



Portlet Development Guide

Working with the Portlet API 1.2
Java Server Pages in Portlets
Security and Single Sign On

Second Edition
March 14, 2003

Authors:

Ted Buckner
Stephan Hesmer
Peter Fischer
Ingo Schuster
WebSphere Portal Development

1.	<i>Abstract</i>	5
2.	<i>Overview</i>	6
2.1.	Portlets and the Servlet API	6
2.2.	Portlet deployment descriptor	8
2.3.	Portlet Concepts	9
2.4.	Portlet Applications	10
2.4.1.	Concrete Portlet Application	11
2.5.	Portlet modes	11
2.6.	Portlet states	12
2.7.	Examples in this document - the Bookmark Portlet	12
3.	<i>Portlet API - Basic elements</i>	14
3.1.	Portlet	14
3.1.1.	getLastModified().....	15
3.1.2.	PortletAdapter	15
3.1.3.	Example.....	15
3.2.	Core Objects	17
3.2.1.	PortletRequest	17
3.2.2.	PortletResponse	19
3.2.3.	PortletSession	20
3.2.4.	Example.....	20
3.3.	Listeners	22
3.3.1.	PortletSessionListener	22
3.3.2.	PortletPageListener	23
3.3.3.	PortletTitleListener.....	24
3.3.4.	Complete Lifecycle	24
3.3.5.	Example.....	24
3.4.	Configuration Objects	27
3.4.1.	PortletConfig	27
3.4.2.	PortletSettings	27
3.4.3.	PortletApplicationSettings	27
3.4.4.	PortletData.....	28
3.4.5.	Example.....	28
3.5.	Miscellaneous Objects	30
3.5.1.	PortletContext.....	30
3.5.2.	PortletWindow	32
3.5.3.	User	33
3.5.4.	Example.....	33
3.6.	The scope of the portlet objects	34
4.	<i>Java Server Pages</i>	36
4.1.	Portlet API Tags	36
4.1.1.	if	36
4.1.2.	log.....	37

4.1.3.	text.....	37
4.1.4.	dataLoop.....	37
4.1.5.	dataAttribute.....	38
4.1.6.	settingsLoop.....	38
4.1.7.	settingsAttribute.....	39
4.1.8.	CreateReturnURI.....	39
4.1.9.	createURI.....	40
4.1.10.	URIParameter.....	40
4.1.11.	URIAction.....	40
4.1.12.	encodeNamespace.....	41
4.1.13.	init.....	41
4.1.14.	bidi.....	42
4.1.15.	encodeURI (deprecated).....	42
4.2.	The example using a JSP.....	42
4.3.	Reserved Parameter Names.....	44
5.	<i>Portlet API - Advanced elements.....</i>	45
5.1.	Portlet events.....	45
5.1.1.	Action events.....	46
5.1.2.	Window events.....	47
5.1.3.	Message events.....	47
5.2.	PortletSettingsAttributesListener.....	49
5.3.	PortletApplicationSettingsAttributesListener.....	49
5.4.	Portlet Caching.....	49
5.4.1.	Refreshing the portlet cache.....	50
5.5.	Portlet Services.....	50
5.5.1.	Writing a portlet service.....	51
5.5.2.	ContentAccessService.....	53
6.	<i>Deployment Descriptors.....</i>	55
6.1.	Web application deployment descriptor.....	55
6.2.	Portlet deployment descriptor.....	55
6.3.	Linking the servlet, portlet, and concrete portlet.....	60
6.4.	Guidelines for portlet application UIDs.....	61
7.	<i>Compiling and testing portlets.....</i>	62
7.1.	Setting up a portlet development environment.....	62
7.2.	Compiling Java source.....	62
7.3.	Creating the deployment descriptors.....	63
7.4.	Setting up the WAR file directory structure.....	63
7.4.1.	Support for multiple markups and locales.....	65
7.5.	Packaging a portlet and resources into a WAR file.....	66
7.5.1.	Testing the sample.....	66
8.	<i>Credential Vault Service.....</i>	68
8.1.	Credential Vault organization.....	68

8.2. Vault Segments	68
8.3. Vault slots.....	69
8.4. Credential objects.....	69
8.4.1. Storing credential objects in the PortletSession	71
8.5. Credential Vault usage scenarios.....	72
8.5.1. Intranet Lotus Notes mail portlet	72
8.5.2. Stock of inventory portlet.....	72
8.5.3. Internet Mail federating POP3 portlet.....	72
8.6. Methods of the CredentialVaultService	73
8.7. Programming Example.....	77
8.8. Using JAAS to retrieve the user's credentials	78
8.8.1. Basic authentication sample	80
8.8.2. LTPA example	82
9. References.....	83
10. Trademarks.....	84

Figures

Figure 1: View of a portlet on the application server and portal server	8
Figure 2: Manifestations of a portlet in the portal server	9
Figure 3: Portlet application	11
Figure 4: Vault segments and vault implementations	69

1. Abstract

The purpose of this document is to show how to develop a portlet using the Portlet API, introducing concepts of the Portlet API along the way. Elements of the Portlet API are described with the help of an example that progresses from a simple portlet with no output to a complex portlet application with more advanced features. Portlet development for the WebSphere Portal environment is also described.

2. Overview

Portals are Web sites that serve as a starting point to information and applications on the Internet or from an intranet. Early Internet portals, such as Yahoo, Excite, and Lycos, categorized Web content and provided search facilities. Over the years, portals have evolved to provide aggregation of content from diverse sources, such as rich text, video, or XML, and personalized services, such as user customization of layout and content.

To accommodate the aggregation and display of such diverse content, a portal server must provide a framework that breaks the different portal components into *portlets*. Each portlet is responsible for accessing content from its source (for example, a Web site, database, or email server) and transforming the content so that it can be rendered to the client. In addition, a portlet might be responsible for providing application logic or storing information associated with a particular user. The portal server provides a framework of services to make the task of writing and managing portlets easier.

From a user's perspective, a portlet is a window in the portal that provides a specific service or information, for example, a calendar or news feed. From an application development perspective, portlets are pluggable modules that are designed to run inside a portlet container of a portal server.

The portlet container provides a runtime environment in which portlets are instantiated, used, and finally destroyed. Portlets rely on the portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, lookup credentials, and to store persistent data. The Portlet API provides standard interfaces for these functions. The portlet container is not a stand-alone container like the servlet container. Instead, it is implemented as a thin layer on top of the servlet container and reuses the functionality provided by the servlet container.

IBM is working with other companies to standardize the Portlet API, making portlets interoperable between portal servers that implement the specification. The Portlet API offered in WebSphere Portal Version 4 is the first step toward the Portlet API standardization. For more information about the portlet specification, see

<http://jcp.org/jsr/detail/168.jsp>

2.1. Portlets and the Servlet API

The abstract `Portlet` class is the central abstraction of the Portlet API. The `Portlet` class extends `HTTPServlet`, of the Servlet API. All portlets extend the `Portlet` class indirectly, and inherit from `HttpServlet` as shown:

```
...
+--javax.servlet.http.HttpServlet
  |
  +--org.apache.jetspeed.portlet.Portlet
    |
    +--org.apache.jetspeed.portlet.PortletAdapter
      |
      +--com.myCompany.myApplication.myPortlet
```

Therefore, portlets are a special type of servlet, with properties that allow them to easily plug into and run in the portal server. Unlike servlets, portlets cannot send redirects or errors to browsers directly, forward requests, or write arbitrary markup to the output stream.

Generally, portlets are administered more dynamically than servlets. The following updates can be applied without having to start and restart the portal server:

- Portlet applications consisting of several portlets can be installed and removed using the portal administration user interface.
- The settings of a portlet can be changed by an administrator with appropriate access rights.
- Portlets can be created and deleted dynamically by administration portlets. For example, the clipping portlet can be used to create new portlet instances whenever an administrator creates a new clipping.

The portlet container relies on the J2EE architecture implemented by WebSphere Application Server. As a result, portlets are packaged in WAR files similar to J2EE Web applications and are deployed like servlets. Like other servlets, a portlet is defined to the application server using the web application deployment descriptor (web.xml). This file defines the portlet's class file, the servlet mapping and read-only initialization parameters.

Figure 1 shows the portlet after its WAR file is deployed. For each portlet deployed on the portal server, it creates a servlet, or *portlet class instance*, on the application server.

