

Stellent[®] Content Portlet Suite

Portlet Developer's Guide



STELLENT™



Copyright

© 1996–2005 Stellent, Inc. All rights reserved. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without written permission from the owner, Stellent, Inc., 7500 Flying Cloud Drive, Suite 500, Eden Prairie, Minnesota 55344 USA. The copyrighted software that accompanies this manual is licensed to the Licensee for use only in strict accordance with the Software License Agreement, which the Licensee should read carefully before commencing use of this software.

Stellent is a registered trademark, and the Stellent logo, Stellent Content Server, Stellent Content Management, Stellent Site Studio, Stellent Content Integration Suite, Stellent Content Portlet Suite, Stellent Desktop Integration Suite, Stellent Dynamic Converter, Stellent Content Publisher, Stellent Inbound Refinery, and Stellent Image Server are trademarks of Stellent, Inc. in the USA and other countries.

Adobe, Acrobat, the Acrobat Logo, Acrobat Capture, Distiller, Frame, the Frame logo, and FrameMaker are registered trademarks of Adobe Systems Incorporated.

Apache is a registered trademark of the Apache Software Foundation. Stellent Content Integration Suite includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

BEA is a registered trademark, and WebLogic is a trademark of BEA Systems, Inc.

HP-UX is a registered trademark of Hewlett-Packard Company.

IBM, Informix, and WebSphere are registered trademarks of IBM Corporation.

Kofax is a registered trademark, and Ascent and Ascent Capture are trademarks of Kofax Image Products.

Linux is a registered trademark of Linus Torvalds.

Microsoft is a registered trademark, and Windows, Word, and Access are trademarks of Microsoft Corporation.

Oracle is a registered trademark of Oracle Corporation.

Plumtree is a registered trademark of Plumtree Software, Inc.

Portions Copyright © 1991-1997 LEAD Technologies, Inc. All rights reserved.

Red Hat is a registered trademark of Red Hat, Inc.

Sun is a registered trademark, and Solaris, Sun ONE, iPlanet, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc.

Sybase is a trademark of Sybase, Inc.

UNIX is a registered trademark of The Open Group.

Verity is a registered trademark of Verity, Incorporated.

All other trade names are the property of their respective owners.

Contents

1 / Introduction	5
Overview	5
CIS Administration Application Help system.....	7
2 / Stellent Portlets	9
Stellent portlets.....	9
Integration architecture	10
Portlet request handling sequence.....	11
3 / Stellent Portlet Software Development Kit.....	12
SDK directory structure	12
Portlet development tips	13
Using the ReferencePortlets and PortletBuilder directories.....	13
Using Ant to build portlet distributions.....	14
Using the Stellent Portlet Tag Libraries	14
4 / Using the Stellent Portlet SDK	16
Portlet framework	16
Portlet construction	17
Creating a dispatch configuration file	17
Keywords	17
Active search dispatch configuration	18
Default Action node	19
Portlet ID node	19
Location node.....	19
Action Mappings node.....	19
Tiles-Definitions node	21
Getting a reference to the portlet API facade.....	22
Creating an action handler	22
Creating a Tile	24
Creating a controller	25
Index	26

CHAPTER 1

Introduction

One key to implementing a corporate portal is ensuring a secure, personalized way to aggregate data for consumption and processing. To provide the best possible combination of a portal solution and a content management solution, Stellent has created Stellent Content Portlet Suite.

With Stellent Content Portlet Suite, you can manage content creation and the distribution process through a set of easy-to-use portlets. By providing access to Stellent Content Server and Stellent Image Server, Stellent Content Portlet Suite enables users to update, search, and view content in an easy, efficient way.

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

In this section:

- Overview
- CIS Administration Application Help system

Notes

- The information in this guide is subject to change as product technology evolves and as hardware and operating systems are created and modified.
- Due to the technical nature of browsers, databases, web servers, and operating systems, Stellent, Inc. cannot warrant compatibility with all versions and features of third-party products.

