Oracle® Universal Content Management

Content Portlet Suite Developer Guide

10*g* Release 3 (10.1.3.3.5)

June 2009

ORACLE®

Oracle Content Portlet Suite Developer Guide, 10*g* Release 3 (10.1.3.3.5)

Primary Author: Will Harris

Contributor: Anand Vaidyanathan, Chandrasekhar Atla

Contents

Index

Preface

The Oracle Content Portlet Suite (CPS) Developer Guide covers general information, portlet descriptions, request handling, and information on using the CPS Portlet Software Development Kit (SDK).

Audience

This guide is intended for application developers and programmers. It offers an overview of the portlets, a presentation of their framework and architecture, and information on using the portlet Software Development Kit (SDK).

Related Documents

For more information, see the following documents:

- Oracle Content Portlet Suite (CPS) Installation Guide
- Oracle Content Portlet Suite (CPS) Release Notes
- Oracle Content Integration Suite (CIS) Administration Guide
- Oracle Content Integration Suite (CIS) Developer Guide
- Oracle Content Integration Suite (CIS) Release Notes

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive

technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.

1 General Information

This chapter provides general information on the Content Portlet Suite (CPS) including an overview of the portlets, the high-level sequence of events for portlet request-handling, and instructions on migrating to version 10*g*R3. It contains the following sections:

1.1 CIS Server Dependency Removed

This version of CPS does not require a deployment of Content Integration Suite (CIS) Server to the application server as a prerequisite or dependency. Internally CPS still depends on CIS classes and these dependent classes are deployed with CPS. This architectural change has been done to eliminate remote method invocation from CPS to CIS and thus improving the performance of CPS.

1.2 Portlet Descriptions

The CPS Portlets use the Universal Content and Process Management API (UCPM API) to communicate with the Oracle Content Server. The Portlet API facade abstracts the common operations within portlet containers thus allowing our framework to work on a variety of platforms including Oracle OC4J, WebLogic, WebCenter, and AquaLogic using the same handler code. Portlet Actions are mapped to a custom MVC framework that uses the UCPM API to perform the desired task.

The Content Portlet Suite implements the following set of portlets that interact with Content Server:

- **Oracle Guest Library portlet**: Content can be presented to users based on their role in the organization.
- **Oracle Guest Search portlet**: Allows the user to perform a keyword or full-text search on the Content Server and permits read-only access to the returned content.
- **Oracle Saved Search portlet**: Allows the user to save frequently used queries.
- **Oracle Contribution portlet**: Allows the user to contribute content to the content server.
- **Oracle Workflow Queue portlet**: The workflow portlet notifies users of their workflow tasks.

- **Oracle Library portlet**: Presents content to users based on their role in the organization, and provides read/write access to the returned content.
- **Oracle Search portlet**: Allows the user to perform a selected metadata and keyword search on the content server and provides read/write access to the returned content.
- **Oracle Metadata Admin portlet**: Allows the administrator to modify the properties of custom metadata.

The portlets are consumers of standard Content Server services (IdcCommand services), such as CHECKIN_UNIVERSAL and GET_SEARCH_RESULTS. However, these services are not called directly by the dispatch handlers from the portlet controller. Rather, the UCPM API abstracts the portlets from the details of communication with the server. The UCPM API allows for rigid parameter validation, dynamic command selection, and standardized integration with a J2EE environment.

