Examine differences between portlet APIs introduction

Time required

To complete this tutorial, you will need approximately **1 hour and 30 minutes**. If you decide to explore other facets of portlet APIs while working on the tutorial, it could take longer to finish.

Prerequisites

In order to complete this tutorial end to end, you should be familiar with the following concepts:

- Basic Java programming
- JSP file coding

Learning objectives

This tutorial examines the differences between the IBM portlet API and the JSR 168 portlet API. The tutorial is divided into several exercises. In the first exercise, you will import sample portlets from the Samples Gallery (**Help > Samples Gallery**). Other exercises discuss the details of the two portlet APIs, using the sample portlets as examples. While completing the exercises, you will learn how the two APIs differ in the following ways:

- Portlet class instances and data
- Java class coding
- Deployment descriptors
- JSP file coding

This tutorial consists of the following exercises:

- Exercise 1.1 shows you how to import the resources.
- Exercise 1.2 explains the conceptual differences between the APIs.
- Exercise 1.3 compares the Java class coding differences between the APIs.
- Exercise 1.4 compares the deployment descriptor differences between the APIs.
- Exercise 1.5 compares the JSP file coding differences between the APIs.
- Exercise 1.6 explains how to select which API to use.
- Exercise 1.7 shows screen captures of the view and edit JSP files used in the bookmark samples.

When you are ready, begin Exercise 1.1: Importing the resources

Exercise 1.1: Importing the resources

In this exercise, you will import the two bookmark sample portlets.

Before you can begin this tutorial, you must first import the required resources:

- Bookmarks (using IBM® portlet API)
- Bookmarks (using JSR 168 API)

Importing the sample project files

Import the two bookmark samples by following these steps:

1. From the Help menu, select **Samples Gallery > Technology samples > Portlet > Basic**.
2. Double-click the `Bookmarks (using IBM® portlet API)` sample. The sample introductory page opens in the right pane.
3. Click **Import the sample**. Importing the sample creates an EAR project named bookmarkIBMEAR and a Portlet project named bookmarkIBM.
4. Import the sample code for the `Bookmarks (using JSR 168 API)`. Importing this sample also creates two projects, bookmarkJSREAR and bookmarkJSR.

About the files used in this tutorial

The samples include the following files:

- BookmarkPortlet.java, the portlet class itself
- PreferencesValidatorImpl.java, the preferences validation class used by the JSR 168 portlet API
- View.jsp, the JSP file used for the View mode of the portlet
- Edit.jsp, the JSP file used for the Edit mode of the portlet
- Portlet.xml, the portlet deployment descriptor

Other portlet samples

The workbench provides several types of portlet coding samples, which are available in the Samples Gallery. From the **Help menu**, select **Samples Gallery > Technology samples > Portlet**.

The samples provided under the Basic Portlet category are listed below. All of these samples illustrate differences between the two portlet APIs. To import any of the samples, open it and click on **Import the sample** on the main page of the sample.

The two bookmarks portlet samples illustrate a portlet that stores Web addresses as bookmarks.

- Bookmarks (using IBM portlet API)
- Bookmarks (using JSR 168 API)

The Content Access portlet samples demonstrate use of the ContentAccessService interface provided by WebSphere® Portal.

- Content Access Service (using IBM portlet API)
- Content Access Service (using JSR 168 API) - (for WebSphere® Portal v5.1 and above)

The cooperative portlet samples show how to exchange information between portlets on the same page, using Click-To-Action (IBM portlet API) and the Property Broker Service (JSR 168 API) .

- Cooperative (using IBM portlet API)
- Cooperative (using JSR 168 API) - (for WebSphere® Portal v5.1 and above)

This tutorial discusses the bookmark portlet samples, explaining the differences between implementing the bookmark code using the IBM portlet API and the JSR 168 portlet API. Some differences that are not illustrated in the bookmark sample are also discussed.

Using JSR 168 portlets with WebSphere Portal

The WebSphere Portal v5.0 test environment within the workbench automatically supports the JSR 168 API. If you are using a remote WebSphere Portal v5.0.2 server, you must configure WebSphere Portal to allow JSR 168 portlets by editing the properties file `ConfigService.properties` in the `<WebSphere_Portal_install_root>/shared/app/config/services` directory, and setting `portal.enable.jsr168 = true`.