Overview

The portlets in Stellent Content Portlets Suite can be enabled for different users, based on their roles and permissions in the organization, and can be easily customized. Users can browse or search content based on their permissions, contribute new content (with the appropriate level of access), and view the progress of their content through workflow.

Keeping portals up-to-date

The ease of use of Stellent Content Portlet Suite addresses a key issue: how to keep content up-to-date. By driving content updates and additions through the portal interface, the process of updating the portal becomes part of using the portal, as opposed to a separate task performed outside of the portal.

Ease of use for contribution

With Stellent, content contribution is simple; users can contribute content by checking in a document. Stellent takes care of normalizing the data with its ability to convert files through templates to your specified markup for viewing in the portal.

Ease of use for workflow

After checking in or updating content, users can track content status through the portal. Users are notified right in the portal of their workflow status—whether to review, edit, or approve content—with links to the content itself. They can click a link to view a content item, approve or edit it, and then send the item on its way for further workflow or publishing. When content is approved, it is published and made available for viewing. Approved content can be published and expired at predetermined times (as in the case of a promotional offer or classified ad).

Ease of use for browsing

Content can be presented to users based on their role in the organization. The support team may log in to see a portlet that shows new updates and fixes, with another portlet that shows current cases. These are automatically updated through the Stellent Content Management System and require no manual intervention. They appear to the users as titles with a link to the content; they may also have a short summary of the content.

Ease of use for searching

Content can be easily searched based on metadata categories and full-text content through one of the Stellent portlets. For instance, users may search for a “support note” content type containing the word “policy,” which removes the need to search through all other occurrences of the word “policy” in other content types. Stellent provides separate portlets for basic and authenticated search. Stellent even provides a portlet that allows users to save searches they use regularly.

Easy to administer

Stellent Content Server and Stellent Content Portlet Suite are easy to administer. They can be set up and left to run indefinitely. Once you have decided your metadata model and workflow paths, Stellent Content Server and Stellent Content Portlet Suite can be installed and linked to the portal, and users can begin contributing and viewing content.

Metadata Admin portlet

Stellent Content Portlet Suite includes the Metadata Admin portlet that allows you to modify properties of the content server metadata fields, which affect the behavior of the Contribution portlet. With the Metadata Admin portlet, an administrator can specify which metadata fields users see, as well as the default value for each field. Administrators can even hide metadata fields and specify what values should be set for those items.

High performance

Stellent Content Portlet Suite is built on top of Stellent Content Integration Suite. This powerful integration layer offers speedy asynchronous access to content.



Easy to customize

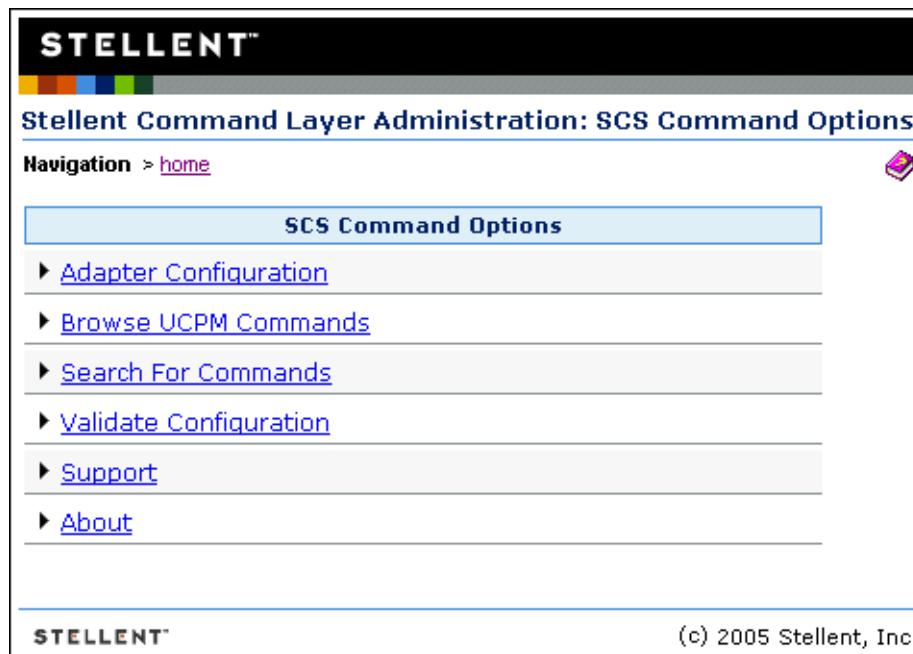
Stellent portlets are designed to be customizable. Customizing the portlets will help you make the most of your portal investment. For example, to create a new portlet to show content of a certain metadata value, you can copy the existing Browse portlet, tweak one simple parameter, and publish the portlet.