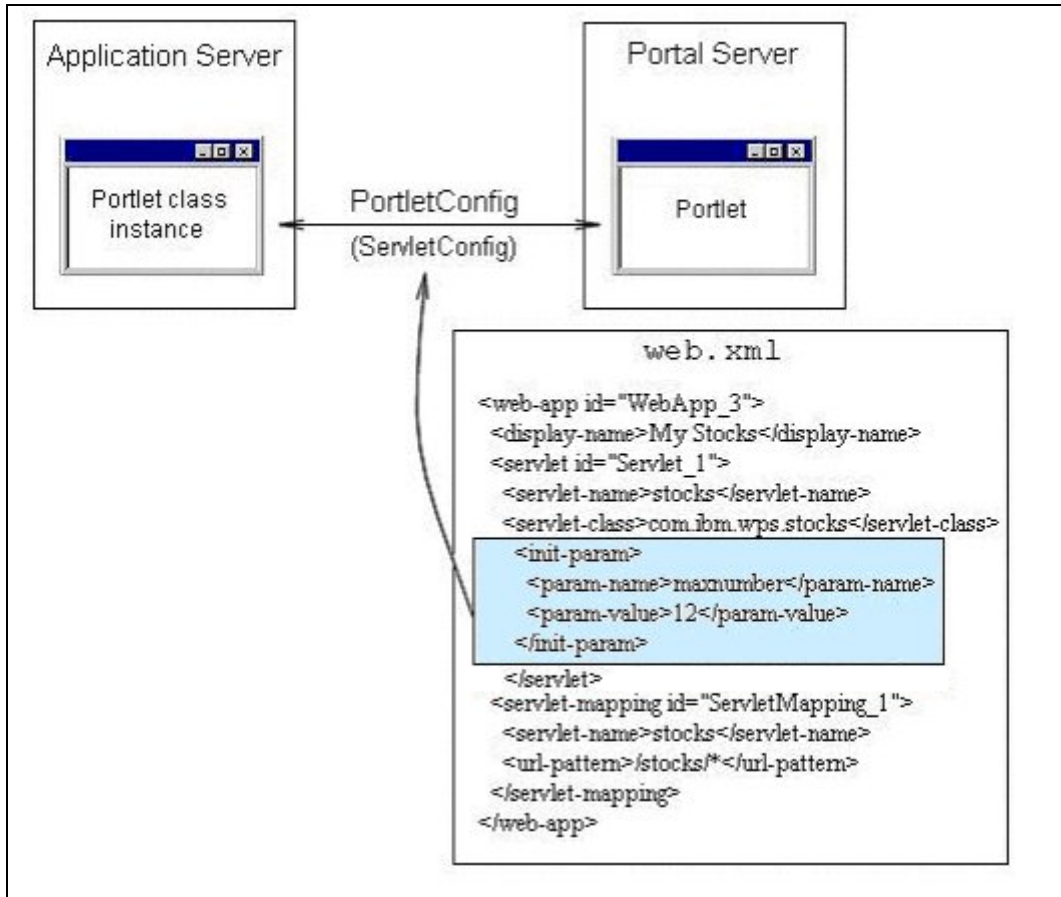


Figure 1: View of a portlet on the application server and portal server

The initialization parameters are set by the portlet developer and can be read by the portlet using the [PortletConfig](#) object. The Web application deployment descriptor can contain multiple servlets, each defined by the `<servlet>` element. In addition, each servlet definition can point to the same portlet class file, thus creating different `PortletConfig` objects with different initialization parameters for each portlet class instance. For more information, see [Web application deployment descriptor](#).

2.2. Portlet deployment descriptor

In addition to the servlet descriptor, portlets must also provide a portlet deployment descriptor (`portlet.xml`) to define the portlet's capabilities to the portal server. This information includes configuration parameters specific to a particular portlet or portlet application as well as general information that all portlets provide, such as the type of markup that the portlet supports. The portal server uses this information to provide services for the portlet. For example, if a portlet registers its support for help and edit mode in the portlet deployment descriptor, the portal server will render icons to allow the user to invoke the portlet's help and edit pages.

The following is an example portlet deployment descriptor with the minimum tags.

Example portlet deployment descriptor


```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN" "portlet 1.1.dtd">
<portlet-app-def>
  <portlet-app uid="com.myCompany.myPortletApp.54321">
    <portlet-app-name>My portlet application</portlet-app-name>
    <portlet id="Portlet_1" href="WEB-INF/web.xml#Servlet_1">
      <portlet-name>My portlet</portlet-name>
      <supports>
        <markup name="html">
          <view output="fragment"/>
        </markup>
      </supports>
    </portlet>
  </portlet-app>

  <concrete-portlet-app uid="com.myCompany.myConcretePortletApp.54321">
    <portlet-app-name>My concrete portlet application</portlet-app-name>
    <concrete-portlet href="#Portlet_1">
      <portlet-name>My concrete portlet</portlet-name>
      <default-locale>en</default-locale>
      <language locale="en US">
        <title>My portlet</title>
      </language>
    </concrete-portlet>
  </concrete-portlet-app>
</portlet-app-def>

```

For detailed information, see [Deployment descriptors](#).

2.3. Portlet Concepts

The following figure shows different variations of a portlet as it is created, placed on a page, and accessed by users. Notice that the first two steps involve the use of persistent data, but for the third step, the data is available only for the duration of the session.

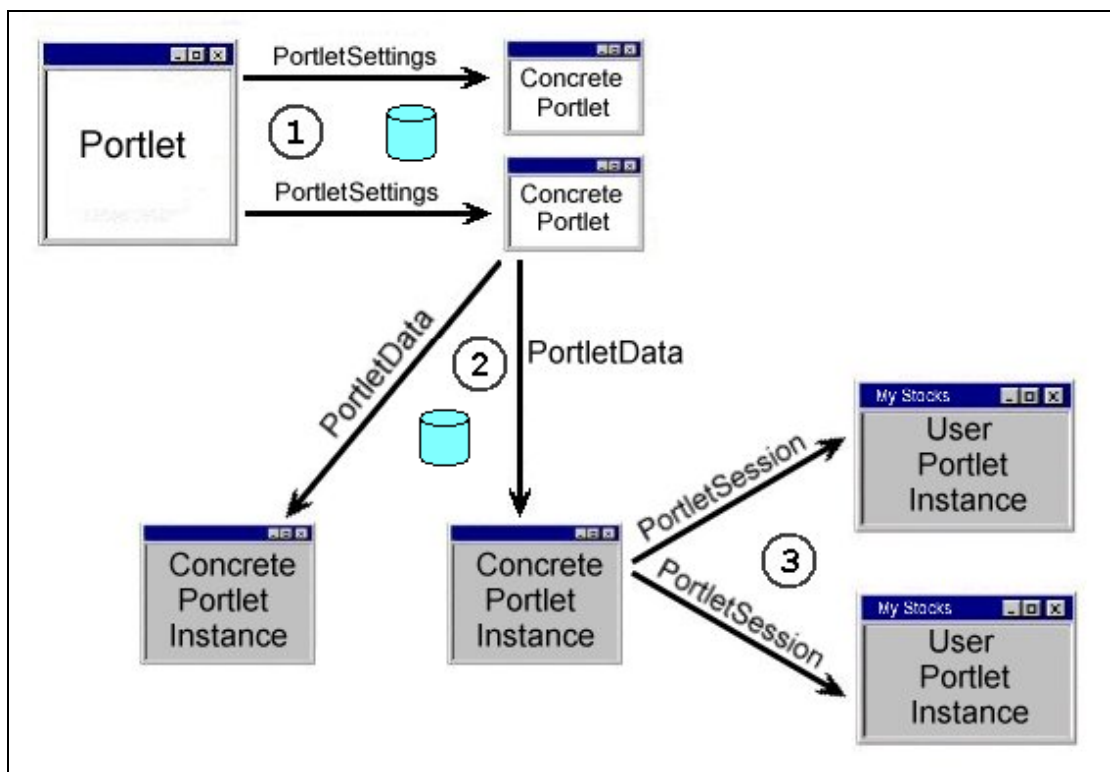


Figure 2: Manifestations of a portlet in the portal server

① The portal administrator uses the administrative interface to deploy a new portlet application WAR file or install a copy of a portlet. Either action creates a *concrete portlet*, which is a portlet parameterized by a single [PortletSettings](#) object. There can be many concrete portlets for each portlet. At least one concrete portlet is defined in the portlet deployment descriptor. After deployment, any number of concrete portlets can be created by the administrator. PortletSettings are read/write accessible and persistent. The PortletSettings contains configuration parameters initially defined in the portlet deployment descriptor and changeable by the administrator during runtime.

The use of concrete portlets allows many instances of a portlet to run with different configurations, without creating extra portlet class instances. During the lifecycle of a single portlet, many concrete portlets can be created and destroyed. There is no object that explicitly represents the concrete portlet. The same concrete portlet can be shared across many users.

② The portlet is placed on a page by a user or an administrator. This creates a *concrete portlet instance*, which is a concrete portlet parameterized by a single [PortletData](#) object. There can be many concrete portlet instances per concrete portlet. PortletData stores persistent information for a portlet that has been added to a page. This information cannot be changed by the administrator; it may only be written by the portlet itself. For example, a user can edit a stock quotes portlet and save a list of stock symbols for the companies to track.

③ The scope of the PortletData depends on the scope of the page that the concrete portlet is on.

1. If an administrator puts a concrete portlet on a group page, then the PortletData object contains data stored for the group of users. This holds true for a group of users who have view access to the page. However, if users have edit access to the portlet on a group page, then a new concrete portlet instance is created for each user that edits the portlet. In this case, PortletData contains data for each user that edits the portlet.
2. If a concrete portlet is put on a user's page, the PortletData contains data for that user.

When a user accesses a page that contains a portlet, that creates a *user portlet instance*. A user portlet instance is a concrete portlet instance parameterized by a single PortletSession. There can be many user portlet instances per concrete portlet instance. The PortletSession stores *transient* information related to a single use of the portlet.

2.4. Portlet Applications

Portlet applications provide the means to package a group of related portlets that share the same context. The context contains all resources, for example, images,

properties files, and classes. All portlets must be packaged as part of a portlet application.

2.4.1. Concrete Portlet Application

A concrete portlet application is a portlet application parameterized with a single [PortletApplicationSettings](#) object. For each portlet application, there may be many concrete portlet applications. PortletApplicationSettings are read/write accessible and persistent. There is no object that explicitly represents the concrete portlet application.

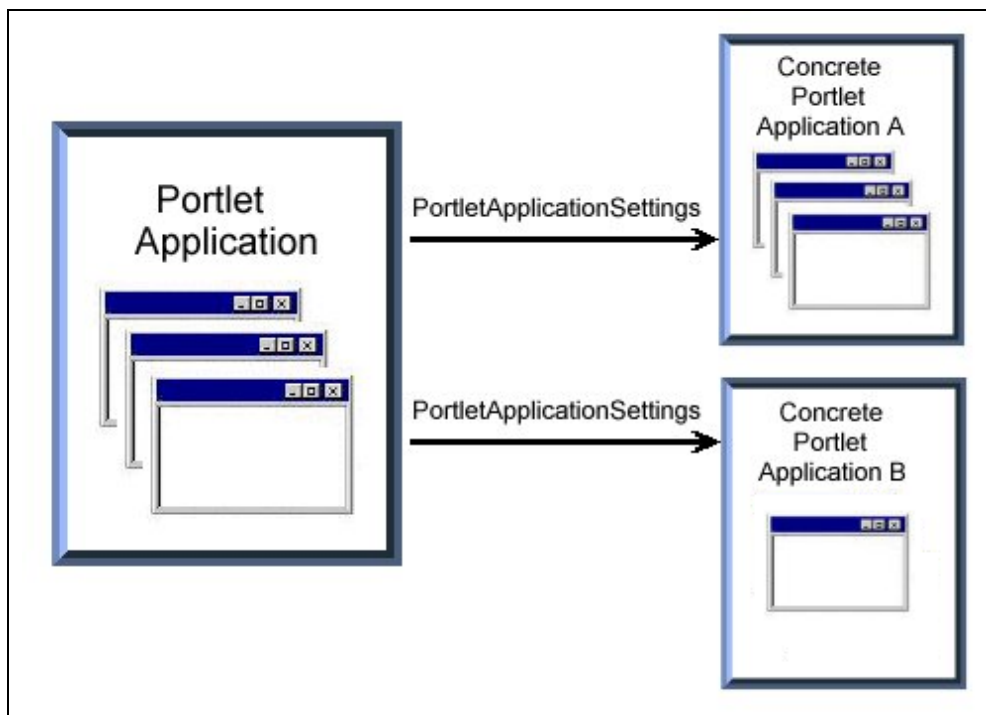


Figure 3: Portlet application

A concrete portlet application contains at least one concrete portlet from the portlet application, but it is not required to contain all of them.