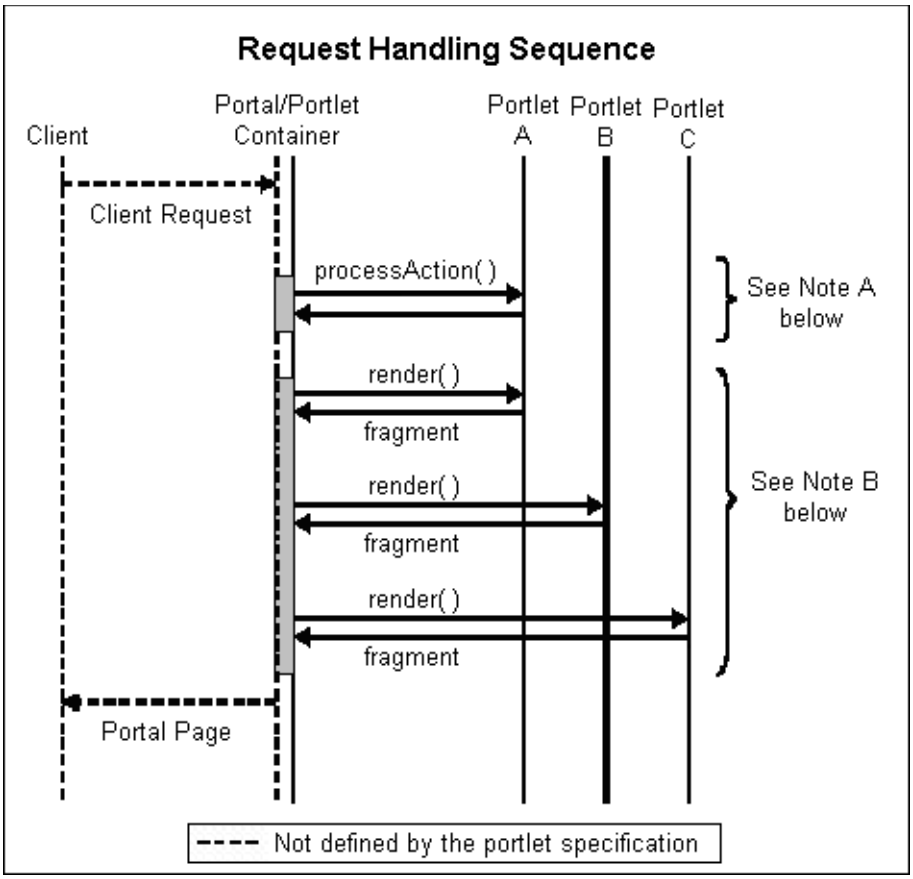
Refer to the CIS Developer Guide for more information (cis-developer-guide.pdf in the /docs directory of the CIS distribution package).

1.3 Request Handling

This is the high-level sequence of events for portlet request-handling (based on the Search portlet):

1. A user enters a query and clicks the Search button.
2. An *action* URL is built and routed to the portlet container, which, in turn, routes the command to the appropriate portlet (in this case, the search portlet).
3. A *processAction* is called on the Search portlet.
4. The Search portlet retrieves the search parameters (they are part of the URL that was built), and calls the *search* method on the CIS API.
5. The Content Integration Suite queries the Content Server via the Resource Pool (a pool of socket connections), retrieves the data, and passes the data object to the Search portlet.
6. The portlet container calls *render* on each of the portlets on the page (including the Search portlet), and each portlet uses the received data, or refreshes the data, and displays HTML fragments to the user.

Figure 1–1 Portlet Request Handling



Note A: The *action* request must finish before the render requests begin.

Note B: The *render* requests are not triggered in any specific order. They may be executed sequentially or simultaneously.

Portlet API Facade

Because portal vendors implement the standard differently, each action handler has access to a facade object that provides an interface that protects the user of the facade from code incompatibilities between various portal vendors.

2

CPS Portlet Software Development Kit

This chapter provides information on the CPS Portlet Software Development Kit (SDK) including information on building, customizing, and distributing portlets. It contains the following sections:

2.1 SDK Directory Structure

The Portlet SDK can be found in the /sdk directory of the CPS distribution package. It consists of these subdirectories:

- **ReferencePortlets**: Contains the source code for the CPS Portlets, including the Java code, JSP pages, and the Ant build.xml file used to create customized portlets. This allows those who want to customize the portlets to have access to the source code for the portlet JSP pages and the Model-View-Controller framework.
- **PortletBuilder**: Provides the structure for creating new portlets using the Model-View-Controller framework and the CIS layer. It includes an Ant build.xml file that can be used to create custom portlets for a target platform such as Oracle OC4J, WebLogic, WebCenter, and AquaLogic.
- **lib**: Contains the CPS SDK tag libraries bundled in JAR files.
- **sample**: Contains a sample development portlet as an example on how to use the PortletBuilder directory to create a custom portlet.