Workbench support for the APIs

The workbench provides support for the two APIs in the new Portlet project wizards:

- **File > New > Portal > Portlet Project (JSR 168)**
- **File > New > Portal > Portlet Project**

When you create new portlets in these projects, they will be JSR 168 portlets or IBM portlets, depending on the project they are created in.

Now you are ready to begin Exercise 1.2: Conceptual differences between the APIs.

Exercise 1.2: Conceptual differences between the APIs

Before you begin, you should complete Exercise 1.1: Importing the resources.

In this exercise, you will learn conceptual differences between the two portlet APIs.

Overview

The IBM portlet API was initially developed for WebSphere Portal Version 4. Support for the JSR 168 portlet API was provided beginning with WebSphere Portal version 5.0.2.

The JSR 168 portlet API is a standardized Java™ portlet specification, developed by a group that was jointly led by IBM and Sun, with input from the major vendors of portal servers. Its purpose is to solve portlet compatibility issues between portal servers from different vendors. The initial specification was approved in October, 2003.

WebSphere Portal supports both APIs. It provides two portal containers: the legacy container for IBM portlet API portlets (hereafter referred to as IBM portlets), and the Standard container for JSR 168 portlet API portlets (hereafter referred to as JSR 168 portlets). Portlets running in different containers can reside on the same portal page.

A portal container provides the run-time environment for portlets. It supports the portlet life cycle, which consists of these three phases:

- Initialization
- Request handling
- Destruction

The request handling phase has these two sub-phases:

- The action processing phase, which prepares the information to be presented on the page
- The content rendering phase, where the response information is returned and written to the screen.

Prior to understanding the specific coding differences, you need to understand some basic differences between the two portlet APIs.

Note that you do not have to work directly with the source code for the portlet deployment descriptors as discussed below. The portlet deployment descriptor editor provides a graphical user interface for portlet.xml, and updates the source code for you.

Class instances and data using the IBM portlet API

When a portlet is initially loaded, it is initialized with parameters from the Web deployment descriptor (web.xml). The parameters are defined on the <init-param> element of the <servlet> element. These parameters can be retrieved using the getInitParameter() method of the PortletConfig object. The resulting portlet is an *abstract portlet*.

Before a portlet is used, parameters from the portlet deployment descriptor (portlet.xml) are loaded into a PortletApplicationSettings object or a PortletSettings object. These parameters are set on the <context-param> element of the <concrete-portlet-app> element and on the <config-param> element of the <concrete-portlet> element.

Parameters on the <concrete-portlet-app> element apply to all portlets in the portlet application; they can be retrieved with the PortletSettings.getApplicationSettings() method. The getApplicationSettings() method returns a PortletApplicationSettings object and PortletApplicationSettings.getAttribute() is used to retrieve the individual parameters. Parameters on the <concrete-portlet> element apply to specific portlets; they can

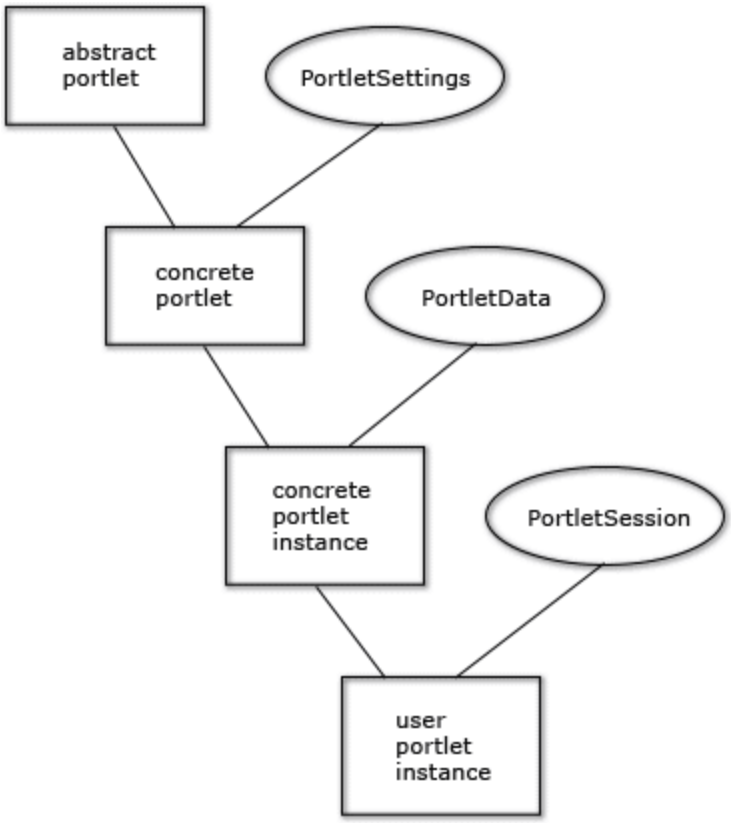
be retrieved with the PortletSettings.getAttribute() method.