Portlet applications provide no code on their own but form a logical group of portlets. Beside this more logical gain, portlets of the same portlet application can also exchange messages.

2.5. Portlet modes

Portlet modes allow a portlet to display a different user interface, depending on the task required of the portlet. The following modes are provided by the Portlet API:

- View When a portlet is initially constructed on the portal page for a user, it is displayed in its view mode. This is the portlet's normal mode of operation.

- **Help** If this mode is supported by a portlet, the portlet provides a help page for users to obtain more information about the portlet.
- **Edit** If this mode is supported by a portlet, the portlet provides a page for users to customize the portlet for their own needs. For example, a portlet can provide a page for users to specify their location for obtaining local weather and events.
- **Configure** If this mode is supported by a portlet, the portlet provides a page for portal administrators to configure a portlet for a user or group of users.

The Portlet API provides methods for the portlet to determine the current mode.

2.6. Portlet states

Portlet states allow users to change how the portlet window is displayed within the portal. In a browser, users invoke these states with icons in the title bar in the same way that Windows applications are manipulated. Portlet states are maintained in the `PortletWindow.State` object with a boolean value.

- **Normal** When a portlet is initially constructed on the portal page, it is displayed in its normal state – arranged on the page along with other portlets.
- **Maximized** When a portlet is maximized, it is displayed in the entire body of the portal, replacing the view of other portlets.
- **Minimized** When a portlet is minimized, only the portlet title bar is displayed on the portal page.

2.7. Examples in this document - the Bookmark Portlet

This guide shows functions of the Portlet API by using only one example portlet which is constantly extended during the sections, starting with a very simple empty portlet. At last, a fully functional portlet is created that demonstrates each aspect of the Portlet API. This portlet allows users to manage bookmarks.

[BookmarkPortlet 0](#) shows how to get, set, and remove variables for a concrete portlet. The portlet, however, does not produce any output.

[BookmarkPortlet 1](#) writes HTML output to the portlet's view mode.

[BookmarkPortlet 2](#) implements listeners.

[BookmarkPortlet 3](#) shows how to retrieve parameters from `PortletConfig` and attributes from `PortletData` and `PortletSettings`.

[BookmarkPortlet 4](#) shows how to get localized text from the resource bundle: [bookmark.properties](#) .

[BookmarkPortlet 5](#) shows how to render the portlet's view mode using [bookmarkView.jsp](#) .

See [Compiling and testing portlets](#) for information about using the samples in this document. The complete source for all BookmarkPortlet versions, as well as any supporting resources, are available in `bookmark_samplelets.zip`, which is located in the `/dev` directory where the portal server is installed.

3. Portlet API - Basic elements

This section describes the basic interfaces and methods of the Portlet API. The following figure shows a map of many of the common objects in the Portlet API.

- Portlet
 - extended by → [PortletAdapter](#)
 - service() ¹ → [PortletResponse](#)
 - createURI() → [PortletURI](#)
 - service() ² → [PortletRequest](#)
 - getPortletSettings() → [PortletSettings](#)
 - getPortletApplicationSettings → [PortletApplicationSettings](#)
 - getMode() → [Portlet.Mode](#)
 - getClient() → [Client](#)
 - getData() → [PortletData](#)
 - getWindow() → [PortletWindow](#)
 - getWindowState() → [PortletWindow.State](#)
 - getPortletSession() → [PortletSession](#)
 - getUser() → [User](#)
 - getPortletConfig() → [PortletConfig](#)
 - getContext() → [PortletContext](#)
 - getLog() → [PortletLog](#)
 - getService() → [PortletService](#)

3.1. Portlet

The abstract `Portlet` class is the central abstraction of the Portlet API. All portlets extend this abstract class by extending one of its subclasses, such as `PortletAdapter`, that provide methods helpful for the developer. The portlet container calls the following methods of the abstract portlet during the portlet's life cycle ³:

- `init()` The portlet is constructed, after portal initialization, and then initialized with the `init()` method. The portal always instantiates only a single instance of the portlet, and this instance is shared among all users, the same way a servlet is shared among all users of an application server.

¹ `PortletRequest` and `PortletResponse` are passed by helper methods of `PortletAdapter`, such as `doView()`.

² `PortletRequest` and `PortletResponse` are passed by helper methods of `PortletAdapter`, such as `doView()`.

³ This represents the methods called on all portlets. Other methods are called for portlets that implement [listeners](#). See the [Complete Lifecycle](#) for more information.

- `initConcrete()` After constructing the portlet and before the portlet is accessed for the first time, the concrete portlet is initialized with the `PortletSettings`.
- `service()` The portal calls the `service()` method when the portlet is required to render its content. During the life cycle of the portlet, the `service()` method is typically called many times. For each portlet on the page, the `service()` method is not called in a guaranteed order and may even be called in a different order for each request.
- `destroyConcrete()` The concrete portlet is taken out of service with the `destroyConcrete()` method. This can happen when an administrator deletes a concrete portlet during runtime of the portal server.
- `destroy()` When the portal is terminating, portlets are taken out of service, then destroyed with the `destroy()` method. Finally the portlet is garbage collected and finalized.

3.1.1. `getLastModified()`

The portlet container provides a built-in caching mechanism to gain better performance. The abstract `Portlet` provides the `getLastModified()` method, which is called by every request to enable the portlet to inform the container when the cache entry for the portlet should be invalidated and therefore the portlet's content should be refreshed. The `getLastModified()` method returns the last time the content of the portlet changed, in milliseconds, between the current time and midnight, January 1, 1970 UTC (`java.lang.System.currentTimeMillis()`) or a negative value if it is unknown. For an example of how this method is used, see [Portlet Caching](#) .

3.1.2. `PortletAdapter`

The `PortletAdapter` provides a default implementation for the abstract `Portlet` class. It is recommended to not implement the abstract `Portlet` directly. Rather, a portlet should derive from the `PortletAdapter` or any other derived class because changes in the abstract `Portlet` class are then mostly likely to be caught by the default implementation rather than breaking your portlet implementation.

Additionally, the `PortletAdapter` enables a portlet to store variables with the concrete portlet. Concrete portlet variables differ from Java instance variables because they are bound to the portlet class or non-concrete portlet. The `PortletAdapter` provides the methods `setVariable()`, `getVariable()` and `removeVariable()` to work with concrete portlet variables.

3.1.3. Example

The following example shows a basic portlet that extends the `PortletAdapter` and stores some URLs as portlet variables. At this stage of development, the portlet only reads out the variables in the `doView` method but does not provide any output. As a result, an empty portlet window is displayed in the portal.

BookmarkPortlet.java version 0

```
package com.mycompany.portlets.bookmark;

import org.apache.jetspeed.portlet.*;
import java.io.IOException;

public class BookmarkPortlet extends PortletAdapter
{
    public void init (PortletConfig config) throws UnavailableException
    {
        super.init(config);

        // do initialization
    }

    public void destroy (PortletConfig config)
    {
        // undo initialization
    }

    public void initConcrete (PortletSettings settings) throws UnavailableException
    {
        try
        {
            this.setVariable("url.1", "http://www.google.com");
            this.setVariable("url.2", "http://de.my.yahoo.com");
        }
        catch (AccessDeniedException e)
        {
            throw new UnavailableException("problem with setting variables" + e);
        }
    }

    public void destroyConcrete (PortletSettings settings)
    {
        try
        {
            this.removeVariable("url.1");
            this.removeVariable("url.2");
        }
        catch (AccessDeniedException e)
        {
            // do nothing
        }
    }

    public void doView (PortletRequest request,
                        PortletResponse response)
        throws PortletException, IOException
    {
        String url1 = (String) this.getVariable("url.1");
        String url2 = (String) this.getVariable("url.2");
    }
}
```

web.xml version 0

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app 2.2.dtd">
<web-app id="WebApp 1">
    <display-name>bookmark0</display-name>

    <servlet id="Servlet_1">
        <servlet-name>Bookmark0</servlet-name>
        <servlet-class>com.mycompany.portlets.bookmark.BookmarkPortlet</servlet-class>
```



```

</servlet>

<servlet-mapping id="ServletMapping 1">
  <servlet-name>Bookmark0</servlet-name>
  <url-pattern>/Bookmark0/*</url-pattern>
</servlet-mapping>
</web-app>

```

portlet.xml version 0

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN" "portlet_1.1.dtd">
<portlet-app-def>
  <portlet-app uid="com.mycompany.portlets.bookmark.1234" major-version="1" minor-version="0">
    <portlet-app-name>Bookmark Application0</portlet-app-name>
    <portlet id="Portlet 1" href="WEB-INF/web.xml#Servlet 1">
      <portlet-name>Bookmark Portlet0</portlet-name>
      <cache>
        <expires>0</expires>
        <shared>NO</shared>
      </cache>
      <allows>
        <maximized/>
        <minimized/>
      </allows>
      <supports>
        <markup name="html">
          <view output="fragment"/>
        </markup>
      </supports>
    </portlet>
  </portlet-app>

  <concrete-portlet-app uid="com.mycompany.portlets.bookmark.1234.1">
    <portlet-app-name>ConcreteSamplets_Bookmark0</portlet-app-name>
    <context-param>
      <param-name>Webmaster</param-name>
      <param-value>huge@system.com</param-value>
    </context-param>
    <concrete-portlet href="#Portlet_1">
      <portlet-name>Bookmark Portlet0</portlet-name>
      <default-locale>en</default-locale>
      <language locale="en">
        <title>My Bookmarks0</title>
        <title-short>Bookmarks0</title-short>
        <description>Portlet showing your personalized bookmarks</description>
        <keywords>bookmarks</keywords>
      </language>
    </concrete-portlet>
  </concrete-portlet-app>
</portlet-app-def>

```

3.2. Core Objects

3.2.1. PortletRequest

The PortletRequest object is passed to the portlet through the [login\(\)](#), [beginPage\(\)](#), [endPage\(\)](#), and [service\(\)](#) methods, providing the portlet with request-specific data and the opportunity to access the following information.

