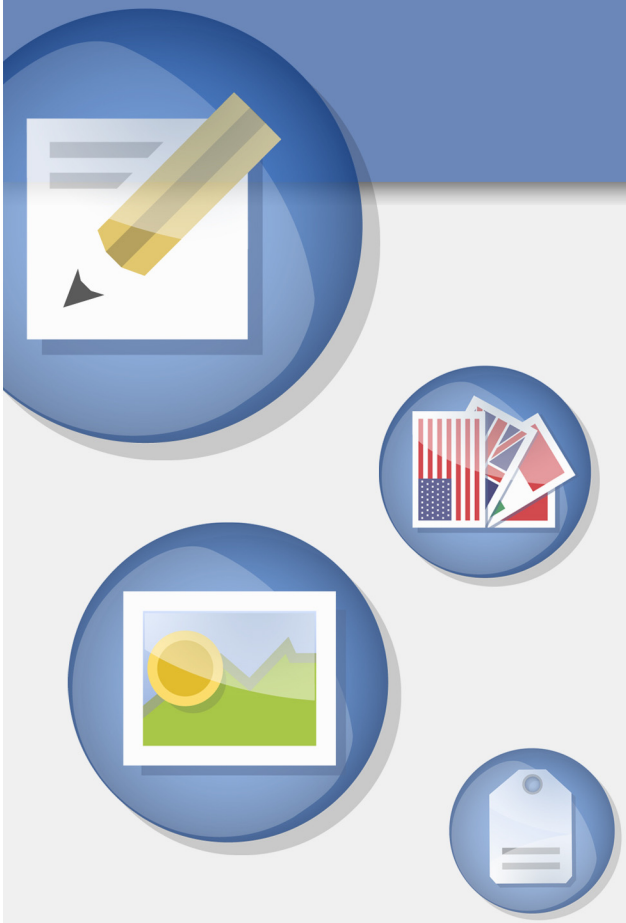
FatWire | Content Server 7

Version 7.6



Delivery Portlet Developer's Guide

Document Revision Date: Mar. 28, 2011

FatWire® SOFTWARE

FatWire Content Server Delivery Portlet Developer’s Guide
Document Revision Date: Mar. 28, 2011
Product Version: 7.6

FatWire Technical Support
www.fatwire.com/Support

FatWire Headquarters
FatWire Corporation
330 Old Country Road
Suite 303
Mineola, NY 11501
www.fatwire.com

Table of

Contents

Chapter 1

Introduction

The purpose of this guide is to help portal developers write their own portlets using the asset model that is implemented in the FatWire Content Server application. The Content Server application contains a sample site, named "Spark," which is used in this guide to illustrate code.

This guide is not intended to be a comprehensive developer's guide or a tag reference. Information regarding the Content Server system of managing content can be obtained from the administrator and user guides that are provided with the applications.

Audience

The intended audience for this guide is developers. It is assumed that the reader has a working knowledge of HTML, Java Server Pages, JSP Tag libraries, and portals. It is also assumed that the reader has a basic understanding of the portal applications and their system of managing content.

Readers who need background information are encouraged to first review the Content Server product documentation: the *Portal Applications User's Guide*, *Administrator's Guide,* and *Developer's Guide.*

Overview

Integration of Content Server with portal servers is based on the JSR-168 standard. As explained on the Sun Developer Network Site, "Java Specification Request" (JSR) 168 enables interoperability among portlets and portals. This specification defines a set of APIs for portlets and addresses standardization for preferences, user information, portlet requests and responses, deployment packaging, and security." For more information, refer to the following URL: `http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/`

In order to take advantage of this standard, all delivery portlets must use a JSR-168 compliant portlet class. This portlet class can contain business logic or simply dispatch the request to a JSP for content delivery.

Chapter 2

Portlet Classes and Sample Portlets

This chapter describes the Spark sample site. It also describes the portlet classes that are used to write portlets, and presents source code for the portlets in the Spark sample site.

This chapter contains the following sections:

Spark Sample Site

The Spark sample site provides you with the following samples:

- Four display portlets: Spark Ads, Spark Documents, Spark Jobs, and Spark News.

  The sample portlets can be found in `sparksample.jar`, located in the `/WEB-INF/lib` folder of the portal application. This jar contains both object code and source code. The example elements included with the Spark sample site can be found in the `/OpenMarket/Flame/SparkSample` folder located in the web root of the application.
- Four content definitions, also called “asset types”: Ads, Contacts, Jobs, and News Items.