The combination of an abstract portlet plus the data from a PortletSettings object is called a *concrete portlet*.

When a portlet is placed on a portal page, a PortletData object is created. The combination of a concrete portlet plus a PortletData object is called a *concrete portlet instance*. The PortletData objects manage the persistent data for the concrete portlet instances. The values are retrieved using the PortletRequest.getData () and PortletData.getAttribute() methods, and stored using the PortletData.setAttribute() and PortletData.store() methods. The data is scoped to a user or a group, depending on the scope of the portal page.

The combination of a concrete portlet instance plus a PortletSession object is called a *user portlet instance*.

The following illustration shows the objects just discussed.



Class instances and data using the JSR 168 portlet API

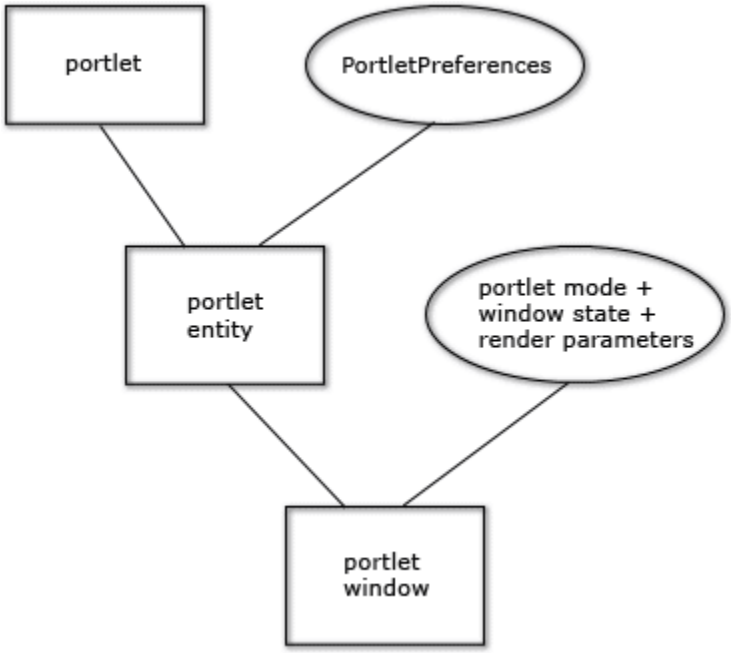
Using the JSR 168 portlet API, initial parameters are defined in portlet.xml with the <portlet-preferences> element. The <init-param> element in web.xml for the IBM portlet API is equivalent to the <init-param> element in portlet.xml for the JSR 168 portlet API.

These initial parameters can be set to read-only, but can be modified at run-time in Config mode. Only an administrator can change read-only initialization parameters. When modified during run-time, a validator class can be called. The name of the validator class is also set on the <portlet-preferences> element. The PortletPreferences object makes these parameters available to the portlet via the RenderRequest.getPreferences(), PortletPreferences.getValue(), and PortletPreferences.getValues() methods. The <portlet-preferences> element is equivalent to the <config-param> element plus a PortletData object in the IBM portlet API.