- **Attributes**
Attributes are name/value pairs associated with a request. Attributes are available only for the scope of the request. The portlet can get, set, and remove attributes during one request.

- **Parameters**
Parameters are name/value pairs sent by the client in the URI query string as part of a request. Often the parameters are posted from a form. Parameters are available for the scope of a specific request. The portlet can get but not set parameters from a request.
- **Client**
The [Client](#) object encapsulates all information about the user agent of a specific request. Information from the Client object includes the manufacturer of the user agent or the type of markup that the client supports.
- **User data**
The [PortletData](#) object represents data for a concrete portlet instance that is saved to persistent store. For example, a user can set a portlet e-mail application to check for new mail every 30 minutes. This preference is stored for the instance in the PortletData object.
- **Session**
The [PortletSession](#) object represents user-specific, transient data for more than one request. In contrast with the request, which does not retain data after the request is completely processed, session attributes can be remembered/saved over more than one request.
- **Portlet settings**
The [PortletSettings](#) object represents the configuration for a concrete portlet that is saved to persistent store. For example, an administrator can set to which host and port a Stock portlet should connect to get live data. This preference is stored for the concrete portlet in the PortletSettings object.
- **Mode**
[Portlet.Mode](#) provides the current or previous mode of the portlet.
- **PortletWindow**
The [PortletWindow](#) object represents the state of the current portlet window. The portlet can access this object to determine if the portlet is currently maximized, minimized, or rendered in its normal view.
- **Portlet.ModeModifier**
The [Portlet.ModeModifier](#) object can be used in a [PortletAction](#) to set the portlet mode to its current, previous, or requested mode before the portlet is rendered. For example, a portlet in edit mode could process a user action and return the portlet to edit mode for more input before returning to view mode.

3.2.1.1. *Client*

The Client object encapsulates request-dependent information about the user agent of a specific request. The Client is extracted from the [PortletRequest](#) using the [getClient\(\)](#) method. The following information can be obtained from the Client:

- User agent

The portlet can get the String sent by the user agent to identify itself to the portal.

- Markup name
The portlet can get the String that indicates the markup language that the client supports, for example, “wml”.
- MIME type
The portlet can get the String that indicates the MIME types supported by the client (for example: “text/vnd.wap.wml”). If the portlet supports multiple types of devices, it should get the markup name rather than the MIME type. The following table shows MIME types and their corresponding markup types.

MIME types	Markup types
text/html	html
text/vnd.wap.wml	wml
text/html	chtml

- Capabilities
The Capability object contains more detailed information than the markup type about what the client can support, such as the level of HTML, Javascript, or WML tables.

3.2.2. PortletResponse

The PortletResponse encapsulates information to be returned from the server to the client. PortletResponse is passed via the beginPage(), endPage() and service() method and is used by the portlet to return portlet content using a Java PrintWriter. The response also includes methods for creating the PortletURI object or qualifying portlet markup with the portlet’s namespace.

Use one of the following methods to create the PortletURI:

- createURI() - Creates a PortletURI object pointing to the calling portlet with the current mode
- createURI(PortletWindow.State *state*) - Creates a PortletURI object pointing to the calling portlet with the current mode and given portlet window state.
- createReturnURI() - Creates a portlet URI pointing at the caller of the portlet. For example, createReturnURI() can be used to create a back button in an edit mode.

Each portlet runs in its own unique namespace. encodeNamespace() is used by portlets to bring attributes in the portlet’s output to avoid name clashes with other portlets. Attributes can include parameter names, global variables, or javascript function names.

3.2.2.1. *PortletURI*

The `PortletURI` object contains a URI pointing to the `Portlet` instance and can be further extended by adding portlet-specific parameters and by attaching actions.

Actions are portlet-specific activities that need to be performed as result of the incoming request, but before the `service()` method of the portlet is called. For example, when a user is entering data in the portlet's edit mode and clicks a "Save" button, the portlet needs to process the posted data before the next markup is generated. This can be achieved by adding a "Save" action to the URI that represents the "Save" button.

The complete URI can be converted to a string which is ready for embedding into markup.

3.2.3. **PortletSession**

The `PortletSession` holds user-specific, transient data for the concrete portlet instance of the portlet, creating a portlet user instance. Concrete portlet instances differ from each other only by the data stored in their `PortletData`. Portlet user instances differ from each other only by the transient data stored in their `PortletSession`. Any persistent data must be stored using [PortletData](#). Information stored in a portlet's instance variables is shared between all concrete portlet instances and even between all concrete portlets – with read and write access. Make sure you do not use instance attributes for user-specific data.

On the other hand, you have to be cautious about what the portlet adds to the session, especially if the portlet ever runs in a cluster environment where the session is being serialized to a shared database. Everything that is stored in the session must be serializable, too.

Like the `HttpSession`, a `PortletSession` is not available on an anonymous page. However, in cases where an administrator places a portlet on an unauthenticated page, such as the Welcome page shipped with WebSphere Portal, the portlet should provide code to handle it as well as message output that would be useful for basic users of the portal.

During login, a `PortletSession` is automatically created for each portlet on a page. To get a `PortletSession`, the `getPortletSession()` method (available from the `PortletRequest`) has to be used. The method returns the current session or, if there is no current session and the given parameter "create" is `true`, it creates one and returns it.

3.2.4. **Example**

The following example, from `BookmarkPortlet` version 1, displays the stored variables as bookmark links. The `PortletResponse`'s `writer` is used to write HTML output for the portlet's view mode. In addition to the bookmark functionality, a counter is implemented and stored in the `PortletSession`. The counter can be hidden by clicking

a link that contains a parameter. This is implemented with the help of a PortletURI object. The corresponding xml files (web.xml and portlet.xml) did not have to be changed as the configuration did not change in comparison to version 0 (except for the version number).

BookmarkPortlet.java version 1

```

package com.mycompany.portlets.bookmark;

...

public class BookmarkPortlet extends PortletAdapter
{
    private static final String COUNTER = "counter";
    private static final String HIDE    = "hide";

    private static final String URL_PREFIX = "url.";
    private static final String NAME_PREFIX = "name.";
    private static final String URL_COUNT = "count";

    public void initConcrete (PortletSettings settings) throws UnavailableException
    {
        ...

        this.setVariable(URL_PREFIX + "1" , "http://www.google.com");
        this.setVariable(NAME_PREFIX + "1" , "Google");
        this.setVariable(URL_PREFIX + "2" , "http://de.my.yahoo.com");
        this.setVariable(NAME_PREFIX + "2" , "Yahoo");
        this.setVariable(URL_COUNT      , "2");

        ...
    }

    public void destroyConcrete (PortletSettings settings)
    {
        ...

        String count = (String) this.getVariable(URL_COUNT);
        if (count != null)
        {
            for (int i=1; i <= Integer.parseInt(count); i++)
            {
                this.removeVariable(URL_PREFIX + i );
                this.removeVariable(NAME_PREFIX + i );
            }
        }

        ...
    }

    public void doView (PortletRequest request,
                       PortletResponse response)
        throws PortletException, IOException
    {
        response.getWriter().println("<p><strong>Predefined bookmarks:</strong></p> ");

        // get the bookmarks
        String count = (String) this.getVariable(URL_COUNT);
        if (count != null)
        {
            for (int i=1; i <= Integer.parseInt(count); i++)
            {
                String url = (String) this.getVariable(URL_PREFIX + i);
                String name = (String) this.getVariable(NAME_PREFIX + i);

                // make them to links
                response.getWriter().println("<p><a href='" + url
                    + "' target=bookmarkwindow>" + name + "</a></p>");
            }
        }

        // build the show/hide counter linke
        PortletURI uri = response.createURI();
        String msg = null;

        String bool = request.getParameter(HIDE);
        Boolean hide = null;
    }
}

```

```

if (bool == null) hide = new Boolean(false);
else
    hide = new Boolean(bool);

// get the counter value
Integer counter = (Integer) request.getSession().getAttribute(COUNTER);
if (counter == null)
{
    counter = new Integer(0);
}

// write out the counter if necessary
if (hide.booleanValue())
{
    hide = new Boolean(false);
    msg = "click to SHOW counter";
}
else
{
    hide = new Boolean(true);
    msg = "click to HIDE counter";

    response.getWriter().println("<p>Reload counter (per session): <strong>" +
        counter.toString() + "</strong>");
}

// increase the counter and put into session
counter = new Integer(counter.intValue()+1);
request.getSession().setAttribute(COUNTER, counter);

// write out the show/hide link
uri.addParameter(HIDE, hide.toString());
response.getWriter().println("<p><a href='" + uri.toString() + "'>" + msg + "</a><p>");
}
}

```

See [Compiling and testing portlets](#) for information about using this or other samples in this document. The source for all BookmarkPortlet versions, as well as any supporting resources, are available in `bookmark_samplets.zip`.

3.3. Listeners

The Portlet API provides listeners, which can add more functionality for a portlet. To enable the listener's functionality, one of the following interfaces has to be implemented by the portlet.

- [PortletSessionListener](#)
- [PortletPageListener](#)
- [PortletTitleListener](#)

The following additional listeners are described in other sections of this document:

- [ActionListener](#)
- [WindowListener](#)
- [MessageListener](#)
- [PortletSettingsAtributesListener](#)
- [PortletApplicationSettingsAtributesListener](#)

3.3.1. PortletSessionListener

In addition to the concrete portlet instance, which exists for each portlet occurrence on a page, a portlet may have an even finer granularity. The `PortletSessionListener` allows a portlet to recognize the lifecycle of a user portlet instance:

- `login()` After a user logs in to a portal, each portlet creates a session for the user. The combination of the concrete portlet instance and the user session creates the *user portlet instance*. The start of a user instance is signaled by the portal calling the `login()` method at the portlet. This method allows the portlet to initialize the user's session instance of the portlet, for example, to store attributes in the session.
- `logout()` When the user ends the session with the portal or the session times out, the portal server calls the `logout()` method to inform the portlet that the user's session instance is terminating. The portlet should then clean up any resources for the portlet user instance.