Apache Ant is a Java-based build tool that must be installed in order to build customized portlets. This tool is available at:

`http://ant.apache.org`

2.2 Portlet Development Tips

Whenever possible, CPS follows web standards and uses the Model-View-Controller design pattern. By following these best practices your portlets will be portable and maintainable. Portlet developers should use these coding guidelines when designing and developing portlets.

This is not intended as a primer for portlet development, as it does not address the fundamentals of portlet programming. Instead, use these guidelines as a checklist during design and code reviews to help promote consistent and quality portlet implementations.

Use these practical recommendations when developing portlets:

- **Use taglibs whenever possible**. Encapsulating Java code within JSP Tag Libraries allows you to more easily reuse common functions and makes the JSP pages easier to update.
- **Do not give portlets and servlets the same name**. Some portal servers use the portlet name to identify the portlet within a web application and may cause errors if encounters a servlet with the same name.
- **Do not use head or body tags**. The portlet JSP page contributes to the content of a larger page. Because the HTML fragment is being added to a table cell <td></td> in the portal, it should not include <html>, <head>, or <body> tags.
- **Avoid client-side JavaScript**. Using JavaScript executed on the browser makes your portlets browser-dependent and requires additional cross-browser testing.
- **Follow the Model-View-Controller design pattern**. CPS uses a Model-View-Controller design pattern based on the open source Struts and Tiles framework. Thus, the presentation of data should be separated from the logic that obtains and organizes the data.
- **Use the JavaServer Pages Standard Tag Library (JSTL)**. The JSTL defines many commonly needed tags for conditions, iterations, formatting, etc. When you see the c: prefix in the code of JSP pages, these tag libraries are being used. You can find more information about these tag libraries at:

  `http://jakarta.apache.org/taglibs/`

2.3 Using ReferencePortlets and PortletBuilder

The CPS Portlet SDK includes the *ReferencePortlets* and *PortletBuilder* directories. The ReferencePortlets directory contains source code and the PortletBuilder directory contains the portlet build files. These directories share a similar build environment and Ant scripts.

This directory structure is used by the supplied Ant file to build a portlet distribution. The PortletBuilder Ant script builds a single portlet as an example of how to package the needed portlet files for a portal container such as Oracle OC4J, WebLogic, WebCenter, and AquaLogic. Developers wanting to build many portlets should adapt the scripts accordingly.

Directory Structure	Definition
lib/compile/$portalvendor	Contains the libraries needed for building the portlets.
lib/deploy/$portalvendor	Contains the libraries needed for deploying the portlets.
resources/$portalvendor	Contains global files and portal vendor specific files needed for portlet packaging.
src	The source files for the new portlet.
build/$appserver	The directory generated during the build to hold the classes and other build-related files.
build/$appserver	The directory generated after the build is run to hold the built portlet.

Directory Structure	Definition
dist/$appserver	Contains the libraries needed for building the portlets.

2.4 Using Ant to Build Portlet Distributions

Both the PortletBuilder and ReferencePortlets root directories contain an Ant file that performs the compilation and packaging of the portlet. This root directory will be referred to as $workingdirectory. The distribution process is invoked by the following commands:

```
cd $workingdirectory
ant dist
```

For this distribution to work correctly, the following two environment variables should be set in the build.properties file in the $workingdirectory directory.

Property Name	Definition
portal.vendor	The name of the portal vendor that is the target for the current distribution.
portlet.name	The name the user wants to use for the current build. This name will be used in the generation of the descriptor files for the portlet.

The newly built portlet can be found in the $workingdirectory/dist/$portal.vendor directory. Apache Ant must be installed for this process to work properly. This tool is available at:

http://ant.apache.org.