The combination of a PortletPreferences object and a portlet is known as a *portlet entity*. A *portlet window* is

defined as the combination of the portlet mode (edit, view), the portlet window state (normal, maximized, minimized), and the render parameters. A portal page can contain more than one portlet window for a given portlet, each associated with a specific mode, state and set of render parameters.

The following illustration shows the objects just discussed.



Portlet life cycle

Both portlet APIs use a life cycle that consists of an initialization phase, a request processing phase, and a destruction phase. The request processing phase is divided into sub-phases: action processing and content rendering. Details of the request processing phase are discussed in Request processing. The specific methods called in each phase are listed below.

IBM portlet API life cycle methods

init(PortletConfig)

The init() method is called when the portlet is placed in service. The PortletConfig parameter provides access to the PortletContext object.

initConcrete(PortletSettings)

The initConcrete() method is used to initialize the concrete portlet with the PortletSettings object.

service(PortletRequest, PortletResponse)

The service() method is called when the portlet is required to render its content. The service method is called many times during the life cycle of a portlet. Service calls methods such as doView() and doEdit(), depending on the window state of the portlet. The service method is usually not overridden in the implementing portlet class.

destroyConcrete(PortletSettings)

The destroyConcrete() method is called when the concrete portlet is taken out of service.

destroy(PortletConfig)

The destroy() method is called when the portlet is taken out of service, providing a place for cleanup of resources.

JSR 168 portlet API life cycle methods

init(PortletConfig)

The init() method is called when the portlet is placed in service. The PortletConfig parameter provides access to the PortletContext object.

render(RenderRequest, RenderResponse)

The render() method is called when the portlet is required to render its content. It calls methods such as doEdit() and doView(), depending on the window state of the portlet. The render method is usually not overridden in the implementing portlet class.

processAction(ActionRequest, ActionResponse)

The processAction() method requires ActionRequest and ActionResponse objects as parameters.

destroy()

The destroy() method is called when the portlet is taken out of service, providing a place for cleanup of resources.

Request processing

Both portlet APIs use two-phase request processing. Actions (or events) are processed first, then the rendering phase is invoked. In the IBM portlet API, the action phase is invoked via the actionPerformed() method, and the rendering phase is invoked by the service() method. In the JSR 168 portlet API, the phases are invoked via the processAction() and render() methods.

A big difference between the two APIs is that the JSR portlet API uses different request and response objects in the action and render phases, while the IBM portlet API uses the same objects in the two phases. Using the IBM portlet API, you can set attributes on the request and response objects during event processing, and retrieve the values during the rendering phase. Using the JSR portlet API, attribute values can be passed using the session object or using render parameters.

Another difference is that the IBM portlet API's actionPerformed() method uses its ActionEvent parameter to gain access to the PortletRequest object. The JSR 168 portlet API's processAction() method has the parameters ActionRequest and ActionResponse, which implement the PortletRequest and PortletResponse interfaces.

Now you are ready to begin Exercise 1.3: Comparing Java™ class differences.

Exercise 1.3: Comparing Java™ class differences

Before you begin, you should complete Exercise 1.2: Conceptual differences between the APIs.

In this exercise, you will learn differences in Java class coding between the two portlet APIs. Examine the two versions of the BookmarkPortlet Java class. Notice these basic differences between the two APIs:

Importing basic portlet classes

The portlet classes that the two APIs import differ.

IBM portlet API

```
import org.apache.jetspeed.portlet.*;
```

JSR 168 portlet API

```
import javax.portlet.*;
```

Java class inheritance

The two APIs inherit from different classes. The IBM portlet API extends org.apache.jetspeed.portlet.PortletAdapter which provides a default implementation of the org.apache.jetspeed.portlet.Portlet interface. This Portlet class extends HttpServlet, so IBM portlets are a type of servlet. The JSR 168 portlet API provides a javax.portlet.GenericPortlet class which implements the javax.portlet.Portlet interface.

IBM portlet API

```
public class BookmarkPortlet extends PortletAdapter implements ActionListener
```

JSR 168 portlet API

```
public class BookmarkPortlet extends GenericPortlet
```

Request and response objects

The names of the request and response objects on the render (JSR 168 API) or service (IBM API) methods, such as doView() and doEdit(), differ. The IBM portlet API uses PortletRequest and PortletResponse objects; the JSR 168 API uses RenderRequest and RenderResponse objects. RenderRequest and RenderResponse extend the PortletRequest and PortletResponse objects, respectively, providing common functionality.