3.3.2. PortletPageListener

A portlet has no control or awareness of the order in which the output from all of the portlets on the page is written. The `PortletPageListener` allows a portlet to insert markup at the beginning and ending of a page.

- `beginPage()` At the beginning of each page and before any `service()` method of any portlet on the page is called, the `beginPage()` method is called for each portlet residing on the page. Like the `service()` method, the `beginPage()` method for each portlet on the page is not called in a guaranteed order and can even be called in a different order for each request. The output of `beginPage()` is inserted into the page aggregation area prior to the rendering of portlets on the page.

This method allows a portlet to output Javascript that is visible to all portlets' `service()` methods or even to set cookies or headers. This method cannot be used to add output to the `<HEAD>` section of the portal page or redirect the portal page.
- `endPage()` At the end of each page, and after all `service()` methods of all portlets on the page are called, the `endPage()` method is called for each portlet residing on the page. Like the `service()` method, the `endPage()` method is not called in a guaranteed order and can even be called in a different order for each request. For example, the portlet can insert Javascript to the end of the page that needs to occur after all other portlets on the page have been written.

These methods are called if the `<wps:pageRender/>` tag includes the attributes `includeBeginPage="yes"` and `includeEndPage="yes"`. The `<wps:pageRender/>` tag can be found in the file `Home.jsp` in the `/screens` directory.

3.3.3. PortletTitleListener

The PortletTitleListener interface is used to allow the portlet title, as it is displayed in the title bar, to be changed based on a condition (for example, the type of device used to access the portal) or user input (for example, a preference that the user sets on the edit page). If the PortletTitleListener is not implemented, the portlet will display the title specified on the <title> element (under <language>) of the portlet deployment descriptor.

3.3.4. Complete Lifecycle

When implementing all listener interfaces that are linked with a portlet's lifecycle, the methods are executed:

- `init()` The non-concrete portlet is initialized with `PortletConfig`.
- `initConcrete()` The concrete portlet is initialized with `PortletSettings`.
- `login()` The user portlet instance is initialized with `PortletSession`.
- `beginPage()` The portlet can render output at the beginning of each page for each request.
- `service()` The portlet may render output in the portlet window for each request.
- `endPage()` The portlet may render output at the end of each page for each request.
- `logout()` The user portlet instance is destroyed.
- `destroyConcrete()` The concrete portlet is destroyed.
- `destroy()` The non-concrete portlet is destroyed.

3.3.5. Example

The following example, from `BookmarkPortlet` version 2, implements the `PortletSessionListener`, the `PortletPageListener` and the `PortletTitleListener`. Therefore, the portlet is able to initialize the session and set an attribute in the `login` method instead of doing it in `doView()`. The attribute is removed on `logout`. Moreover, it declares Javascript methods in `beginPage()` and sets a dynamic title in `doTitle()`.

BookmarkPortlet.java version 2

```
package com.mycompany.portlets.bookmark;
import org.apache.jetspeed.portlet.*;
```



```

import org.apache.jetspeed.portlet.event.*;
import java.io.IOException;
import javax.servlet.http.HttpSession;

public class BookmarkPortlet extends PortletAdapter
implements PortletSessionListener, PortletPageListener, PortletTitleListener
{
    private static final String COUNTER = "counter";
    private static final String HIDE = "hide";

    private static final String URL_PREFIX = "url.";
    private static final String NAME_PREFIX = "name.";
    private static final String URL_COUNT = "count";

    // PortletSessionListener interface methods
    public void login(PortletRequest request)
    {
        request.getSession().setAttribute(COUNTER, new Integer(0));
    }

    public void logout(PortletSession session)
    {
        session.removeAttribute(COUNTER);
    }

    // PortletPageListener interface methods
    public void beginPage (PortletRequest request,
        PortletResponse response)
    throws PortletException, IOException
    {
        // write stuff into the header of the page (e.g. JavaScript)
        response.getWriter().println("<!--insert your JavaScript stuff here:\n");
        response.getWriter().println("    function " +
            response.encodeNamespace("myMethodName") + "() {}");
        response.getWriter().println("\n-->");
    }

    public void endPage (PortletRequest request,
        PortletResponse response)
    throws PortletException, IOException
    {
    }

    // PortletTitleListener interface methods
    public void doTitle (PortletRequest request,
        PortletResponse response)
    throws PortletException, IOException
    {
        response.getWriter().print("My Bookmarks2");

        String bool = request.getParameter(HIDE);
        if ((bool == null) || !Boolean.valueOf(bool).booleanValue())
        {
            response.getWriter().print(" showing counter");
        }
    }

    public void initConcrete (PortletSettings settings) throws UnavailableException
    {
        try
        {
            this.setVariable(URL_PREFIX + "1" , "http://www.google.com");
            this.setVariable(NAME_PREFIX + "1" , "Google");
            this.setVariable(URL_PREFIX + "2" , "http://de.my.yahoo.com");
            this.setVariable(NAME_PREFIX + "2" , "Yahoo");
            this.setVariable(URL_COUNT , "2");
        }
        catch (AccessDeniedException e)
        {
            throw new UnavailableException("problem with setting variables" + e);
        }
    }

    public void destroyConcrete (PortletSettings settings)
    {
        try
        {
            String count = (String) this.getVariable(URL_COUNT);

```

```

        if (count != null)
        {
            for (int i=1; i <= Integer.parseInt(count); i++)
            {
                this.removeVariable(URL_PREFIX + i );
                this.removeVariable(NAME_PREFIX + i );
            }
        }
    }
    catch (AccessDeniedException e)
    {
        // do nothing
    }
}

public void doView (PortletRequest request,
                   PortletResponse response)
throws PortletException, IOException
{
    response.getWriter().println("<p><strong>Predefined bookmarks:</strong></p> ");

    // get the bookmarks
    String count = (String) this.getVariable(URL_COUNT);
    if (count != null)
    {
        for (int i=1; i <= Integer.parseInt(count); i++)
        {
            String url = (String) this.getVariable(URL_PREFIX + i);
            String name = (String) this.getVariable(NAME_PREFIX + i);

            // make them to links
            response.getWriter().println("<p><a href='" + url
                + "' target=bookmarkwindow>" + name + "</a></p>");
        }

        // build the show/hide counter links (the counter will only be hidden
        // on a request basis)
        PortletURI uri = response.createURI();
        String msg = null;

        String bool = request.getParameter(HIDE);
        Boolean hide = null;

        if (bool == null) hide = new Boolean(false);
        else hide = new Boolean(bool);

        // get the counter value - it was set in the login method
        Integer counter = (Integer) request.getSession().getAttribute(COUNTER);

        // write out the counter if necessary
        if (hide.booleanValue())
        {
            hide = new Boolean(false);
            msg = "click to SHOW counter";
        }
        else
        {
            hide = new Boolean(true);
            msg = "click to HIDE counter";

            response.getWriter().println("<p>Reload counter (per session): <strong>" +
                counter.toString() + "</strong></p>");
        }

        // increase the counter and put into session
        counter = new Integer(counter.intValue()+1);
        request.getSession().setAttribute(COUNTER, counter);

        // write out the show/hide link (the counter will only be hidden
        // on a request basis)
        uri.addParameter(HIDE, hide.toString());
        response.getWriter().println("<p><a href='" + uri.toString() + "'>" + msg + "</a></p>");
    }
}
}

```

3.4. Configuration Objects

3.4.1. PortletConfig

The PortletConfig provides the non-concrete portlet with its initial configuration. The configuration holds information about the portlet class. This information is valid for every concrete portlet derived from the portlet.

A portlet's configuration is initially read from its associated servlet from the web deployment descriptor. This information is set by the portlet developer. The configuration is read-only and cannot be changed by the portlet.

The PortletConfig is passed to the portlet in the init() method of the abstract Portlet and is used to access portlet-specific configuration parameters using getInitParameters(). PortletConfig parameters are name/value pairs available for the complete life cycle of the non-concrete portlet. Non-concrete portlet's parameters are defined by the <init-param> tag in the associated servlet in the Web deployment descriptor.

3.4.2. PortletSettings

The PortletSettings provide the concrete portlet with its dynamic configuration. The configuration holds information about the concrete portlet. This information is valid for every concrete portlet instance of the concrete portlet.

A concrete portlet's configuration is initially read from the portlet deployment descriptor. The configuration is read only and can be written by the portlet only when the portlet is in [configure mode](#). This information is normally maintained by the portal administrator and may be changed while the portal server is running. The portlet can get, set, and remove attributes during one request. To commit the changes, the store() method has to be called.

The PortletSettings object can be accessed with the getPortletSettings() method, available from the PortletRequest. Often, it is used to access portlet-specific configuration parameters using getAttribute(). Attributes are name/value pairs available for the complete life cycle of a concrete portlet. Concrete portlet attributes are defined by the <config-param> tag in the portlet deployment descriptor.

3.4.3. PortletApplicationSettings

The PortletApplicationSettings object provides the concrete portlet application with its dynamic configuration. The configuration holds information about the portlet application that is shared across all concrete portlets included in the application.

A concrete portlet application's configuration is initially read from the portlet deployment descriptor. The configuration is read only and can be written by the portlet only when the portlet is in [configure mode](#). This information is normally maintained by the portal administrator and may be changed while the portal server is running. A portlet in the application can get, set, and remove attributes during one request. To commit the changes, the `store()` method has to be called.

The `PortletApplicationSettings` can be accessed with the `getApplicationSettings()` method, available from the `PortletSettings` object. It is used to access portlet-specific configuration parameters using `getAttribute()`. Attributes are name/value pairs available for the duration of a concrete portlet application. Concrete portlet application attributes are defined by the `<context-param>` tag in the portlet deployment descriptor.

3.4.4. PortletData

The `PortletData` holds data for the *concrete portlet instance*. For each occurrence on a page there is a concrete portlet instance. The `PortletData` contains **persistent** information about the concrete portlet instance while the `PortletSession` contains only the **transient** data of the user portlet instance.