  The sample asset types are composed of one or more of the sample attributes listed in Table 1.

Table 1: Sample attributes

Name	Description	Type	Notes
Title	Title	String	
Body	Body	Text	Textbox attribute editor
PostDate	Post Date	Date	
Image	Image	Blob	
Requirements	Requirements	Text	Textbox attribute editor
Contact	Contact	Asset	Asset Type is SparkContact
Phone	Phone Number	String	
Email	Email Address	String	

Portlet Classes

The JSR 168 specification defines an interface—`javax.portlet.Portlet`—that portlets can implement to interact with compliant portal servers. The same package contains `javax.portlet.GenericPortlet`, a default implementation of the interface. The JSR 168 specification also defines three portlet modes—View, Edit, and Help. Of the three modes, only View is mandatory. Helper methods for these modes are defined in the `GenericPortlet` class. In the sample portlets, the helper method do View, for the View mode, is overridden.

Sample portlets are based on the CS implementation of these portlets. A developer can write portlets by extending the class `Satellite.java`. This class provides four methods that may be overridden. Below is a description of these exposed methods.

getRenderPage

protected String `getRenderPage(RenderRequest request, RenderResponse response)`

Parameters: `request` - `RenderRequest` for the current request to render
response - The `RenderResponse` for the current request to render
Throws:
Returns: The default pagename to handle render requests for this portlet. This pagename is an element residing on the Content Server
Description: Gets the default CS pagename to handle render requests for this portlet.

getEditPage

protected String `getRenderPage(RenderRequest request, RenderResponse response)`

Parameters: `request` - `RenderRequest` for the current request to render
`response` - The `RenderResponse` for the current request to render
Throws:
Returns: The default pagename to handle edit mode render requests for this portlet. This pagename is an element residing on Content Server.
Description: Gets the default CS pagename to handle edit mode render requests for this portlet.

getHelpPage

protected String `getHelpPage(RenderRequest request, RenderResponse response)`

Parameters: `request` - `RenderRequest` for the current request to render
`response` - The `RenderResponse` for the current request to render
Throws:
Returns: The default pagename to handle help mode render requests for this portlet. This pagename is an element residing on Content Server.
Description: Gets the default CS pagename to handle help mode render requests for this portlet.

setRenderParameters

protected String `setRenderParameters(RenderRequest request, RenderResponse response)`

Parameters: `request` - `RenderRequest` for the current request to render
`response` - The `RenderResponse` for the current request to render
Throws:
Returns:
Description: Sets rendering parameters. Can be overridden. Is called by the `processAction` method.

The first three methods are executed according to the portlet mode. The `setRenderParameters` method is executed at `processAction` of the portlet. If you

need to set render parameters, you can do this by overriding the `setRenderParameters` (*`ActionRequest request, ActionResponse response`*) method.

Note

`getRenderPage(RenderRequest request, RenderResponse response)` is called by `doView` of the portlet.

`getEditPage(RenderRequest request, RenderResponse response)` is called by `doEdit` of the portlet.

`getHelpPage(RenderRequest request, RenderResponse response)` is called by `doHelp` of the portlet.

Sample Portlets

This section presents source code that illustrates the AdsPortlet, DocumentsPortlet, JobsPortlet, and NewsPortlet implementations in the Spark sample site.

Example 1. Source Code of the AdsSatellitePortlet

This portlet does not override any method. Instead, the portlet calls its default implementation and renders its default page. This default render page can be specified by the developer using the parameter `com.fatwire.cs.portals.portlet.CSPortlet.config.renderpage` in the `portlet.xml` file as shown in the example. Thus, the value for the default render page should be the Content Server element `OpenMarket/Flame/SparkSample/Ads`.

```
package com.fatwire.sparksample;
import com.openmarket.Satellite.portlet.Satellite;

/**
 * This portlet displays a random ad image from Spark
 *
 */

public class AdsSatellitePortlet extends Satellite {
}
---------------
portlet.xml Definition for this portlet

<portlet>
<portlet-name>SparkAds</portlet-name>
<portlet-class>com.fatwire.sparksample.AdsSatellitePortlet
</portlet-class>
<init-param>
<name>com.fatwire.cs.portals.portlet.CSPortlet.config.renderpage
</name>
```

```
<value>OpenMarket/Flame/SparkSample/Ads</value>
</init-param>
<supports>
<mime-type>text/html</mime-type>
<portlet-mode>view</portlet-mode>
</supports>
<portlet-info>
<title>Spark Ads</title>
</portlet-info>
</portlet>
<portlet>
```