IBM portlet API

```
public void doEdit(PortletRequest request, PortletResponse response)
```

JSR 168 portlet API

```
public void doEdit(RenderRequest request, RenderResponse response)
```

Including JSP files

The IBM portlet API uses the PortletContext object to include JSP files; the JSR 168 portlet API uses the PortletRequestDispatcher object. The include action invokes the JSP file specified.

IBM portlet API

```
getPortletConfig().getContext().include(EDIT_JSP, request, response);
```

JSR 168 portlet API

```
PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(jspName);
rd.include(request, response);
```

Portlet data

The IBM portlet API stores user data in a PortletData object. The JSR 168 portlet API stores similar information in a PortletPreferences object.

IBM portlet API

```
PortletData prefs = portletRequest.getData()
```

JSR 168 portlet API

```
PortletPreferences prefs = renderRequest.getPreferences()
```

Action processing

In the IBM portlet API, the Java class must implement the ActionListener interface by providing an actionPerformed() method. Using the JSR 168 portlet API, the Java class must provide a processAction() method; no listener is needed.

IBM portlet API

```
public void actionPerformed(ActionEvent event) throws PortletException
```

JSR 168 portlet API

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

Namespace encoding

Namespace encoding is used to ensure that variables used within a portlet are unique within the portal container. The excerpts below also show namespace encoding methods for use in a JSP file.

IBM portlet API

```
in a Java class: PortletResponse.encodeNamespace()
in a JSP file:   <portletAPI:encodeNamespace/>
```

JSR 168 portlet API

```
in a Java class: RenderResponse.getNamespace()
in a JSP file:   <portlet:namespace/>
```

Now you are ready to begin Exercise 1.4: Comparing deployment descriptor differences.

Exercise 1.4: Comparing deployment descriptor differences

Before you begin, you should complete Exercise 1.3: Comparing Java class differences.

In this exercise, you will learn differences between the deployment descriptors for the two portlet APIs. Examine the two versions of the portlet deployment descriptor (portlet.xml). The basic differences illustrated in the samples are shown below. To edit the portlet deployment descriptor, use the portlet deployment descriptor editor.

Tagging rules for portlet.xml

The tagging rules for the IBM portlet API are defined by a DTD; the JSR 168 portlet API is defined by an XML schema. This requires different XML definition statements at the top of the portlet deployment descriptor.

IBM portlet API

```
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN"
          "portlet_1.1.dtd ">
```

JSR 168 portlet API

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             version="1.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             id="bookmark_03_jsr.1">
```

The id attribute on the <portlet-app> element

The two APIs use different names for the id attribute on the <portlet-app> element. The IBM portlet API uses uid, while the JSR 168 portlet API uses id.

IBM portlet API

```
<portlet-app uid="com.ibm.etools.portal.portletexamples.bookmark.legacy"
```

JSR 168 portlet API

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             version="1.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd "
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             id="bookmark_03_jsr.1">
```

The href attribute on the <portlet> element

Using the IBM portlet API, the href attribute of the <portlet> element must point to the corresponding id of the <servlet> element in the Web deployment descriptor (web.xml); using the JSR 168 portlet API, the id attribute on the <portlet-app> element uniquely identifies the portlet to the server. JSR 168 portlets are not servlets, and do not need the reference to web.xml.