There is one concrete portlet instance for each occurrence of a portlet on a page. A page can be owned by either a single user (personal page) or by a single group of users (group page). `PortletData` contains user-specific data on a personal page and group-specific data on a group page.

The `PortletData` stores attributes as name/value pairs. The portlet can get, set, and remove attributes during one request. To commit the changes, the `store()` method has to be called. The data is read only and can be written by the portlet only when the portlet is in [edit mode](#).

3.4.5. Example

Version 3 of `BookmarkPortlet` stores the predefined bookmarks in the portlet's `PortletSettings`, which are set as `config-param` entries in the `portlet.xml` and can be accessed via the `PortletRequest`. User-defined bookmarks are read from the `PortletData`, but as they are not set, there is nothing to display yet. Moreover, an image is set as an `init-param` in the `web.xml`. The image name is retrieved via the `PortletConfig` and encoded before turned into a link.

BookmarkPortlet.java version 3

```
package com.mycompany.portlets.bookmark;

import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlet.event.*;
import java.io.IOException;

import java.util.Enumeration;

public class BookmarkPortlet extends PortletAdapter
implements PortletTitleListener
{
```

```

private static final String URL_PREFIX = "url.";
private static final String NAME_PREFIX = "name.";
private static final String URL_COUNT = "count";

private static final String IMAGE = "image.name";

// PortletTitleListener interface methods
public void doTitle (PortletRequest request,
                    PortletResponse response)
throws PortletException, IOException
{
    response.getWriter().print(
        request.getPortletSettings().getTitle(request.getLocale(),
                                             request.getClient() )
        );

    String count = (String) request.getData().getAttribute(URL_COUNT);
    if (count != null)
    {
        response.getWriter().print(" - User defined bookmarks: " + count);
    }
}

public void initConcrete (PortletSettings settings) throws UnavailableException
{
    // predefined urls are now in the settings
}

public void destroyConcrete (PortletSettings settings)
{
    // predefined urls are now in the settings
}

public void doView (PortletRequest request,
                   PortletResponse response)
throws PortletException, IOException
{
    response.getWriter().println("<table><tr><td>");
    response.getWriter().println("<strong>Predefined bookmarks:</strong> ");

    // get the bookmarks
    PortletSettings settings = request.getPortletSettings();

    String count = settings.getAttribute(URL_COUNT);
    if (count != null)
    {
        for (int i=1; i <= Integer.parseInt(count); i++)
        {
            String url = (String) settings.getAttribute(URL_PREFIX + i);
            String name = (String) settings.getAttribute(NAME_PREFIX + i);

            // make them to links
            response.getWriter().println("<p><a href='" + url
                + "' target=bookmarkwindow>" + name + "</a></p>");
        }
    }

    response.getWriter().println("<p><strong>Userdefined bookmarks:</strong></p> ");

    // show user defined bookmarks from PortletData
    PortletData data = request.getData();

    count = (String) data.getAttribute(URL_COUNT);
    if (count != null)
    {
        for (int i=1; i <= Integer.parseInt(count); i++)
        {
            String url = (String) data.getAttribute(URL_PREFIX + i);
            String name = (String) data.getAttribute(NAME_PREFIX + i);

            // make them to links
            response.getWriter().println("<p><a href='" + url
                + "' target=bookmarkwindow>" + name + "</a></p>");
        }
    }

    response.getWriter().println("</td><td>");

```

```

String imageName = this.getPortletConfig().getInitParameter (IMAGE);
response.getWriter().println(
    "<img src='" + response.encodeURL("/images/"+imageName) + "'>");

response.getWriter().println("</td></tr></table>");
}
}
}

```

web.xml version 3

```

<?xml version="1.0" encoding="UTF-8"?>
...
<servlet-class>com.mycompany.portlets.bookmark.BookmarkPortlet</servlet-class>
  <init-param>
    <param-value>bookmarks.gif</param-value>
  </init-param>
...

```

portlet.xml version 3

```

<?xml version="1.0" encoding="UTF-8"?>
...
<concrete-portlet-app uid="com.mycompany.portlets.bookmark.1234.1">
  <portlet-app-name>ConcreteSamplets Bookmark3</portlet-app-name>
...
  <concrete-portlet href="#Portlet_1">
    <portlet-name>Bookmark Portlet3</portlet-name>
    <default-locale>en</default-locale>
    <language locale="en">
      <title>My Bookmarks3</title>
      <title-short>Bookmarks3</title-short>
      <description>Portlet showing your personalized bookmarks</description>
      <keywords>bookmarks</keywords>
    </language>
    <config-param>
      <param-name>url.1</param-name>
      <param-value>http://www.google.com</param-value>
    </config-param>
    <config-param>
      <param-name>name.1</param-name>
      <param-value>Google</param-value>
    </config-param>
    <config-param>
      <param-name>url.2</param-name>
      <param-value>http://de.my.yahoo.com</param-value>
    </config-param>
    <config-param>
      <param-name>name.2</param-name>
      <param-value>Yahoo</param-value>
    </config-param>
    <config-param>
      <param-name>count</param-name>
      <param-value>2</param-value>
    </config-param>
  </concrete-portlet>
</concrete-portlet-app>

```

3.5. Miscellaneous Objects

3.5.1. PortletContext

The PortletContext interface defines a portlet's view of the portlet container within which each portlet is running. The PortletContext also allows a portlet to access

resources available to it. For example, using the context, a portlet can access the portlet log, access context parameters common to all portlets within the portlet application, obtain URL references to resources, or access [portlet services](#) .

The most important information related to the PortletContext is described in detail below:

- **InitParameters**
Parameters are name/value pairs available to all portlets within the web application. These are defined in the web deployment descriptor under the <context-param> element. For example, if a group of portlets share a context parameter called “Webmaster” that contains the portal site’s administrator email, each portlet could get that value and provide a “mailto” link in their help.
- **Attributes**
Attributes are name/value pairs available to all portlets within the web application.
Hint: *Attributes of the context are stored on a single machine and are not distributed in a cluster.*
- **Localized text**
The getText() method is used by the portlet to access resource bundles within a given locale.
- **Resources**
It is through the PortletContext that a portlet can load or include resources located in the portlet’s application scope. Available methods are include() and getResourceAsStream(). The include() method is typically used to invoke JSPs for output.
- **Messaging**
Through messaging, it is possible to communicate between portlets and share data or send notifications. A message is sent by using the send() method. For more information, see [Portlet events](#).
- **PortletServices**
PortletServices allow portlets to use pluggable services via dynamic discovery. See [Portlet Services](#) for more information.
- **Log**
The [PortletLog](#) provides the portlet with the ability to log informational, warning, or error messages.

3.5.1.1. PortletLog

Portlets can write message and trace information to log files, which are maintained in the `wps_root/log/` directory. The log is maintained by the portlet container. Whether logging is enabled or not is at the discretion of the portlet container. The log files help the portal administrator investigate portlet errors and special conditions and help the portlet developer test and debug portlets. The Portlet API provides the

PortletLog class, which has methods to write message and trace information to the logs.

debug()

Writes trace information to `wps_[timestamp].log`.

info()

Writes informational messages to `wps_[timestamp].log`.

error()

Writes error messages to `wps_[timestamp].log`.

warn()

Writes warning messages to `wps_[timestamp].log`.

[timestamp] has the following format:

```
year.month.date-hour.minute.second
```

For example, `wps_2002.03.08-14.00.00.log` was written on March 8, 2002, at 2:00 pm.

If you access the portlet log multiple times in a method it is good idea to assign the log reference to a variable, for example:

```
private PortletLog myLogRef = getPortletLog();
```

Since logging operations are expensive, PortletLog provides methods to determine if logging is enabled for a given level. Your portlets should write to the log of a given level only if the log is tracking messages of that level. For example:

```
if( getPortletLog().isDebugEnabled() )
{
    myLogRef.debug("Warning, Portlet Resources are low!");
}
```

3.5.2. PortletWindow

The PortletWindow represents the window that encloses a portlet. For example, on an HTML page, the portlet window can typically be rendered as a table cell. The portlet window can send events on manipulation of its various window controls, like when the user clicks minimize or close. The portlet, in turn, can interrogate the window about its current visibility [state](#). For example, a portlet may render its content differently depending on whether its window is maximized or not. The PortletWindow is available using the `getWindow()` method of the PortletRequest object.

3.5.3. User

The User class represents the users of the portal. The User class contains methods for accessing attributes that make up the user profile, such as the user's full name or the username. The User class abstracts the underlying physical implementation of the one or more data stores which actually hold the user information.

In WebSphere Portal, the User class is part of the `com.ibm.wps.puma` package. This class and several others represent the portal server's API to the user subsystem in Member Services. To obtain the Javadoc for this API, locate the document titled "Javadoc for the WebSphere Portal User and Group objects" on the software support site, <http://www.ibm.com/software/support>.

Note: The `getUser()` method is located available from the `PortletRequest` and `PortletSession` objects. The `portletsession.getUser()` method is deprecated and, in subsequent releases, will only return null.

3.5.4. Example

In version 4 of the BookmarkPortlet, the `getText()` method is used to retrieve the localized value of the `PreDefinedBookmarks` and the `UserDefinedBookmarks` parameters, which are set in a properties file. This value is added to the `PrintWriter` to be rendered in the portlet.

Note: This sample follows standard Java conventions for resource bundles. The `getText()` method retrieves the resource bundle as "nls.bookmark". The resource bundle is packaged in the `/WEB-INF/classes/nls` directory of the WAR file as `bookmark.properties`. For more information about resource bundles, see *Accessing Resources in a Location-Independent Manner* in the JDK documentation.