2.5 Using the CPS Portlet Tag Libraries

The CPS Portlet Tag Libraries includes several tags that may be useful when building customized portlets. The CPS Portlet Tag Libraries are located in the /lib directory and are bundled in JAR files. This section contains the following topics:

2.5.1 URI Creation

Creates a URL through the PortletAPIFacade. The *mode* parameter is optional and, if used, the created URL will cause the portlet mode to be switched to the user-specified value. This tag is often used in conjunction with the following two nested tags:

```
<SCS:CreateURI mode=("edit" | "help" | "view"">
```

Modify the created URL by specifying an action to perform. This action name is defined in the PortletDispatch.xml file (see Portlet Dispatch Framework).

```
<SCS:URIAction name="$actionName">
```

Add the name/value pair to the generated URL.

```
<SCS:URIParameter name="$paramName" value="$paramValue">
```

Code Sample

```
<a href="
<scsportlet:createURI>
<scsportlet:URIAction value="checkOut"/>
<scsportlet:URIParameter name="documentID" value='<%=id%>' />
</scsportlet:createURI>">
Check Out File
</a>
```

2.5.2 Error Handling

Determines if an error is present. If an error is found, the body of the tag is evaluated and the error object variable is set.

```
<SCS:Error id="$errorObject">
```

Code Sample

```
<scsportlet:error id="error">
<div class="portlet-msg-error">
<%=error.getMessage ("%></div>
</scsportlet:error>
```

2.5.3 Portlet Preferences

Retrieves the specified portlet preferences and stores the result in the specified variable name.

```
<SCS:GetPreference preference="$prefName" result="$esultVar">
```

Code Sample

```
<scsportlet:getPreference preference="maxResults" result="maxResultsVar" />
```

3

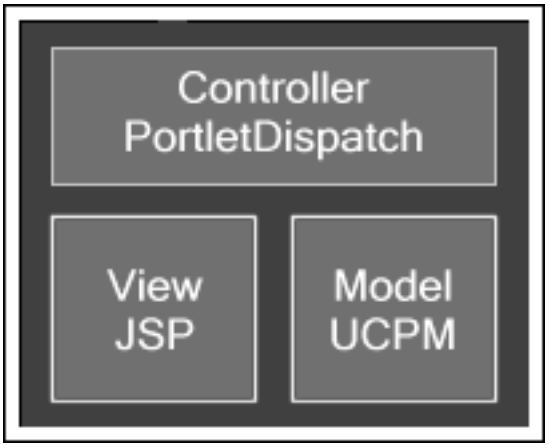
Using the CPS Portlet SDK

This chapter provides information on the using the CPS Portlet Software Development Kit (SDK), which is used to develop new portlets. It contains the following sections:

3.1 Model-View-Controller Framework

CPS uses a Model-View-Controller design pattern based on the open source Struts and Tiles framework. The presentation of data is separated from the logic that obtains and organizes the data. The *model* is the UCPM layer that encapsulates the data access layer, the *view* is the JSP pages that render the model as a user interface element, and the *controller* is the PortletDispatch handler that processes and responds to events, typically user actions, and invokes changes on the model.

Figure 3–1 MVC Framework



Refer to the CIS Developer Guide for additional information (cis-developer-guide.pdf in the /docs directory of the CIS distribution package).

3.2 Portlet Construction

Any portlet you build with the CPS Portlet SDK contains a dispatch configuration file, a set of JavaServer Pages, and a set of action handlers. When the user clicks a link in a specific portlet, the associated action handler is executed and the result of the action is placed on the request. The Tile configured to be the destination after the action is executed is then retrieved and the associated JSP pages are inserted. The JSP page then models the data that was the result of the action.

3.3 Creating a Dispatch Configuration

A dispatch configuration file defines each action handler, each Tile, and information about the portlet itself. By default, the naming convention is *stellent <portletname> dispatch.xml*.

Example: stellentactivesearchdispatch.xml

The entry point to CPS Portlets is the SCSPortlet class (there may be different implementations of this per container). This class extends the GenericPortlet class. At initialization, this class looks for a configuration file in the /WEB-INF/config directory.