By integrating Stellent Content Portal Suite and your portal server with Stellent Content Server, you provide an easy way to keep your portal up-to-date. You also improve efficiency, lower costs, and increase your return on your portal investment.

CIS Administration Application Help system

With Stellent Content Portlet Suite built on top of Stellent Content Integration Suite (CIS), you may need assistance with the CIS Administration Application, which is the administration interface for CIS. Both online Help and the complete CIS Installation Guide can be accessed from any of the CIS Administration Application screens.

- Each screen of the Administration Application has a **Help**  icon, which you can click to view a topic that describes the functionality of that screen.
- Once you have opened any Help topic, click the **Show Navigation**  button in the navigation bar to view the complete CIS Installation Guide online.










The online Help can be viewed in a web browser such as Internet Explorer or Netscape Navigator, with conventional web browser controls (Back, Forward, Refresh, and so on) used to navigate the Help system.

The online Help contains the following navigation options:

- **Contents:** The Contents tab contains an expandable list of Help topics. Click a book icon to expand or collapse that section of the Help system and then select the desired Help topic to view a topic.
- **Index:** The Index tab provides immediate access to individual Help topics by keyword.

- **Search:** The Search tab provides a full-text search of the Help system. Type the word or phrase you are looking for and then click **Go** (or press Enter on your keyboard).
- **Favorites:** The Favorites tab provides a means to save topics that you have found useful and anticipate viewing again. (When viewing a topic, click **Favorites** and then click **Add** to add the topic to your Favorites list.) The Favorites tab does not display if your browser does not support the Java applet used by the Help system.

The online Help also includes a toolbar with the following options:

Toolbar	Definition
	Highlights the title of the topic currently being viewed in the Contents tab. Note: If you open a Help topic by clicking the Help button in a dialog box, the Show Navigation button will display, instead.
	Displays the topic that precedes the currently displayed topic. (The order of topics can be seen on the Contents tab.)
	Displays the topic that follows the currently displayed topic. (The order of topics can be seen on the Contents tab.)
	Opens a new, blank email message addressed to Stellent Technical Support, with the name of the Help topic in the Subject line. Please use this feature to provide feedback to Stellent on the topic itself or how the topic is documented.
	Prints the currently displayed topic by using your web browser print feature.
	Opens a PDF version of the CIS Installation Guide. You can read the PDF version online or print it out, if you prefer.
	Connects to the Stellent web site (www.stellent.com), where you can learn more about Stellent products, upcoming events, and technical support.

Notes

- The Help systems use a secure Java applet to display the Contents, Index, Search, and Favorites tabs. If the Java applet is not installed or enabled in the web browser used to view Help, a JavaScript rendition of Help will display, instead. If both Java and JavaScript are disabled in the web browser, the browser will not be able to display Help.
- The Java applet is not supported by Internet Explorer on UNIX, Netscape 4.x on Macintosh, Netscape 6.x on any platform, or by any machine running a newer version of the Java Runtime Environment (JRE) 1.3.1_02. For these browsers and operating systems, the JavaScript rendition of Help will display (assuming it is not disabled in the browser).

CHAPTER 2

Stellent Portlets

A portlet is a Java technology–based web component, managed by a portlet container, that processes requests and generates dynamic content. Portlets are used by portals as pluggable, user-interface components that provide a presentation layer to information systems.