IBM portlet API

```
portlet.xml: <portlet id="Bookmark" href="WEB-INF/web.xml#Servlet_1086938566718"
web.xml:     <servlet id="Servlet_1086938566718">
```

JSR 168 portlet API

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             version="1.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd "
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
```

```
             id="bookmark_03_jsr.1">
```

Portlet versions

For the IBM portlet API, use the major-version and minor-version attributes of the <portlet-app> and <portlet> elements; for the JSR 168 portlet API, use the version attribute of the <portlet-app> element.

IBM portlet API

```
<portlet-app uid="com.ibm.etools.portal.portletexamples.bookmark.legacy"
             major-version="1" minor-version="0">
<portlet id="Bookmark" href="WEB-INF/web.xml#Servlet_1086938566718"
             major-version="1" minor-version="0">
```

JSR 168 portlet API

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             version="1.0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd "
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             id="bookmark_03_jsr.1">
```

Supported markup types

In the portlet deployment descriptor, the IBM portlet API declares supported markup types, while the JSR 168 portlet API declares supported MIME types. IBM provides an extension, the `wps.markup` initialization parameter, that lets you define the supported markup types for JSR 168 portlets. This is used to differentiate between markup types, like HTML and cHTML, that use the same MIME type.

IBM portlet API

```
<supports>
   <markup name="html">
      <view />
      <edit />
   </markup>
   <markup name="chtml">
      <view />
   </markup>
</supports>
```

JSR 168 portlet API

```
<init-param>
   <name>wps.markup</name>
   <value>html,chtml</value>
</init-param>

<supports>
   <mime-type>text/html</mime-type>
   <portlet-mode>VIEW</portlet-mode>
   <portlet-mode>EDIT</portlet-mode>
</supports>
```

Supported modes

Supported modes must be defined in the portlet deployment descriptor for both APIs, although the syntax is slightly different. Both APIs support edit, view and help modes. The IBM portlet API also supports a config mode. The JSR 168 portlet API supports custom modes such as about, preview, print, edit defaults, and config. For both APIs, the view mode is required; all other modes are optional.

IBM portlet API

```
<supports>
   <markup name="html">
      <view />
      <edit />
   </markup>
```

```
</supports>
```

JSR 168 portlet API

```
<supports>
   <portlet-mode>VIEW</portlet-mode>
   <portlet-mode>EDIT</portlet-mode>
</supports>
```

Allowable window states

The normal state is automatically provided for both APIs. With the IBM portlet API, other states, such as solo, must be explicitly declared. With the JSR 168 portlet API, maximized and minimized states are also automatically provided; custom states must be explicitly declared.

IBM portlet API

```
<allows>
   <maximized/>
   <minimized/>
</allows>
```

JSR 168 portlet API

no custom states are shown in the example, but the <custom-window-state> element would be used

Localization

Some localized strings that define a portlet are defined in the portlet deployment descriptor. The IBM portlet API's portlet.xml defines localized values for the title, short title, description and keywords. The IBM API also has a setting for the default locale. The JSR 168 portlet API defines localized values for the portlet description and display name in portlet.xml. Values for title, short title and keywords are located in a portlet resource bundle. As shown below, the portlet resource bundle, `Portlet.properties`, is located in the `nls` directory of the sample. The JSR 168 API has settings to define the resource bundle and supported locales.

IBM portlet API

```
<default-locale>en</default-locale>
<language locale="en">
   <title>Bookmark portlet (IBM)</title>
   <title-short>Bookmark</title-short>
   <keywords>Bookmark</keywords>
</language>
<language locale="de">
   <title>Lesezeichen Portlet (IBM)</title>
   <title-short>Lesezeichen</title-short>
   <keywords>Lesezeichen, Bookmark</keywords>
</language>
```

JSR 168 portlet API

```
<resource-bundle>nls.Portlet</resource-bundle>
<supported-locale>en</supported-locale>
<supported-locale>de</supported-locale>
<description xml:lang="EN">English description</description>
<display-name xml:lang="EN">English display name</display>-name>
<description xml:lang="DE">German description</description>
<display-name xml:lang="DE">German display name</display>-name>

Sample portlet resource bundle, Portlet.properties:
javax.portlet.title = Bookmark Portlet
javax.portlet.short-title = Bookmark
javax.portlet.keywords = Bookmark
```

Cache expiration

In the IBM portlet API, portlets can explicitly invalidate the cache (*invalidation-based caching*) using the invalidateCache() method of the PortletRequest object. The default cache expiration and scope are set in the portlet deployment descriptor. The value of the <expires> element can be 0 (always expires), -1 (never expires), or the number of seconds before expiration. The <shared> element value of `no` means that the

cache is not shared between portlet instances.