This section contains the following topics:

3.3.1 Keywords

These special keywords can be used as view targets:

- **default**: Renders the default page for the portlet as defined by the default-action node; if the user is in edit mode, the default edit mode page is displayed.
- **previous**: Renders the previous page in the stack.
- **login**: Specifying an action node with this name will cause the framework to execute the action handler upon detection of a new login.
- **error**: CPS displays a default error page that assumes a throwable error has been placed on the request, you may override this error page by creating a new action definition using the *error* keyword.

3.3.2 Active Search Dispatch Configuration

Here is an example of the active search dispatch configuration coding:

```
<portletdispatch-config>
<!--
  Default action parameters, name for the default action, cacheResult is a
  boolean that specifies whether the default behavior is to cache the action
  result on the session. If the value is set to false, the action will be
  performed each time the portlet is rendered, the result data is discarded
  each time.

  The cacheResult value here can be overriden by the action definition itself,
  it the action does not specify, the default value is used.
-->
```

```
<default-action view="showHome" edit="showEdit" cacheResult="true"/>

  <!--
  Portlet-id is used to ensure that unique HTML form, javascript names are
  used, this value will be available on the request object as
  ISCSAction.PORTLET_ID
  -->

<portlet-id value="active_search_portlet"/>

  <!--
  Definitions for all the action types available to this portlet
  -->

<action-mappings>
  <forward name="showHome" authRequired="true" path="active.search.main.page"/>
  <forward name="showEdit" authRequired="true" path="active.search.edit.page"/>
    <location path="/WEB-INF/actions/active_search_actions.xml"/>
    <location path="/WEB-INF/actions/active_document_actions.xml"/>
</action-mappings>

  <!--
  Definitions for UI components available to this portlet
  -->

<tiles-definitions>
  <definition name=".mainLayout" path="/stellent/ui/layouts/mainlayout.jsp">
    <put name="header" value="/stellent/ui/fragment/header.jsp"/>
    <put name="footer" value="/stellent/ui/fragment/footer.jsp"/>
    <put name="content" value="/stellent/ui/layouts/defaultContent.jsp"/>
  </definition>
  <location path="/WEB-INF/tiles/active_search_tiles.xml"/>
  <location path="/WEB-INF/tiles/active_document_tiles.xml"/>
</tiles-definitions>
</portletdispatch-config>
```

3.3.3 Types of Child Nodes

The top-level node, <portletdispatch-config>, can have these types of child nodes:

- "Default Action Node" on page 3-3
- "Portlet ID Node" on page 3-4
- "Location Node" on page 3-4
- "Action Mappings Node" on page 3-4
- "Tiles-Definitions Node" on page 3-5

3.3.3.1 Default Action Node

The <default-action> node is used to specify the action to execute or Tile to display when the portlet or the edit mode is first visited. It will also be the action executed when the keyword *default* is the target of another action.

view/edit attribute: Specifies the view/edit default; the values are the name of a defined action and the default edit action. For example, the defined action of *showHome* and the default edit action of *showEdit*.

cacheResult attribute: Indicates whether or not the result of the action should be cached on the session or re-executed each time the render () method is called. When

users perform actions on other portlets it generates a render call which asks the portlet to redraw itself. If the cacheResult is set to *true*, this redraw will not re-execute the action but instead uses the cached result. In the case of the Active Search portlet, it is set to true by default. Individual actions can override the default for the portlet, this value is only used when not specified by the action definition.

3.3.3.2 Portlet ID Node

The <portlet-id> node is used to specify a unique name for the portlet. The string value specified here is made available on the request with the parameter name ISCSAction.PORTLET_ID. This ID is mainly used to uniquely identify HTML elements such as forms and JavaScript functions so they do not conflict with other portlets on the same page.

3.3.3.3 Location Node

The <location> node is used to specify another dispatch configuration file in which to load definitions from. It takes a path attribute and indicates where to look for the configuration file to be loaded.

