getServletContext();

        // Check for logoff notification before doing anything else. Multiple
        // portlet instance can have information stored in the HttpSession,
        // but the logoff request will only be processed once. Use of an
        // HttpSessionBindingListener is recommended if resources need to be
        // cleaned up when the session is invalidated.
        if (RemotePortletToolkitUtil.isLogoffRequest(request)) {
            session.invalidate();
            return false;
        }

        // Retrieve the remote portlet context that is needed to process
        // the request.
```

```
RemotePortletContextData pcontext =
        RemotePortletContextDecoder.decode(request);
if (pcontext == null) {
    // If there is no remote portlet context, goto error page.
    request.setAttribute(ERROR_MESSAGE, "No remote portlet context");
    RequestDispatcher rd =
       servletContext.getRequestDispatcher("/jsp/Error.jsp");
    rd.forward(request, response);
    return false;
}

// The portlet id is uniquely identifies a portlet instance.
String pid = pcontext.getId();

// A remote portlet does not have access to the portlet Configuration
// object, so it is responsible for persisting data. This codes uses
// the HttpSession to cache data, but does not implement a persistent
// store.
if (session.getAttribute(DATA_KEY1 + pid) == null)
    session.setAttribute(DATA_KEY1 + pid, "initial state");

String jspurl = null;
String action = request.getParameter(PORTLET_ACTION);

if (action.equals("default")) {
    // This portlet was called with either the default action or
    // no action. Both indicate a page refresh.
    jspurl = "/jsp/Viewer.jsp";
} else if (action.equals("startedit")) {
    // Start edit mode.

    // The following call resets the portlet mode back to
    // display mode when the submit URL is issued and thus
    // needs to be called before NavigationUtil.buildBaseURL
    //pcontext.resetMode();

    //Create the URLs for the OK and Cancel buttons.
    String submiturl =
       NavigationUtil.buildBaseURL(pcontext, request, "endedit");
    request.setAttribute(EDIT_OK, submiturl);
    request.setAttribute(EDIT_CANCEL, submiturl);

    jspurl = "/jsp/Editor.jsp";
} else if (action.equalsIgnoreCase("endedit")) {
    // Exit edit mode

    String submitAction = request.getParameter("submit");
    if ((submitAction != null)
         && submitAction.equalsIgnoreCase("Ok")) {
        // This is where you would add code to persist changes and
        // update your model.
        data = request.getParameter(EDIT_FIELD1);
        session.setAttribute(DATA_KEY1 + pid, data);
    }

    // When finished processing, redisplay.
```

```
            jspurl = "/jsp/Viewer.jsp";
        } else if (action.equalsIgnoreCase("help")) {
            // Help mode.
            jspurl = "/jsp/Help.jsp";
        } else {
            // An error occurred.
            request.setAttribute(ERROR_MESSAGE,
               "Unknown portlet action: " + action);
            jspurl = "/jsp/Error.jsp";
        }

        RequestDispatcher rd = servletContext.getRequestDispatcher(jspurl);
        rd.forward(request, response);
    } catch (Exception ex) {
        ex.printStackTrace();
        throw new ServletException(ex);
    }

    return true;
  }
}
```

Create a directory named **`sample`** under the ***`portlet-source-directory`*`/Static/wars/sample.remote/source`** directory, and then create a directory named **`remote`** under the ***`portlet-source-directory`*`/Static/wars/sample.remote/`** directory. Store the class source code in a file named ControllerServlet.java in the ***`portlet-source-directory`*`/Static/wars/sample.remote/source/sample/remote`** directory.

Step 7: Create the Portlet Deployment Descriptor

The portlet deployment descriptor is an XML file that provides all of the information that the SAS Information Delivery Portal needs to deploy one or more portlets. The following example shows the contents of the portlet deployment descriptor file for the SampleRemote. For more information about portlet deployment descriptor files, see "Creating a Portlet Deployment Descriptor" on page 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlets SYSTEM "http://www.sas.com/idp/portlet.dtd">

<portlets>
    <remote-portlet name="@webapp.testportlet.contextroot@"
                    title="@webapp.testportlet.display.name@"
                    passContextId="true"
                    editorType="portlet"
                    showEditProperties="true">
        <localized-resources locales="en" />
        <deployment scope="user" autoDeploy="false" userCanCreateMore="true" />
        <portlet-path>/sas/portlets/remote</portlet-path>

     <portlet-actions>
         <!--
            The recommended way to specify the url is to use the component URL as listed below
            <url component="@webapp.testportlet.display.name@">Controller?rp.action=startedit</url>
        -->
         <portlet-action name="default" default="true">
```

```
            <url component="@webapp.testportlet.display.name@">Controller?rp.action=default</url>
        </portlet-action>

        <portlet-action name="startedit" editor="true">
            <url component="@webapp.testportlet.display.name@">Controller?rp.action=startedit</url>
        </portlet-action>

        <portlet-action name="endedit">
            <url component="@webapp.testportlet.display.name@">Controller?rp.action=endedit</url>
        </portlet-action>

        <portlet-action name="help" help="true">
            <url component="@webapp.testportlet.display.name@">Controller?rp.action=help</url>
        </portlet-action>

        <portlet-action name="logoff" remote-logoff="true">
            <url component="@webapp.testportlet.display.name@">Controller?rp.action=logoff</url>
        </portlet-action>

      </portlet-actions>
   </remote-portlet>
</portlets>
```