Example 2. Source Code of the DocumentSatellitePortlet

The implementation for this portlet is the same as in Example 1. Source Code of the AdsSatellitePortlet, the only difference being the Content Server element that is called (`OpenMarket/Flame/SparkSample/Document`, in the current example).

```
package com.fatwire.sparksample;

import com.openmarket.Satellite.portlet.Satellite;
/**
 * This portlet displays all the Human Resources documents
 */
public class DocumentSatellitePortlet extends Satellite
{

}

---------------
portlet.xml Definition for this portlet

<portlet>
<portlet-name>SparkDocuments</portlet-name>
<portlet-class>com.fatwire.sparksample.DocumentsPortlet
</portlet-class>
<init-param>
<name>com.fatwire.cs.portals.portlet.CSPortlet.config.renderpage
</name>
<value>OpenMarket/Flame/SparkSample/Document</value>
</init-param>
<supports>
<mime-type>text/html</mime-type>
<portlet-mode>view</portlet-mode>
</supports>
<portlet-info>
<title>Spark Documents</title>
</portlet-info>
</portlet>
<portlet>
```

Example 3. Source Code for the SatelliteSparkJobPortlet

This portlet overrides the `getRenderPage` method. This method calls the element `OpenMarket/Flame/SparkSample/IndexPage` with the `assettype` parameter as a `SiteCatalog` argument. It returns a list of jobs and their details, accessible through supplied links.

```
package com.fatwire.sparksample;

import com.openmarket.Satellite.portlet.Satellite;


import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.PortletConfig;

/**
* This serves as a Satellite portlet for displaying a list of
    assets with a link to a detailed display.
* It requires two parameters to function properly; assettype and
    detailsjsp. The assettype parameter is
* passed through SiteCatalog entry.
* The asset type parameter indicates the name of the asset type to
   be listed.
* The detailsjsp parameter indicates the relative or absolute url
    of the Element that will display the details
* of a particular asset. The detailsjsp parameter is passed as
    init-param (portlet.xml).
* The developer can optionally specify a JSP to be used for the
    listing
*  using the init-param indexjsp.
*/
    public class SatelliteSparkJobPortlet extends Satellite {
    public static final String INDEX_JSP_PARAM     = "indexjsp";
    public static final String DETAILS_JSP_PARAM   = "detailsjsp";
    public static final String SPARK_JOB_INDEX_JSP = "OpenMarket/
         Flame/SparkSample/IndexPage";
    public static final String DISPLAY_TYPE = "displaytype";
    public static final String INDEX_DISPLAY_TYPE = "index";
    public static final String DETAILS_DISPLAY_TYPE = "details";

   /**
     * Get the default CS pagename to handle render requests for
         this portlet
     *
     * @param request  The RenderRequest for the current request
         to render
     * @param response The RenderResponse for the current request
         to render
     * @return The default pagename to handle render requests for
         this portlet.
     */
```

```
    public String getRenderPage(RenderRequest request,
                                RenderResponse response)
     {

         PortletConfig config     = getPortletConfig();
         String sDisplayType      = request.getParameter(
             DISPLAY_TYPE );
         if( sDisplayType == null )
             sDisplayType = INDEX_DISPLAY_TYPE;  //Default to list
         String sDisplayJSP = SPARK_JOB_INDEX_JSP;


          if( sDisplayType.equals( DETAILS_DISPLAY_TYPE ) )

         {
             String sDetailsJSP = config.getInitParameter(
                 DETAILS_JSP_PARAM );
             if( sDetailsJSP != null )
                 sDisplayJSP = sDetailsJSP;
         }
         else //Default to list if it's not set to details
         {
             String sIndexJSP = config.getInitParameter(
                 INDEX_JSP_PARAM );
             if( sIndexJSP != null )
                 sDisplayJSP = sIndexJSP;
         }
        // PortletRequestDispatcher rd =
           getPortletContext().getRequestDispatcher( sDisplayJSP );
        // rd.include( request, response );
        return  sDisplayJSP;

     }
}


----------
portlet.xml Definition for this portlet

<portlet>
<portlet-name>SparkJobs</portlet-name>
<portlet-class>com.fatwire.sparksample.SatelliteSparkIndexDetails
</portlet-class>
<init-param>
<name>assettype</name>
<value>Spark_Job</value>
</init-param>
<init-param>
<name>detailsjsp</name>
<value>OpenMarket/Flame/SparkSample/Jobs</value>
</init-param>
<supports>
```

```
<mime-type>text/html</mime-type>
<portlet-mode>view</portlet-mode>
</supports>
<portlet-info>
<title>Spark Jobs</title>
</portlet-info>
</portlet>
```