The Java Community Process has produced a specific standard for portlet development: JSR 168 - Portlet Specification. The specification can be found at: <http://www.jcp.org/en/jsr/detail?id=168>

Each portlet in Stellent Content Portlets Suite (CPS) conforms to this standard and should run on any portlet container that supports this specification.

In this section:

- Stellent portlets
- Integration architecture
- Portlet request handling sequence

Important: Stellent Content Integration Suite (CIS) version 7.6 must be installed. CIS 7.6 uses the Universal Content and Process Management API (UCPM API), which replaces the LW API used in CIS 7.2. (The two APIs are not compatible.) For more information, see “Migrating from version 7.2” in the UCPM API Developer’s Guide ([ucpm-dev-guide.pdf](#)).

Note: See the CPS JavaDoc for information on the Class/Interface, Field, and Method descriptions. The JavaDoc is in the `/docs/cps-javadoc.zip` file.

Stellent portlets

Stellent Content Portlet Suite implements the following set of portlets that interact with Stellent Content Server, Stellent Image Server, or both:

Stellent Content Server

- **Library:** Presents content to users based on their role in the organization.
- **Search:** Allows the user to perform a keyword or full-text search on the content server and permits read-only access to the returned content.
- **Saved Search:** Allows the user to save frequently used queries.
- **Contribution:** Allows the user to contribute content to the content server.
- **Workflow Queue:** Notifies users of their workflow tasks.
- **Authenticated Library:** Presents content to users based on their role in the organization, and provides read/write access to the returned content.
- **Authenticated Search:** Allows the user to perform a selected metadata and keyword search on the content server and provides read/write access to the returned content.
- **Metadata Admin:** Allows the user to modify the properties of custom metadata.

Stellent Image Server

- **Image Server Search:** Allows the user to perform a search on the image server and permits read-only access to the returned content.

Stellent Content Server and Stellent Image Server

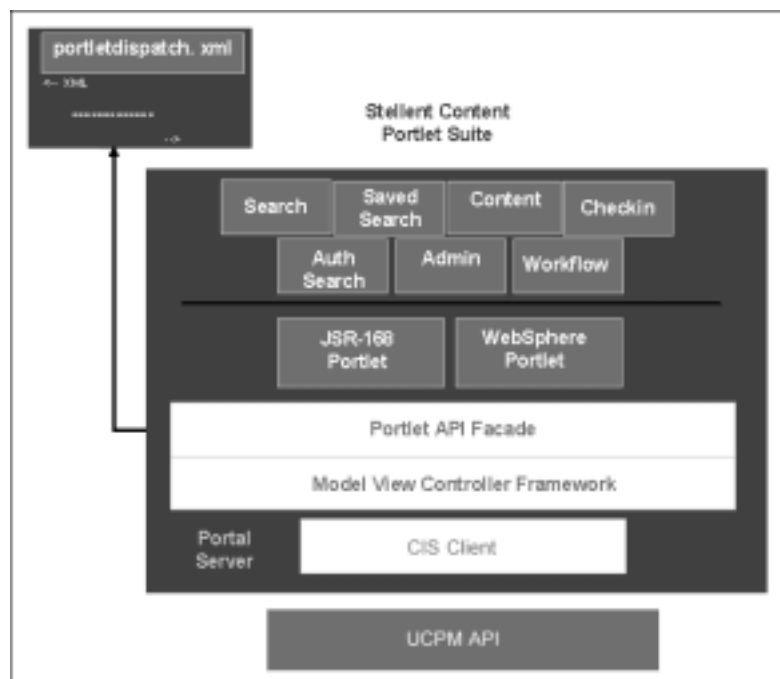
- **Federated Search:** Facilitates the integration of an image server with one or more content servers and allows queries of all the repositories.

Each of these portlets requires that Stellent Content Integration Suite (CIS) be installed and available. CIS provides a Java API into the content and image servers, and is capable of running in either a J2EE application server environment (e.g., WebSphere or WebLogic) or a servlet container environment (e.g., Tomcat).