BookmarkPortlet.java version 4

```
package com.mycompany.portlets.bookmark;

import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlet.event.*;
import java.io.IOException;
import java.util.Enumeration;

public class BookmarkPortlet extends PortletAdapter
implements PortletTitleListener
{
    private static final String URL_PREFIX = "url.";
    private static final String NAME_PREFIX = "name.";
    private static final String URL_COUNT = "count";

    private static final String IMAGE = "image.name";

    ...

    public void doView (PortletRequest request,
                       PortletResponse response)
    throws PortletException, IOException
    {
        if (getPortletLog().isDebugEnabled())
            getPortletLog().debug("BookmarkPortlet doView()");
    }
}
```

```

response.getWriter().println("<table><tr><td>");

PortletContext context = getPortletConfig().getContext();

String localizedText = context.getText("nls.bookmark",
                                     "PreDefinedBookmarks",
                                     request.getLocale());

response.getWriter().println("<strong>" + localizedText + "</strong> ");

// get the bookmarks
PortletSettings settings = request.getPortletSettings();

...

localizedText = context.getText("nls.bookmark",
                               "UserDefinedBookmarks",
                               request.getLocale());

response.getWriter().println("<p><strong>" +
                             localizedText +
                             "</strong></p> ");

PortletData data = request.getData();

...
}
}

```

```

bookmark.properties

# localized text of the bookmark portlet
PreDefinedBookmarks = Predefined bookmarks:
UserDefinedBookmarks = User defined bookmarks:
Yahoo = My Yahoo Portal
Google = Google Web Search

```

3.6. The scope of the portlet objects

The following table shows the scope of portlet objects as they correspond to other elements of the Portlet API.

Scope \ Objects	Request	User Portlet Instance	Concrete Portlet Instance	Concrete Portlet	Concrete Portlet Application	Non-concrete Portlet	Non-concrete Portlet Application
PortletRequest	✓	-	-	-	-	-	-
PortletResponse	✓	-	-	-	-	-	-
PortletSession	☑	✓	-	-	-	-	-
PortletData	☑	☑	✓	-	-	-	-
PortletSettings	☑	☑	☑	✓	-	-	-
PortletApplicationSettings	☑	☑	☑	☑	✓	-	-
PortletConfig	☑	☑	☑	☑	☑	✓	-
PortletContext	☑	☑	☑	☑	☑	☑	✓

Legend:

- ✓, means that the *object* is directly related to the *scope*. You can read it like this and replace the cursive words with the appropriate object/scope:

There is one *object* per *scope*.

For example, there is one PortletSession per user portlet instance.

- ☑, means that the *object* is inside of the *scope*, meaning that the *scope* is part of the matching (marked with ✓) scope to this *object*.
- -, means that the *object* is outside of the *scope*.

4. Java Server Pages

4.1. Portlet API Tags

The portlet container provides tags for use in portlet JSPs. To make these tags available in a JSP, the following directive is required at the beginning of the JSP:

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

4.1.1. if

4.1.1.1. Description

Through the attributes of this tag, several conditions can be checked. If the condition is true, the content of the tag is written to the page. Otherwise the content is skipped. More than one condition can be evaluated. All conditions must evaluate to be true for the tag contents to be written. Also, each parameter can contain multiple values separated by commas or semicolons. Only one value in the list needs to be true for the condition to evaluate as true.

4.1.1.2. Parameters

Include at least one of the following attributes:

- `mode = [Portlet.Mode]`
evaluates the current mode of the portlet. For example, `mode="Edit"` is true if the user has placed the portlet in Edit mode.
- `previousMode = [Portlet.Mode]`
evaluates the previous mode of the portlet. For example, `previousMode="View"` is true after a portlet is moved from View to Edit mode.
- `state = [PortletWindow.State]`
evaluates the state of the portlet. For example, `state="Minimized"` is true after the user clicks the minimize icon of the portlet.
- `markup = [string]`
evaluates the markup language supported by the client. For example, `markup="wml"` is true if the client supports WML.
- `mimetype = [string]`
evaluates the MIME type supported by the client. For example, `mimetype="text/html"` is true if the client supports HTML.
- `capability = [Capability]`
evaluates the client's capabilities. For example, `capability="HTML_CSS"` is true if the client supports cascading style sheets.

4.1.1.3. Example

In the following example, the portlet must be in View mode and the portlet state must be either Normal or Maximized for the image to be rendered:

```
<portletAPI:if mode="View" state="Normal,Maximized">  
  
```

```
</portletAPI:if>
```

4.1.2. log

4.1.2.1. Description

Writes a string in the portlet log.

4.1.2.2. Parameters

- `text = [string]` (*required*)
indicates the message or trace string to be recorded in the portlet log.
- `level = [ERROR | WARN | INFO | DEBUG]` (*optional, default: ERROR*)
indicates the level of the message to be logged. DEBUG writes to PortletTraces.log; the other levels write to Portlet.log.

4.1.2.3. Example

```
<portletAPI:log text="This is an error message!" />
```

4.1.3. text

4.1.3.1. Description

Writes a localized string to the output stream. When no resource bundle can be found, the content between the start and end tag is written to the output stream.

4.1.3.2. Parameters

- `bundle = [string]` (*mandatory*)
the resource bundle
- `key = [string]` (*mandatory*)
the key to be found in the resource bundle.

4.1.3.3. Example

In the following example, the value of the `welcome_message` parameter is retrieved from the portlet's `stockportlet.properties` file.

```
<portletAPI:text bundle="nls.stockportlet" key="welcome_message">
Welcome!
</portletAPI:text>
```

4.1.4. dataLoop

4.1.4.1. Description

Loops through all attributes in PortletData of the current concrete portlet instance. Use the [dataAttribute](#) tag to get the attributes between the dataLoop start and end tag.

4.1.4.2. Parameters

- pattern = [string] (*optional*)
a regular expression that defines which attributes should be looped.

4.1.4.3. Example

see [dataAttribute](#)

4.1.5. dataAttribute

4.1.5.1. Description

Returns the value of one or more PortletData attributes. The attribute can be specified by the name parameter. You can also use this tag within a [dataLoop](#) tag to retrieve all attributes or a subset of the attributes from PortletData.

4.1.5.2. Parameters

- name = [string] (*optional*)
indicates the attribute name.

4.1.5.3. Example

The following example loops through the PortletData attributes, returning only the values with names that start with stock.user.

```
Your user settings of the stock portlet:<br>
<portletAPI:dataLoop pattern="stock.user.*">
  <portletAPI:dataAttribute/><br>
</portletAPI:dataLoop>
```

```
One specific user setting of the stock portlet:<br>
<portletAPI:dataAttribute name="stock.user.quotes-count"/><br>
```

4.1.6. settingsLoop

4.1.6.1. Description

Loops through all attributes in PortletSettings of the current concrete portlet. Use the [settingsAttribute](#) tag to get the attributes between start and end tag.

4.1.6.2. Parameters

- `pattern = [string]` (*optional*)
a regular expression that defines which attributes should be looped.

4.1.6.3. Example

see [settingsAttribute](#)

4.1.7. settingsAttribute

4.1.7.1. Description

Returns the value of one or more PortletSettings attributes. The attribute can be specified by the name parameter. You can also use this tag within a [settingsLoop](#) tag to retrieve all attributes or a subset of the attributes from PortletSettings.

4.1.7.2. Parameters

- `name = [string]` (*optional*)
indicates the attribute name.

4.1.7.3. Example

The following example loops through the PortletSettings attributes, returning only the values with names that start with stock.config.

```
Your configuration settings of the stock portlet:<br>
<portletAPI:settingsLoop pattern="stock.config.*.">
  <portletAPI:settingsAttribute/><br>
</portletAPI:settingsLoop>
```

```
One specific configuration setting of the stock portlet:<br>
<portletAPI:settingsAttribute name="stock.config.hostname"/><br>
```

4.1.8. CreateReturnURI

4.1.8.1. Description

Creates a URI that points to the caller of the portlet. You can pass a parameter or action in the URI by including [URIParameter](#) or [URIAction](#) between the CreateReturnURI start and end tags.

4.1.8.2. Example

The following example creates a URI to provide a link that returns the user to the portlet's previous mode or state.

```
<a href="<portletAPI:CreateReturnURI/>">
  <portletAPI:text bundle="nls.my_portlet" key="back_label">
```

```
    Back
  </portletAPI:text>
</a>
```

For examples of passing a parameter or action with this tag, see [URIPParameter](#) and [URIAction](#).

4.1.9. createURI

4.1.9.1. Description

Creates an URI that points to the current portlet with the given parameters. You can pass a parameter or action in the URI by including [URIPParameter](#) or [URIAction](#) between the createURI start and end tags.

4.1.9.2. Parameters

- `state = [PortletWindow.State]` (*optional*) indicates the state of the portlet. For example, `state="Maximize"` creates a URI for a link that maximizes the portlet. If state is not specified, the URI points to the portlet's current state.

4.1.9.3. Example

See [URIPParameter](#) and [URIAction](#)

4.1.10. URIPParameter

4.1.10.1. Description

Adds a parameter to the URI of the createURI and CreateReturnURI tags.

4.1.10.2. Parameters

- `name = [string]` (*mandatory*)
The name of the parameter to add.
- `value = [string]` (*mandatory*)
The value of the parameter to add.

4.1.10.3. Example

```
<portletAPI:createURI state="Maximized">
  <portletAPI:URIPParameter name="showQuote" value="IBM"/>
</portletAPI:createURI>
```

4.1.11. URIAction

4.1.11.1. Description

Adds a default action to the URI of the createURI and createReturnURI tags. The portlet must have a registered action listener to handle the event.

4.1.11.2. Parameters

- name = [string] (*mandatory*)
The name of the parameter to add.

4.1.11.3. Example

```
<portletAPI:createURI>  
  <portletAPI:URIAction name="Add" />  
</portletAPI:createURI>
```

4.1.12. encodeNamespace

4.1.12.1. Description

Maps the given string value into this portlet's namespace. Use this tag for named elements in portlet output (for example, form fields or Javascript variables) to uniquely associate the element with this concrete portlet instance and avoid name clashes with other elements on the portal page or with other portlets on the page.

4.1.12.2. Parameters

- name = [string] (*mandatory*)
The value to be mapped into the portlet's namespace.

4.1.12.3. Example

In the following example, a text field with the name 'email' is encoded to ensure uniqueness on the portal page.

```
<input border="0"  
  type="text"  
  name="<portletAPI:encodeNamespace name='email' />">
```

4.1.13. init

4.1.13.1. Description

Provides the following variables that the JSP can use to access the corresponding objects of the portlet API:

- portletRequest
- portletResponse
- portletConfig

4.1.13.2. Example

After using the `init` tag, the JSP invokes the `encodeNamespace()` method of the `portletResponse`.

```
<portletAPI:init/>
<%=portletResponse.encodeNamespace("test")%>
```

4.1.14. bidi

4.1.14.1. Description

Supports text for bidirectional languages. Bidirectional languages are those which are typically read from right to left except when left-to-right text strings are displayed (for example, URLs, code samples, or directory and file names).