Store this portlet deployment descriptor source text in a file named portlet.xml.orig in the ***portlet-source-directory*/Configurable/pars/sample.remote** directory. The testportlet scripting facility performs name/value pair substitution on this file to produce the portlet.xml file.

Step 8: Create a Title and Description for the Portlet

The SampleRemote portlet uses a display resources file to provide a description that is placed in the portlet's metadata for display to users.

You can supply multiple display resources files if you want the SAS Information Delivery Portal to localize the portlet title and description at the time of deployment, according to the default locale for the SAS Information Delivery Portal. For more information, see “Creating Display Resources Files” on page 6.

For the SampleRemote portlet, create a display resources file with the following contents:

```
portlet.title=Remote Portlet Sample
portlet.description=Remote Portlet Sample
```

Store this text in a file named portletDisplayResources.properties in the ***portlet-source-directory*/Static/pars/sample.remote/SampleRemote/classes** directory.

Step 9: Create the Application Metadata for the Portlet

The following example shows the contents of an application metadata file for the SampleRemote portlet:

```
<ApplicationMetadata xmlns="http://www.sas.com/xml/schema/namespace/ApplicationMetadata-9.4">
  <Application Name="@webapp.testportlet.display.name@" Desc="@webapp.testportlet.display.name@"
    Folder="/System/Applications/@config.currprod.legalname@/@webapp.testportlet.display.name@"
   ParentComponent="SAS Application Infrastructure" Platform=""
   ProductName="@webapp.testportlet.display.name@">
```

```
    <ApplicationUri Host="@webapp.portal.host@" Port="@webapp.portal.port@"
       Protocol="@webapp.portal.protocol@" Service="/@webapp.testportlet.contextroot@"/>
    <Directives>
      <URLDirective LogonTarget="true" Name="SampleRemotePortletLogon" Uri="/index.jsp"/>
    </Directives>
    <Configuration>
      <Property Desc="" Name="Logon.Target" Value="SampleRemotePortletLogon"/>
    </Configuration>
  </Application>
</ApplicationMetadata>
```

Store this descriptor in a file named SampleRemote.appxml.orig in the ***portlet-source-directory*/Source/Config/Deployment/Metadata** directory.

From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory, run the configuration script with the following arguments:

```
cfg configure -Dmetadata.connection.passwd="password"
```

For the *password* value, you must supply the unrestricted user password for your SAS installation.

Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

Step 10: Compile the Remote Portlet

The action class that was defined in Step 6 must be compiled before the portlet can be used. SAS uses a Versioned JAR Repository to manage the JAR files that are shipped with SAS products. The testportlet scripting facility integrates with the Versioned JAR Repository by requiring a picklist to define which JAR files are used for compiling the portlet and building the WAR file. If your portlet requires additional JAR files, they must also be added to the picklist.

Follow these steps to compile the SampleRemote portlet:

1. Create a picklist for this sample portlet. As a starting point, copy the SAS Information Delivery Portal picklist file from the ***SAS-installation-directory*/SASInformationDeliveryPortal/4.4/Picklists/wars/sas.portal** directory into the ***portlet-source-directory*/Picklist/wars/sample.remote** directory.
2. Copy any custom or third-party JAR files that are not defined in the SAS picklist but that are needed to compile the custom portlet into the ***portlet-source-directory*/Static/lib** directory.
3. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory, run the configuration script with the following arguments to compile the Java class:

   ```
   cfg compileRemotePortlet -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

4. Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory to determine whether any errors occurred.

Step 11: Create the EAR and PAR Files and Deploy and Test the Portlet

The last step in developing the SampleRemote portlet is to archive its files into EAR and PAR files and deploy the web application and the new portlet. For a remote portlet, the EAR file includes all of the web application's supporting files, including the files created in Steps 2 through 6. The PAR file contains the portlet definition created in Steps 7 and 8.

To create the EAR and PAR files and deploy the web application and the portlet, follow these steps:

1. Stop the web application server on which the SAS Information Delivery Portal is deployed so that development of the new portlet will not affect the running system.
2. For deployments that use multiple web application servers only: Edit the file vfabrictcsvr.server.assignments.properties that is located in the ***SAS-configuration-directory*/Lev1/Web/Scripts/AppServer/src/Config/vfabrictcsvr/** directory. Locate the **webappsrv.javaportal.server=** property, and then add a new property named **webapp.testportlet.server=** on the following line. For the new property, specify the same value that is specified for **webappsrv.javaportal.server=** (for example, sasserver1).
3. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory, run the configuration script with the following arguments:

   ```
   cfg buildWebapps -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

   Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory to determine whether any errors occurred.
4. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory, run the configuration script with the following arguments:

   ```
   cfg deployWebapps -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

   The sample.remote.ear file containing the web application is copied to the staging area under the SAS configuration directory and deployed to your web application server.

Review the customconfig.log file in the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory to determine whether any errors occurred.

5. From the ***SAS-configuration-directory*/Lev1/CustomAppData/SampleRemote** directory, run the configuration script with the following arguments:

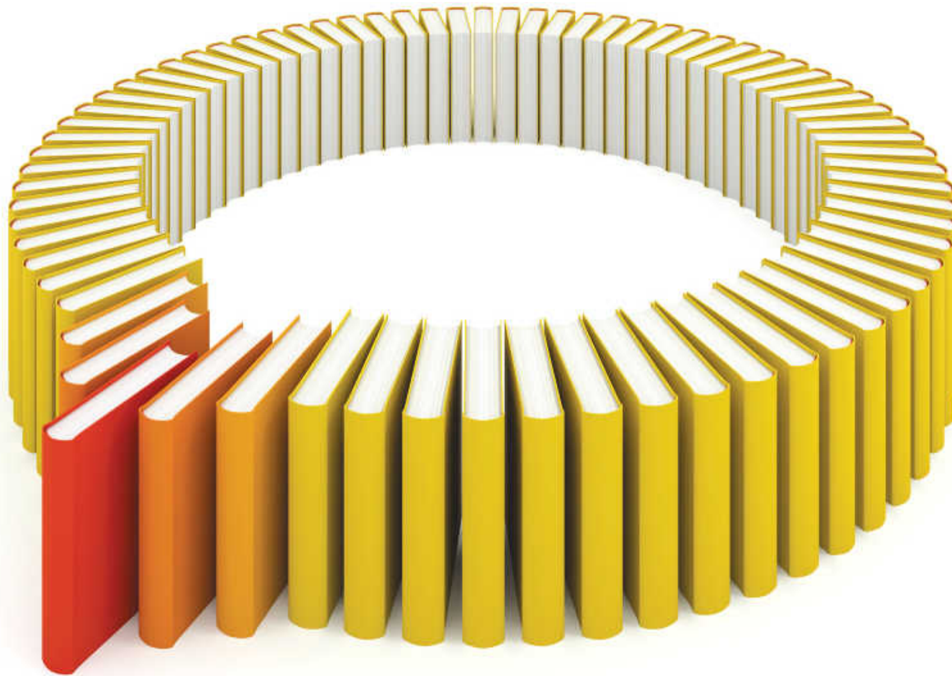
   ```
   cfg startMidtierServers -Dmetadata.connection.passwd="password"
   ```

   For the *password* value, you must supply the unrestricted user password for your SAS installation.

   Note: You can specify the password either in clear text or in encoded form. For information about generating the encoded form, see "The PWENCODE Procedure" in *Encryption in SAS*.

6. Rebuild the sas.portal4.4.ear file using the SAS Deployment Manager. This step is required because the sas.portal4.4.ear file contains files associated with each portlet.
7. Manually redeploy the sas.portal4.4.ear file into the web application server.
8. Start the web application server on which the SAS Information Delivery Portal is deployed. The SampleRemote portlet should now be available to the portal.

It is a good practice to deploy new portlets into a staging area (that is, a test installation of the SAS Information Delivery Portal) for verification and testing before deploying them into a production environment.



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613