The JSR portlet API uses *expiration-based caching*, where the cache expiration time is defined in the deployment descriptor, but portlets can reset the value using the EXPIRATION_CACHE value with the setAttribute() method of the RenderResponse object. The <expiration-cache> element uses the same values as the IBM portlet API uses for the <expires> element.

IBM portlet API

```
<cache>
   <expires>-1</expires>
   <shared>no</shared>
</cache>
```

JSR 168 portlet API

```
<expiration-cache>0</expiration-cache>
```

Now you are ready to begin Exercise 1.5: Comparing JSP file coding differences.

Exercise 1.5: Comparing JSP file coding differences

Before you begin, you should complete Exercise 1.4: Comparing deployment descriptor differences.

In this exercise, you will learn the differences in JSP file coding between the two portlet APIs. Examine the two versions of the Edit and View JSP files. The basic differences are discussed below.

Tag libraries

IBM portlet API tags are declared in the `portlet.tld` tag library. The tags use the *portletAPI* prefix. The JSR 168 portlet API uses the `std-portlet.tld` tag library and the *portlet* prefix. Other tag libraries, such as the JavaServer Pages Standard Tag Library (JSTL), defined in `fmt.tld` , can also be used. As shown in the sample code below, the JSTL tag library uses the *fmt* prefix.

IBM portlet API

```
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init />

<%@ taglib prefix="fmt" uri="/WEB-INF/tld/fmt.tld" %>
<fmt:setBundle basename="nls.Text" />
```

JSR 168 portlet API

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects />

<%@ taglib prefix="fmt" uri="/WEB-INF/tld/fmt.tld" %>
<fmt:setBundle basename="nls.Text"/>
```

Making objects available to JSP files

In the IBM portlet API, the <portletAPI:init> tag makes the PortletRequest, PortletResponse, and PortletConfig objects available to JSP files. In the JSR 168 portlet API, the <portlet:defineObjects> tag makes the RenderRequest, RenderResponse, and PortletConfig objects available to JSP files.

IBM portlet API

```
<portletAPI:init />
```

JSR 168 portlet API

```
<portlet:defineObjects />
```

MIME type declarations

The two APIs differ in how they set the MIME type for the render response. IBM portlets declare the MIME type on the page directive of the JSP file. JSR 168 portlets declare the MIME type using the setContentType () method of the RenderResponse object in the render methods (doView(), doEdit()).

IBM portlet API

```
<%@ page contentType="text/html"
         import="java.util.*,
                 com.ibm.etools.portal.portletexamples.bookmark.legacy.*,
                 org.apache.jetspeed.portlet.*"
         session="false"%>
```

JSR 168 portlet API

```
response.setContentType("text/html");
```

Portlet references

References to a portlet, portlet page or portlet resource must be encoded in a portlet URI (JSR 168 uses the term *URL*). The IBM portlet API uses createURI to point to the calling portlet in the current mode, and createReturnURI to point to the calling portlet in the previous mode. The JSR 168 portlet API creates URLs for the action phase (actionURL) and the render phase (renderURL).

IBM portlet API

```
in a JSP file:   <portletAPI:createURI/>
                 <portletAPI:createReturnURI/>
in a Java class: PortletResponse.createURI()
                 PortletResponse.createReturnURI()
```

JSR 168 portlet API

```
in a JSP file:   <portlet:actionURL/>
                 <portlet:renderURL/>
in a Java class: RenderResponse.createActionURL()
                 RenderResponse.createRenderURL()
```

URL encoding

Portlet JSP files must encode URLs that refer to resources in the associated WAR file, such as images, applets, and other JSP files. The JSR 168 portlet API also requires that the context path be included in the URL.

IBM portlet API

```
<%= response.encodeURL("images/photo01.jpg") %>
```

JSR 168 portlet API

```
<%= renderResponse.encodeURL(renderRequest.getContextPath() + "/images/photo01.jpg") %>
```

Namespace encoding

Namespace encoding, for both Java classes and JSP files, is discussed in Namespace encoding in exercise 1.3.

Resource bundles

The example code for both APIs shows use of the JSTL tag <fmt:setBundle>. This tag refers to a standard Java resource bundle, `Text.properties`, in the `JavaSource/nls` directory of the samples. Compare this to resource bundles that define the portlet.

IBM portlet API