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 talking to the server. The UCPM API allows for rigid parameter validation, dynamic command selection, and standardized integration with a J2EE environment.

Integration architecture

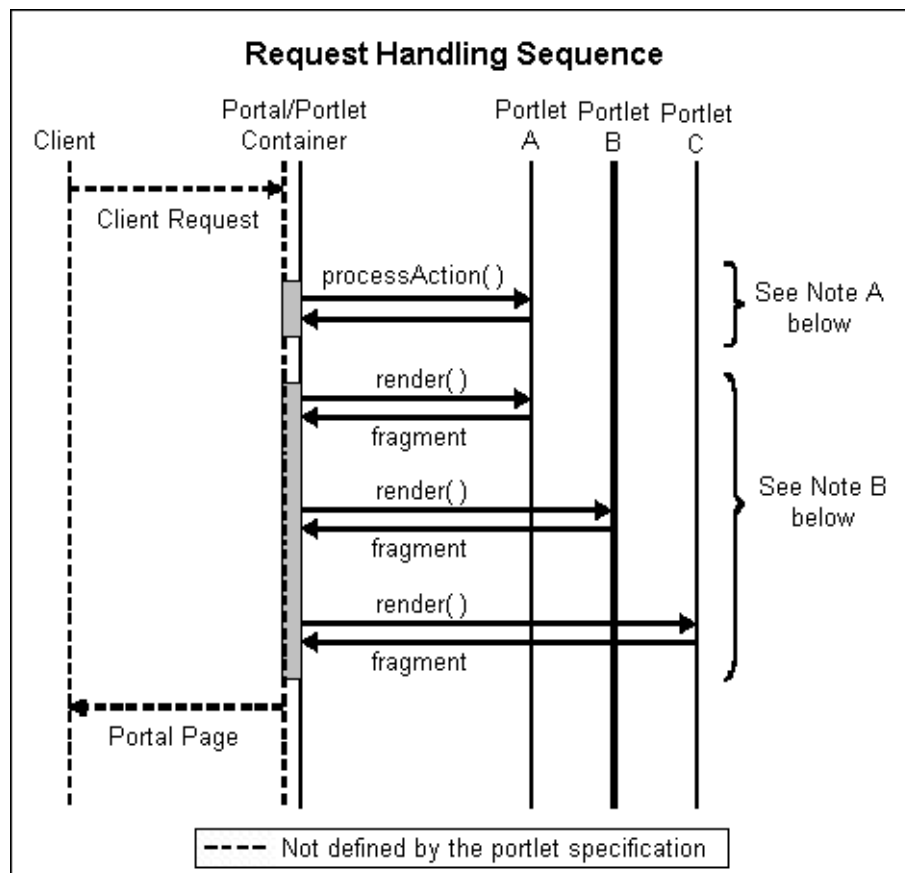
The Stellent portlets were developed using the JSR 168 standard (or leverages the WebSphere environment), and use the UCPM API to communicate back to the content server or image server. However, not every portal vendor supports the JSR 168 standard completely or in exactly the same way. Thus, CPS uses an API facade to the portlet container which abstracts common operations, so our implementation will work on a variety of platforms using the same handler code. Portlet actions are mapped to a custom Model-View-Controller framework that uses the UCPM API to perform the desired task.



Portlet request handling sequence

As an example of request handling, here is the high-level sequence of events using 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 layer.
5. The CIS layer queries the content server, 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.



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

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

CHAPTER 3

Stellent Portlet Software Development Kit

The Stellent Portlet Software Development Kit (SDK) provides everything you need to build, customize, and distribute portlets.

In this section:

- SDK directory structure
- Portlet development tips
- Using the ReferencePortlets and PortletBuilder directories
- Using Ant to build portlet distributions
- Using the Stellent Portlet Tag Libraries

See “Using the Stellent Portlet SDK” on page 16 for more information on how to create portlets.

Important: Stellent Content Integration Suite (CIS) version 7.6 must be installed. CIS 7.6 uses the Universal Content and Process Management API (UCPM API), which replaces the LW API used in CIS 7.2. (The two APIs are not compatible.) For more information, see “Migrating from version 7.2” in the UCPM API Developer’s Guide (ucpm-dev-guide.pdf).