Example 4. Source Code of the SatelliteSparkNewsPortlet

The implementation for this portlet is the same as in Example 3. Source Code for the SatelliteSparkJobPortlet, except that it returns a list of news and details of the news.

```
package com.fatwire.sparksample;

import com.openmarket.Satellite.portlet.Satellite;

import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.PortletConfig;

/**
 * This is also same except it list the News and News Detail
 */
public class SatelliteSparkNewsPortlet extends Satellite{

    public static final String INDEX_JSP_PARAM        =
      "indexjsp";
    public static final String DETAILS_JSP_PARAM      =
      "detailsjsp";
    public static final String SPARK_NEWS_INDEX_JSP   =
      "OpenMarket/Flame/SparkSample/IndexPageNews";
    public static final String DISPLAY_TYPE           =
      "displaytype";
    public static final String INDEX_DISPLAY_TYPE     =
      "index";
    public static final String DETAILS_DISPLAY_TYPE   =
       "details";

   /**
     * Get the default CS pagename to handle render requests for
       this portlet
     *
     * @param request  The RenderRequest for the current request
          to render
     * @param response The RenderResponse for the current request
          to render
     * @return The default pagename to handle render requests for
          this portlet.
     */
public String getRenderPage(RenderRequest request,
                            RenderResponse response)
```

```
 {


   PortletConfig config = getPortletConfig();
   String sDisplayType  = request.getParameter( DISPLAY_TYPE );
   if( sDisplayType == null )
       sDisplayType = INDEX_DISPLAY_TYPE;  //Default to list

   String sDisplayJSP = SPARK_NEWS_INDEX_JSP;
   if( sDisplayType.equals( DETAILS_DISPLAY_TYPE ) )
   {
      String sDetailsJSP = config.getInitParameter(
          DETAILS_JSP_PARAM );
      if( sDetailsJSP != null )
          sDisplayJSP = sDetailsJSP;
   }
      else //Default to list if it's not set to details
      {
       String sIndexJSP = config.getInitParameter(
         INDEX_JSP_PARAM );
       if( sIndexJSP != null )
           sDisplayJSP = sIndexJSP;
      }
      // PortletRequestDispatcher rd =
          getPortletContext().getRequestDispatcher( sDisplayJSP );
      // rd.include( request, response );
      return  sDisplayJSP;
 }
}

--------------
portlet.xml portlet Definition file

<portlet>
<portlet-name>SparkNews</portlet-name>
<portlet-class>com.fatwire.sparksample.
   SatelliteSparkIndexDetails</portlet-class>
<init-param>
<name>assettype</name>
<value>Spark_News</value>
</init-param>
<init-param>
<name>detailsjsp</name>
<value>OpenMarket/Flame/SparkSample/News</value>
</init-param>
<supports>
<mime-type>text/html</mime-type>
<portlet-mode>view</portlet-mode>
</supports>
<portlet-info>
<title>Spark News</title>
</portlet-info>
</portlet>
```

Display Elements

Display elements use the tags that are provided (along with examples) in the *Content Server Tag Reference*.

News Detail Element

The News detail JSP (`OpenMarket/Flame/SparkSample/News`) is the simplest of the sample display pages. It demonstrates how to read and display an asset's attributes. The id of a News asset is passed in the request to the JSP and stored in the local `variablesArticleId`. An assetset is created for the specified News asset.

This lists the assets for which we will read attribute values. In this case, the assetset contains only one asset identified by `sArticleId`, but it could contain multiple assets as in `documents.jsp`. The `assetset:getattributevalues` retrieves the values and places them in a list of the given name. The values are placed in a list because attributes can have multiple values.

The `ics:listget` tag outputs the value of the attribute at the current position in the list unless a value is specified for the output tag attribute. In this case it is known that the attributes are all specified to have a single value so the code does not attempt to iterate through the list. The value of the `PostDate` attribute is output to a variable so it can be formatted with the `dateformat` tags.