```
<fmt:setBundle basename="nls.Text" />
```

JSR 168 portlet API

```
<fmt:setBundle basename="nls.Text" />
```

Now you are ready to begin Exercise 1.6: Deciding which API to use.

Exercise 1.6: Deciding which API to use

Before you begin, you should complete Exercise 1.5: Comparing JSP file coding differences.

In this exercise, you will learn how to choose which portlet API to use.

Deciding which API to use

IBM will continue to support the IBM portlet API in current and future releases of WebSphere Portal. IBM is committed to open standards, and will continue to enhance the JSR 168 specification. WebSphere Portal provides full support for JSR 168.

The recommended practice is to use the JSR 168 portlet API unless you need a function that is only available in the IBM portlet API. The JSR 168 portlet API is required for portability and compatibility, or if your portlet will be published as a Web Service for Remote Portlets (WSRP) service. WSRP is a portal-based standard used to integrate remote portlets, provided by Web services, to the local portal page.

Concepts unique to the IBM portlet API

The following functions are only available using the IBM portlet API.

- Events can be sent between portlets.
- Portlets can add content to the portal navigation menu.
- Portlets can explicitly invalidate cached content.
- Portlets can use portlet services provided by WebSphere Portal. Some services will be available to JSR 168 portlets in WebSphere Portal v5.1.
- Portlets on the same page can exchange properties using the Property broker (Click-to-Action) service.

Concepts unique to the JSR 168 portlet API

The following functions are only available using the JSR 168 portlet API.

- Portlets can store their navigational state using render parameters.
- Portlets can make data available to the entire Web application.
- During the action phase, portlets can redirect to other Web resources.
- Portlets can adapt to the calling portal using PortletContext objects.
- Portlets can access a portal user profile.
- Portlets can validate preference properties using a preference validator class.

Now you are ready to begin Exercise 1.7: Viewing the sample portlets.

Exercise 1.7: Viewing the sample portlets

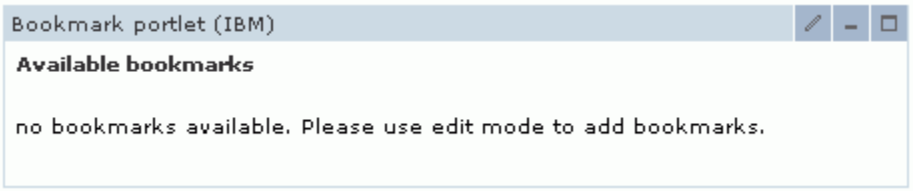
Before you begin, you should complete Exercise 1.6: Deciding which API to use.

In this exercise, you will see the rendered output of the portlets in the two bookmark sample portlets.

The following images show the rendered portlets.

View mode

The following images show both portlets after they have been placed on a page. No bookmarks have been added. The View.jsp file for the JSR 168 portlet displays the bookmark predefined in the <portlet-preferences> element of portlet.xml.



Edit mode

The following images show the edit mode for both portlets.

Final view

The following images show both portlets after the same edits have been performed.

Finish your tutorial by reviewing the materials in the Summary.

Examine differences between portlet APIs summary

Congratulations! This tutorial has taught you the differences between the two primary portlet APIs that are available to portlet programmers: the IBM portlet API and the JSR 168 portlet API.

Completed learning objectives

If you have completed all of the exercises, you should now understand these concepts:

- Portlet class instances and data
- Java class coding
- Deployment descriptors
- JSP file coding

More information

If you want to learn more about the topics covered in this tutorial, consider the following sources:

Web sites

- IBM portlet API for Portal 5.0
- JSR 168 portlet API specification
- Portlet Development Best Practices and Coding Guidelines
- Best practices: Developing portlets using JSR 168 and WebSphere Portal V5.02
- Documentation Libraries for WebSphere Portal
- WebSphere Portal zone

Workbench online help

- Portlet APIs
- Developing portlet applications in **Help > Help Topics**.
- Developing portal applications in **Help > Help Topics**.