Note: See the CPS JavaDoc for information on the Class/Interface, Field, and Method descriptions. The JavaDoc is located in the /docs/cps-javadoc.zip file.

SDK directory structure

The Stellent Portlet SDK can be found in the sdk/ directory of the unbundled CPS distribution file. It consists of these sub-directories:

ReferencePortlets: Contains the source code for the Stellent portlets, including the Java code, JSP pages, and Ant build.xml file that is used to create customized portlets. Thus, those who wish to customize the portlets 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 that use 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 (WebSphere, WebLogic, Plumtree, or Sun ONE).

lib: Contains the SDK tag libraries bundled in JAR files.

sample: Contains a sample development portlet that shows how to use the PortletBuilder directory to create a custom Stellent portlet.

Note: Apache Ant is a Java-based build tool that must be installed in order to build customize portlets. This tool is available at: <http://ant.apache.org>

Portlet development tips

Whenever possible, CPS follows web standards such as JSR 168 and the Model-View-Controller design pattern. Thus, your portlet implementations should also follow these standards.

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 this 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-dependant 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 JavaServer Pages Standard Tag Library (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/>

Using the ReferencePortlets and PortletBuilder directories

The Stellent 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 few portal vendors (WebSphere,

WebLogic, Plumtree, and Sun ONE). Users wishing to build many portlets should adapt the scripts accordingly.

Directory Structure	Description
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 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.
dist/\$appserver	The directory generated after the build is run to hold the built portlet.

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	Description
portal.vendor	The name of the portal vendor that is the target for the current distribution.
portlet.name	The name the user wishes 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.

Note: Apache Ant must be installed for this process to work properly. This tool is available at: <http://ant.apache.org>

Using the Stellent Portlet Tag Libraries

The Stellent Portlet Tag Libraries includes several tags that may be useful when building customized portlets. The Stellent Portlet Tag Libraries are located in the lib/ directory and are bundled in JAR files.

URI creation

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

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

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

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

Add the name/value pair to the generated URL.

Code sample:

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

Error handling

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

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.

Code sample:

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

Portlet preferences

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

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

Code sample:

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

CHAPTER 4

Using the Stellent Portlet SDK

Stellent Content Portlet Suite (CPS) includes the Stellent Portlet Software Development Kit (SDK), which is used to develop new portlets. See “Stellent Portlet Software Development Kit” on page 12 for an overview of the Stellent Portlet SDK, an explanation of the various subdirectories, and a description of how Ant and the tag libraries are used to build portlet distributions.

In this section:

- Portlet framework
- Portlet construction
- Creating a dispatch configuration file
- Getting a reference to the portlet API facade
- Creating an action handler
- Creating a Tile
- Creating a controller

Important: Stellent Content Integration Suite (CIS) version 7.6 must be installed. CIS 7.6 uses the Universal Content and Process Management API (UCPM API), which replaces the LW API used in CIS 7.2. (The two APIs are not compatible.) For more information, see “Migrating from version 7.2” in the UCPM API Developer’s Guide (ucpm-dev-guide.pdf).

Note: See the CPS JavaDoc for information on the Class/Interface, Field, and Method descriptions. The JavaDoc is located in the /docs/cps-javadoc.zip file.

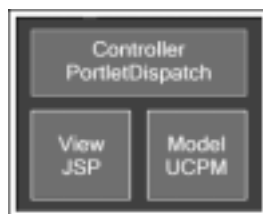
Portlet framework

The Stellent Portlet SDK is composed of two pieces:

- Model-View-Controller framework
- API facade to the portlet container

Model-View-Controller framework

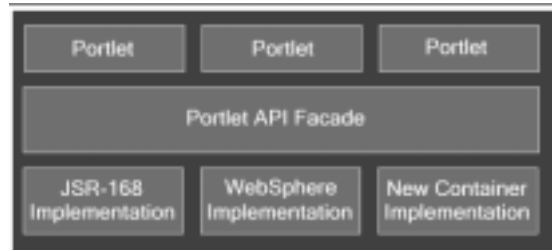
CPS uses a Model-View-Controller design pattern based on the open source Struts and Tiles framework. The View is usually handled by JSP pages. The Model is generally a UCPM SCSObject that represents data retrieved from the content server or image server. The Controller are the Java classes in the com.stellent.portlet.framework package that controls the execution of actions.