This node can be a child of the Action Mappings node or the Tile Definitions node. If there is a name conflict, the Action Mappings definitions in the current configuration XML takes precedence over the loaded action definitions.

3.3.3.4 Action Mappings Node

The <action-mappings> node is the container for action definitions, which are usually defined by an Action node; two exceptions are the Forward node and the Location node.

Action Node

The <action> node specifies several attributes, which are used to perform the desired action. Here is an example of an action definition:

```
<!--
Shows the form to add new saved search.
-->

<action
  name="active.search.showAddSavedSearch"
  class="com.stellent.portlet.components.search.active.handlers.
        ShowAddSavedSearchHandler"
  bean="com.stellent.portlet.components.search.active.forms.
        AddSavedSearchForm"
  authRequired="true"
  addToStack="false">
  <forward name="success" path="active.search.savedsearch.add.page"/>
</action>
```

The attributes are:

- **name**: The name of the action. This is used when executing this action within a JSP page.
- **class**: The fully qualified class name of the class that implements the ISCSActionHandler interface. This class is where you will add your action handler code. Both the name and class attributes are required to define an action.
- **bean**: The fully qualified class name of a class that implements the ISCSActionForm interface. This is passed into your action handler when the

handleAction method is called. This bean can be populated through an HTML form post or by explicit definition through a special CPS portlet tag. This is an optional attribute, if the action does not need any input parameters this attribute can be omitted, the ISCSActionForm passed into the handleAction will be null.

- **authRequired**: Controls whether the framework will execute the action if an unauthenticated portal user tries to perform an action. It defaults to false and is an optional parameter. If it is set to *true* and an unauthenticated user attempts to execute the action, a special system JSP page will displayed asking the user to login before attempting to use the portlet.
- **addToStack**: Defines whether the portlet framework will execute this action again or cache the result when *render* is called for a redraw. It defaults to true so that portlets will show the last state when redrawing. However, some actions should not be performed more than once. Thus, you can define that the framework not save the result or remember it as the last action.

The <action> node is also a container for any number of forward actions, which specify which Tile or JSP page the portlet should display upon completion of the action. You may specify as many different *forwards* as you want in this list, as long as they have unique names. The action handler code itself specifies which *forward* to use upon completion of the action. If the handler does not explicitly state the view name to forward to upon completion of the action, it defaults to the success forward.

In addition to these framework properties, you may specify an arbitrary number of custom attributes for an action definition. These attributes will be available to the action handler via the ISCSActionHandler getAttributes() method. For example, the Contribution portlet adds a custom property called *async* that indicates whether contributions should leverage the Java Messaging Service (JMS) to perform document contributions asynchronously.

Forward Node

The <forward> node is a special type of action that does not actually execute any code but rather automatically forwards the display to the specified Tile definition or explicit JSP page location. The path attribute accepts either of these values.

3.3.3.5 Tiles-Definitions Node

The Tiles and Struts design pattern tells the framework how to render a particular view by specifying a main JSP page, various regions of content, and an optional controller class that are all used to create the final view.

Example:

```
<definition name=".mainLayout" path="/stellent/ui/layouts/mainlayout.jsp">
  <put name="header" value="/stellent/ui/fragment/header.jsp"/>
  <put name="footer" value="/stellent/ui/fragment/footer.jsp"/>
  <put name="content" value="/stellent/ui/layouts/defaultContent.jsp"/>
</definition>
```

This defines a Tile of the name *.mainLayout* and specifies the JSP page */stellent/ui/layouts/mainlayout.jsp* to be the main JSP page to use when rendering this view. Notice the Put nodes, which specify the regions of content available to the main JSP page. In this example, three regions are available: a header, footer, and content region. Each of these Put nodes specify a name for the region and the corresponding JSP page to use to render the region.