The URL for the link to return to the news listing is generated by the `portlet:renderURL` tag provided by the portal vendor. As a developer you can specify parameters for a portlet URL with this tag (see `index.jsp`). With no parameters, the portlet will return to the default state. In this case, the default state is the index (see `SparkIndexDetails` portlet).

Ad Display Element

The Ads JSP (`OpenMarket/Flame/SparkSample/Ads`) demonstrates how to retrieve and display digital assets (or blob data) – specifically images. The Ads portlet is intended to randomly select an ad and display its image. It begins by creating a list of all Ads assets using the `asset:list` tag.

Using the Java interface for a list—`Ilist`—it randomly selects a index within the list's range. Once an asset is selected, the value of the Image attribute is read using the `assetset:getattributevalues` tag. In the case of blob-type attributes like Image, this is not the blob data itself, but an id in the blob data table `MungoBlobs`. The image is displayed using a service in Content Server called "BlobServer."

The `satellite:blob` tag outputs the correct BlobServer url to reference the image. When using this tag in Content Server, you can use the same values for the blobtable, blobcol, and blobkey parameters as shown in this example.

Jobs Detail Element

The Jobs detail JSP (`OpenMarket/Flame/SparkSample/Jobs`) is similar to the News detail JSP in that it displays attributes of a particular asset. The new concepts introduced include reading information from the base asset and working with asset-type attributes.

Since an asset in Content Server is composed of a base asset that includes fields such as id, name and description, and a number of attributes (one row per value, stored in a separate

table), different tags are used to read each type of information. To read the value of name, we use the `asset:load` tag, which creates an object to represent the Display Elements asset in memory, followed by the `asset:get` tag, which reads one field from the loaded asset.

Like blob-type attributes, the values for asset-type attribute are ids. In this case, the id refers to a `Spark_Contact` attribute. Once the id is obtained, the asset can be loaded and the attributes read in the same manner as any other asset.

Documents Element

The Documents JSP (`OpenMarket/Flame/SparkSample/Document`) lists all human resources documents. It demonstrates how to filter assetsets using searchstates, retrieve multiple attributes for multiple assets and generate a link to blob data.

A searchstate is used to specify selection criteria for assets that will be included in an assetset. In this example, we specify that the asset must have a value for the Keyword attribute equal to HR to select only human resources documents.

To retrieve multiple attribute values for multiple assets, a container must be created to specify the attributes and hold the results. The `listobject` tags are used to create this container, with a row for each attribute to be read. Instead of using the `assetset:getattributevalues` tag, we must use the `assetset:getmulitplevalues` tag. This tag uses conventions that include the asset's id to name the lists containing the attribute values.

Finally, to create links that point to the document data, we use the `satellite:blob` tag with different arguments. The first difference is the blobheader. We set this to `application/octet-stream`, which is a generic binary data content type. Some browsers will automatically detect the actual type of the document and launch the associated application, while others might prompt the user to save the file to disk.

The second change is the omission of the service attribute. In the Ads portlet, we specified "`img src`" as the service because we wanted to display an image using the `img` html tag.

In this case, we want a direct URL to the blob data, so we do not specify a service. The third change is the inclusion of the outstring parameter so that we can use the value to link the document to the title.

Generic Index Elements

The Generic Index Elements (`OpenMarket/Flame/SparkSample/IndexPage`,`OpenMarket/Flame/SparkSample/IndexPageNews`) point to the same JSP `OpenMarket/Flame/SparkSample/index.jsp` using different `assettype` parameter. The generic index JSP lists all assets of a given type, linking their names to a detailed display of the asset. This JSP is used for both the Jobs portlet and the News portlet. It demonstrates how to add parameters to a portlet link. As in the Ads portlet, we use the `asset:list` tag to get a list of all assets of a given type.

In this case, the type is specified in the portlet configuration and passed into the JSP by the portlet class (see `SparkIndexDetails` portlet above). Then the portlet iterates through the list of assets and passes the id of each asset in a portlet link using the `portlet:param` tag with the `portlet:renderUrl` tag. We also pass a parameter that indicates the

details of the asset should be displayed. This causes the portlet class to dispatch the request to the details JSP specified in the portlet configuration.

Note

If an element is called by multiple portlets and this element is using something common (such as Java script functions), use the function name `<ics:getnamespace>_function_name`. This will make the function appear different in each portlet. This treatment is equivalent to `<portlet:namespace>` in the portal environment.