See the UCPM API Developer's Guide for more information (ucpm-dev-guide.pdf).

API facade to the portlet container

CPS uses an API facade to the portlet container which allows portlets developed with the Stellent Portlet SDK to be run identically on a variety of platforms. The Stellent Portlet SDK supports the JSR 168-compliant containers WebSphere, WebLogic, Plumtree, and Sun ONE out of the box.



Note: While many portlet containers support the JSR 168 standard, some implementations are not completely compliant or else they differ in their handling characteristics.

Portlet construction

Any portlet you build with the Stellent 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 found, and its associated JSP pages are inserted. The JSP page then models the data that was the result of the action.

Creating a dispatch configuration file

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 Stellent portlets is the SCSPortlet class (there may be different implementations of this per container). This class extends the GenericPortlet class. At initialization, it looks for its configuration file in the "WEB-INF/config/" directory.

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:** Rpecifying 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 has been placed on the request, you may override this error page by creating a new action definition with the keyword name 'error'.

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 overridden 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>
```

Types of child nodes

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

- Default Action node
- Portlet ID node
- Location node
- Action Mappings node
- Tiles-Definitions node

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.

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 collide with other portlets on the same page.

Location node

The <location> node is used to specify another dispatch configuration file in which to load definitions from. It simply takes a path attribute and it 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 in the current configuration XML take precedence over the loaded action definitions.

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 actually execute the action if an unauthenticated portal user tries to execute the action. It defaults to false and is an optional parameter. If it is set to true and an unauthenticated user attempts to execute the method a special system JSP page will displayed asking the user to first 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, so you may tell the framework not to 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 do document contributions asynchronously.

Forward node

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

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 case processing is needed. Notice that this Tile definition overrides the "content" region and changes the JSP page that is used to render this region.

Getting a reference to the portlet API facade

You may at any point in the code execute the following line to get the stateless API facade object:

```
IPortletAPIFacade facade = PortletAPI.getInstance().getPortletAPIFacade();
```

This will return a facade object that is relevant to your current container. If the facade is unable to determine which vendor the container was implemented for, it creates a generic JSR 168-compliant facade object. To write a new detector, see the `com.stellent.portlet.api.PortletVendor` class and add your new detector based on the other implementations.

Creating an action handler

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 name “active.search.doSearch” maps to an action definition created in the configuration file. (See “Action node” on page 20 for additional information.)

The action definition specifies a class name. The class name should be an object that implements the `ISCSActionHandler` interface, which has a variety of methods to implement that 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.

Note: 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 fear of collision.

The return type is that of 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 name "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 to be 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 needs only the document ID of the document that 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 really 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 `SCSObject` on the request by calling `result.setVariable(name, object)`. These objects will now be available to the JSP page rendering the view.

Note: See the UCPM API Developer's Guide for more information ([ucpm-dev-guide.pdf](#)).

```
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.
        ISCSActiveDocumentCheckoutAPI checkoutAPI =
            getCISApplication ().getUCPMAPI ().getActiveAPI
            ().getDocumentCheckoutAPI ();
        ISCSActiveDocumentID docID =
            getCISApplication ().getUCPMAPI ().getActiveAPI
            ()._createActiveDocumentID (cof.getDocumentID (),
            ctx.getAdapterName ());
        ISCSDocumentActionResponse resp =
            checkoutAPI.checkoutFileByID (ctx, docID);

        //Get document info.
        ISCSActiveDocumentInformationAPI 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;
}

```

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. For example, this is an example of the file for the portlets included with Stellent Content Portlet Suite:

```

<%@ 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/il8n.tld" prefix="il8n" %>
<%@ 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 simple lines use the insert tag included with CPS 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 include the JSP page it finds defined in the location specified by the insert tag.