You can also specify a controller for Tile definitions and specify inheritance, as in the following definition:

```
<definition name="active.search.edit.page" extends=".mainLayout"
  controllerClass="com.stellent.portlet.components.search.active.
  controllers.EditController">

<put name="content" value="/stellent/ui/layouts/search/active/
  active_search_edit.jsp"/>
</definition>
```

This Tile extends the *.mainLayout* Tile that we defined earlier and inherits its configuration. We add a controllerClass to this Tile which is an object that implements the ISCSController interface and provides a hook to execute Java code before the Tile is rendered in situations where processing is required. Notice that this Tile definition overrides the *content* region and changes the JSP page that is used to render this region.

3.4 Getting a Reference to the Portlet API Facade

An action definition is invoked through a JSP page, like the following:

```
<form name="subAuthSearch" method="POST" onSubmit="prepareAuthScsSearch()"
    action='<scsportlet:createURI><scsportlet:URIAction
    value="active.search.doSearch"/>
    </scsportlet:createURI>'>
```

In this example, the CPS tag *createURI* invokes the *active.search.doSearch* action when the form is submitted. The action *active.search.doSearch* maps to an action definition created in the configuration file. See Action Node (page 4-5) for additional information.

The action definition specifies a class name. The class name should be an object that implements the ISCSActionHandler interface. This interface has a variety of methods that can be implemented which the framework uses to exercise the object. However, by extending the abstract base class SCSActionHandler, the developer need only implement one method:

```
/**
    * Handle an action from the portlet
    * @param portletRequest
    * @throws com.stellent.portlet.dispatcher.PortletDispatcherException
    */
public ISCSActionResult handleAction (ISCSActionForm form,
    Object portletRequest)
    throws PortletDispatcherException, CommandException, RemoteException;
```

This method will be called each time the action is invoked through the portlet framework. The method is passed in an ISCSActionForm, which is a bean that represents the parameters that are made available to this action.

This class will be of the type specified in the Action node.

The already initialized CISApplication object will be available to the action handler through the getCISApplication() method when inside the handleAction method, as will any other attributes specified on the Action node via the getAttributes() method. You may also access a unique ID for this handler via the getID() method. This can be used to store information on the session without conflicts.

The return type is an action result object. Usually, this is simply a container for the result parameters that are to be stored on the request for access within the JSP page. However, you may specify other parameters to this result, such as the view that

should be used upon return. It defaults to *success* if you use the base class SCSActionResult.

Sample Action Handler

The action definition specifies the action *active.document.checkOut*, the ISCSActionHandler class that performs the action, the ISCSActionForm, which is a bean that represents the parameters passed into the handler, and the resulting Tile that is displayed upon completion of the action.

```
<!--
Attempts to check out the specified document.
-->

<action
  name="active.document.checkOut"
  class="com.stellent.portlet.components.document.
        active.handlers.CheckOutHandler"
  bean="com.stellent.portlet.components.document.
        active.forms.CheckOutForm"
  authRequired="true" >

  <forward name="success" path="active.document.checkout.page"/>
</action>
```

The SCSActionForm code represents the parameters the checkout handler needs to complete its action. In the following example, checkout only requires the document ID of the document we are planning to check out:

```
public class CheckOutForm extends SCSActionForm {
  private String m_documentID;

  public String getDocumentID () {
    return m_documentID;
    }

  public void setDocumentID (String documentID) {
    m_documentID = documentID;
  }
}
```

The SCSActionHandler code first checks to see if the passed-in form is an instance of CheckOutForm. It errors out if this is not the case. Otherwise, it checks out the file through the UCPM API. This puts the resulting ICISObject on the request by calling result.setVariable(name, object). These objects will now be available to the JSP page rendering the view.

Refer to the CIS Developer Guide for additional information (cis-developer-guide.pdf in the /docs directory of the CIS distribution package).

```
public class CheckOutHandler extends SCSActionHandler {
/**
  * Checks out the specified content.
  *
  * @param portletRequest
  * @throws com.stellent.portlet.dispatcher.PortletDispatcherException
  */

public ISCSActionResult handleAction (ISCSActionForm form,
        Object portletRequest)
        throws PortletDispatcherException, CommandException, RemoteException {
```

```
  ISCSActionResult result = new SCSActionResult ();

  if (form instanceof CheckOutForm) {
    CheckOutForm cof = (CheckOutForm)form;
    ISCSContext ctx = SCSSession.getSCSContext (portletRequest);

    //Get checkout response
    ISCSDocumentCheckoutAPI checkoutAPI =
        getCISApplication ().getUCPMAPI ().getActiveAPI
        ().getDocumentCheckoutAPI ();
    ISCSDocumentID docID =
        getCISApplication ().getUCPMAPI ().getActiveAPI
        ()._createDocumentID (cof.getDocumentID (),
        ctx.getAdapterName ());
    ISCSDocumentActionResponse resp =
        checkoutAPI.checkoutFileByID (ctx, docID);

    //Get document info
    ISCSDocumentInformationAPI docInfoAPI =
        getCISApplication ().getUCPMAPI ().getActiveAPI
        ).getDocumentInformationAPI ();
    ISCSDocumentInformationResponse docInfoResp =
        docInfoAPI.getDocumentInformationByID
        (SCSSession.getSCSContext (portletRequest), docID);

        result.setVariable ("checkoutResponse", resp);
        result.setVariable ("infoResponse", docInfoResp.getDocNode ());
    } else {
        throw new PortletDispatcherException ("Unexpected form type,
        expected 'CheckOutForm', got " + form);
    }

  return result;
}
```

3.5 Creating a Tile

A Tile consists of a definition, an optional controller class, and a collection of JSP classes that will make up a portlet view. The definition section contains XML code that identifies the main layout JSP page.

The following example specifies three different regions that reference three JSP pages:

```
<%@ include file="/stellent/ui/fragment/jspimport.inc" %>
<scsportlet:insert name="header"/>
<scsportlet:insert name="content"/>
<scsportlet:insert name="footer"/>
```

The *include* at the top includes commonly defined imports and taglib definitions. This is an example of the file for the portlets:

```
<%@ page import="com.stellent.portlet.api.IPortletAPIFacade" %>
<%@ page import="com.stellent.portlet.api.PortletAPI" %>
<%@ include file="/stellent/ui/fragment/page.inc" %>
<%@ taglib uri="/WEB-INF/tlds/i18n.tld" prefix="i18n" %>
<%@ taglib uri="/WEB-INF/tlds/scsportlet.tld" prefix="scsportlet" %>
<%@ taglib uri="/WEB-INF/tlds/c.tld" prefix="c" %>
<%@ taglib uri="/WEB-INF/tlds/scs-databinder.tld" prefix="db" %>

<%
```

```
  //the api facade class
  IPortletAPIFacade apiFacade = PortletAPI.getInstance ().getPortletAPIFacade ();
%>
```

In the JSP page, three lines use the *insert* tag to tell the view to put the header first, the content next, and the footer last. For each region, the insert tag tells the framework to look up the definition and to include the JSP page specified by the insert tag.

3.6 Creating a Controller

A controller is a hook that allows the Tile author to execute Java code before the Tile itself is rendered. To create a controller, you need to implement the ISCSController class, which requires several methods that the portlet framework uses to control its lifetime.

In most cases, the abstract base class SCSController performs all the operations you need, with this one exception:

```
/**
  * Method is called before a Tile is rendered.
  *
  * @param portletRequest The portlet request that generated the Tile render.
  * @param portletResponse The portlet response associated with Tile render.
  * @throws ServletException If a portlet container error occurs.
  * @throws IOException If a portlet container error occurs.
  * @throws CommandException If a CIS framework error occurs.
  * @throws RemoteException If a CIS communication error occurs.
  */

public void perform (Object portletRequest,
        Object portletResponse)
        throws ServletException, IOException, CommandException, RemoteException;
```

This method will get called immediately before the Tile is rendered and any objects you place on the request will be available to the resulting JSP page.

Index

G

H

I

J

L

M

O

P

R

S

T

U

V

W

X