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 only the `ISCSController` class, which requires a variety of methods that the portlet framework uses to control its lifetime.

Fortunately, there is an abstract base class included (the `SCSController` class), which in most cases performs all the operations you need, save one:

```
/**
 * 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 right before the Tile is rendered and any objects you place on the request will be available to the resulting JSP page.

Index

A

- action handler
 - creating 22
 - sample 23
- Action Mappings node 19
 - Action node 20
 - Forward node 21
- Action node 20
 - addToStack attribute 20
 - authRequired attribute 20
 - bean attribute 20
 - class attribute 20
 - name attribute 20
- Ant
 - build.xml file 12
 - for portlet distribution 14
- Apache Ant. See also Ant. 12
- Authenticated Library portlet 9
- Authenticated Search portlet 9

B

- build.properties file 14
 - portal.name variable 14
 - portal.vendor variable 14
- building portlet distributions 14

C

- CheckoutForm 23
- Contribution portlet 9
- controller, creating 25
- CreateURI tag 14

D

- data object 11
- Default Action node 19
 - cacheResult attribute 19
 - view/edit attribute 19
- dispatch configuration file 17
 - Action Mappings node 19
 - child nodes 19
 - coding for active search 18
 - Default Action node 19
 - default keyword 17

- error keyword 18
- Location node 19
- login keyword 18
- naming convention 17
- Portlet ID node 19
- previous keyword 17
- Tiles-Definition node 19
- top-level node 19

E

- error handling tag 15
- Error id tag 15

F

- Federated Search portlet 10
- Forward node 21

G

- GetPreference tag 15

I

- Image Server Search portlet 10
- ISCSActionForm 22
- ISCSActionHandler class 23
- ISCSActionHandler interface 22

J

- JavaDoc 9
- JSR 168
 - compliant containers 17
 - façade object 22
 - Portlet Specification 9

L

- Library portlet 9
- Location node 19

M

- Metadata Admin portlet 9
- Model-View-Controller 12
- Model-View-Controller framework 10

P

- Plumtree 17
- portal.vendor variable 14
- Portlet API Façade 10, 17
- Portlet ID node 19
- portlet preferences 15
- Portlet Specification (JSR 168) 9
- portlet.name variable 14
- PortletBuilder directory 13
- portlets
 - Ant build.xml file 12
 - Authenticated Library 9
 - Authenticated Search 9
 - construction 17
 - Contribution 9
 - dispatch configuration file 17
 - Federated Search 9
 - framework 16
 - Image Server Search 9
 - Library 9
 - Metadata Admin 9
 - Portlet API Façade 10, 17
 - Saved Search 9
 - Search 9
 - source code 12
 - tag libraries 12, 14
 - using Ant for distributions 14
 - using Ant to compile and package 14
 - Workflow Queue 9
- processAction 11

R

- ReferencePortlets directory 13

S

- Saved Search portlet 9
- SCSActionForm 23
- SCSActionForm code 23
- SCSActionHandler class 22
- SCSActionHandler code 23
- SCSActionResult class 23
- Search portlet 9
- Stellent Portlet SDK

- creating a controller 25
- creating a dispatch configuration file 17
- creating a Tile 24
- creating an action handler 22
- portlet framework 16
 - tag libraries 12, 14
- Stellent portlets. See portlets.
- Sun ONE 17

T

- tag libraries 12, 14
- tags
 - CreateURI 14
 - createURI 22
 - error handling 15
 - Error id 15
 - GetPreference 15
 - portlet preferences 15
 - URI creation 14
 - URIAction 15
 - URIParameter 15
- Tile, creating 24
- Tiles-Definitions node 19, 21

U

- Universal Content and Process Management API 10
- URI creation tag 14
- URIAction tag 15
- URIParameter tag 15

W

- WebLogic 17
- WebSphere 17
- Workflow Queue portlet 9