4.1.14.2. Parameters

- `dir = rtl|ltr`
Indicates the normal direction of text in the language.
 - For `dir="rtl"`, the tag content is written only if the client's locale belongs to a bidirectional language. This is the default setting if `dir` is not specified.
 - For `dir="ltr"`, the tag content is written only if the client's locale does not belong to a bidirectional language.
- `locale`
The tag content is written only if the language belongs to the bidirectional languages defined in `LocalizerService.properties`.

4.1.15. encodeURI (deprecated)

Use `encodeURL()` of the `ServletResponse` instead. For example:

```
<%=response.encodeURL("/images/results.gif")%>
```

The output of this method returns a relative URL to the resource, without the protocol, host, and domain part of the path.

4.2. The example using a JSP

BookmarkPortlet.java version 5

```
package com.mycompany.portlets.bookmark;

import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlet.event.*;
import java.io.IOException;
import java.util.Enumeration;
```

```

public class BookmarkPortlet extends PortletAdapter
implements PortletTitleListener
{
    private static final String URL_PREFIX = "url.";
    private static final String NAME_PREFIX = "name.";
    private static final String URL_COUNT = "count";

    private static final String IMAGE = "image.name";
    private static final String VIEW_JSP = "jsp.view";

    // PortletTitleListener interface methods
    public void doTitle (PortletRequest request,
                        PortletResponse response)
    throws PortletException, IOException
    {
    ...
    }

    public void doView (PortletRequest request,
                       PortletResponse response)
    throws PortletException, IOException
    {
        if (getPortletLog().isDebugEnabled())
            getPortletLog().debug("BookmarkPortlet doView()");

        String jspName = getPortletConfig().getInitParameter(VIEW_JSP);

        getPortletConfig().getContext().include(
            "/WEB-INF/jsp/"+jspName, request, response);
    }
}

```

bookmarkView.jsp version 5

```

<!--!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"-->

<%@ page session="false" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<%@ page import="com.mycompany.portlets.bookmark.*" %>
<%@ page import="java.util.*" %>

<portletAPI:log text="bookmarkView.jsp" level="DEBUG"/>

<table class="Portlet" width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
<td>

<table class="Portlet" border="0" cellspacing="0" cellpadding="0">
<tr>
<td>
<strong><portletAPI:text key="PreDefinedBookmarks" bundle="nls.bookmark">
Predefined Bookmarks (fallback):
</portletAPI:text></strong>
</td>
</tr>
<tr>
<td>
<portletAPI:settingsLoop pattern="name.*">
<p><portletAPI:settingsAttribute/></p>
</portletAPI:settingsLoop>
</td>
<td>
<portletAPI:settingsLoop pattern="url.*">
<a href='<portletAPI:settingsAttribute/>' target=bookmarkwindow>click</a>

<portletAPI:if state="Maximized">
-> <portletAPI:settingsAttribute/>
</portletAPI:if>
<BR>
</portletAPI:settingsLoop>
</td>
</tr>
</table>

<!-- Empty row -->
<tr> <td>&nbsp;</td> </tr>

```

```

<tr>
  <td>
    <p><strong><portletAPI:text key="UserDefinedBookmarks" bundle="nls.bookmark">
      Userdefined Bookmarks (fallback):
    </portletAPI:text></strong></p>
  </td>
</tr>
<tr>
  <td>
    <portletAPI:dataLoop pattern="name.*">
      <p><portletAPI:dataAttribute/></p>
    </portletAPI:dataLoop>
  </td>
  <td>
    <portletAPI:dataLoop pattern="url.*">
      <p><a href='<portletAPI:dataAttribute/>' target=bookmarkwindow>click</a>

      <portletAPI:if state="Maximized">
        -> <portletAPI:dataAttribute/>
      </portletAPI:if>
    </p>
    </portletAPI:dataLoop>
  </td>
</tr>
<tr>
  <td>
    <p>
      <portletAPI:if state="Normal">
        <a href='<portletAPI:createURI state="Maximized"/>'>click to MAXIMIZE</A>
      </portletAPI:if>
      <portletAPI:if state="Maximized">
        <a href='<portletAPI:createURI state="Normal"/>'>click to RESTORE</A>
      </portletAPI:if>
    </p>
  </td>
</tr>
</table>
</td>
<td>
  <img src='<%=response.encodeURL("/images/bookmarks.gif")%>'>
</td>
</tr>
</table>

```

web.xml version 5

```

<?xml version="1.0" encoding="UTF-8"?>
...
  <servlet-class>com.mycompany.portlets.bookmark.BookmarkPortlet</servlet-class>
  <init-param>
    <param-name>image.name</param-name>
    <param-value>bookmarks.gif</param-value>
  </init-param>
  <init-param>
    <param-name>jsp.view</param-name>
    <param-value>bookmarkView.jsp</param-value>
  </init-param>
...

```

4.3. Reserved Parameter Names

The following parameter names may not be used in portlet JSPs without namespace encoding as they are reserved by the WPS framework:

- names starting with a point ‘.’
- names starting with an underscore ‘_’

5. Portlet API - Advanced elements

This chapter describes features of the Portlet API that are implemented by more advanced portlets. You should already be familiar with the [basic elements](#) before you use the features described in this chapter. Advanced features include capturing and processing portlet events, messaging between portlets, caching, and portlet services.

5.1. Portlet events

Portlet events contain information about an event to which a portlet might need to respond. For example, when a user clicks a link or button, this generates an action event. To receive notification of the event, the portlet must have an event listener implemented within the portlet class. There are three types of events:

- **ActionEvents:** generated when an HTTP request is received by the portlet container that is associated with an action, such as when the user clicks a link.
- **MessageEvents:** generated when a portlet sends a message to another portlet.
- **WindowEvents:** generated when the user changes the state of the portlet window.

A portlet has a different processing and rendering sequence than a servlet. A servlet does all of its processing in the `service()` method. A portlet, on the other hand, uses a two phase processing that is split between an action processing and service. This split is necessary to accommodate communication between portlets before rendering output in the service stage. The action processing is guaranteed to complete before a portlet is called to render.

During action processing, the portlet implements an `ActionListener` interface. The `ActionListener` interface provides the `actionPerformed()` method, to which an `ActionEvent` object is passed. When a user clicks on a link or a submit button, an `ActionEvent` can be generated. The `PortletAction` can be obtained from the `ActionEvent`, which describes the triggering event. When the `actionPerformed()` method is invoked, a response object is not available because this is not a rendering step. All state changes should be handled during action processing.

Portlets should use the service phase only to render portlet output. The `service()` method is not only called following the `actionPerformed()` processing when a user clicks on a link or button in a portlet, but is also called when the portal page is refreshed. Thus, given a page with two portlets, A and B, when the user clicks on a link in portlet A, `actionPerformed()` and `doView()` is called for portlet A, but only the `doView()` method is called for portlet B. Once the content generation phase has started, no further events will be delivered. For example, messages cannot be sent from within the `beginPage()`, `service()` and `endPage()` methods. The resulting message event would not be delivered and essentially discarded.

The event listener is implemented directly in the portlet class. The listener can access the `PortletRequest` from the event and respond using the `PortletRequest` or `PortletSession` attributes.

5.1.1. Action events

An `ActionEvent` is sent to the portlet when an HTTP request is received that is associated with a `PortletAction`. To receive action events, the portlet class must implement the `ActionListener` interface and a portlet action. A portlet action can be one of the following types:

[Simple portlet action String](#)
[PortletAction object](#)

These actions are explained below.

5.1.1.1. *Simple portlet action String*

Actions created as simple actions can be executed multiple times, enabling a user's back button to work. Links created with simple portlet actions are represented in the URL rather than in the session. Therefore, portlets with simple actions can be placed on an anonymous page where no session exists. Simple portlet actions are associated with action events using the `getActionString()` method.

```
PortletURI.addAction(String simpleAction);  
String ActionEvent.getActionString();
```

Simple portlet actions are not available in the Portlet API prior to WebSphere Portal V4.2. A portlet can determine whether the portal server supports simple actions by checking the Portlet API version. The version of the Portlet API on servers that support simple actions has changed from 1.1 to 1.2. Here is example code which illustrates how to check for simple action support:

```
if ( (portletContext.getMajorVersion() <= 1 ) && (portletContext.getMinorVersion() <= 1) )  
{  
    // cannot use simple actions  
} else {  
    // simple action support is present on this server  
}
```

5.1.1.2. *PortletAction object*

The `PortletAction` object has been deprecated in favor of simple portlet action strings. It is maintained in the Portlet API to support existing portlets that use `PortletActions`.

5.1.2. Window events

A `WindowEvent` is sent by the portlet container whenever a user clicks on one of the control buttons that change the window's state, such as maximize, minimize or restore. A `WindowEvent` can be used, for example, to display more information when the user maximizes the portlet that would be shown in its normal state. To receive window events, the `WindowListener` interface must be implemented at the portlet class.

The Portlet API provides a `WindowAdapter` class that implements empty methods of the `WindowListener`. By extending `WindowAdapter`, the portlet developer needs to implement only those callback methods needed by the portlet. Without the `WindowAdapter`, you must implement all callback methods, even if the method is empty. See [Refreshing the portlet cache](#) for an example of a `WindowListener`.

5.1.3. Message events

Message events can be sent from one portlet to others if the recipient portlets are members of the same portlet application and are placed on the same page as the sending portlet. Additionally, a `DefaultPortletMessage` can cross portlet application boundaries and may be sent to all portlets on a page. A `MessageEvent` can be sent actively to other portlets only when the portlet is in the event processing cycle of the portlet container, otherwise an exception is thrown. There are two different types of messages:

- Single addressed messages: Messages sent to a specific portlet by specifying the portlet name on the `send()` method. Additionally, the message can be distinguished by
 - Self-created message: The message is sent to the target portlet in the boundary of the concrete portlet application on the same page.
 - `DefaultPortletMessage`: The message is sent to any portlet with the target name across all concrete portlet applications on the same page.
- Broadcast messages: Messages sent to all portlets of the same portlet application remaining on the same page.
 - Self-created message: The message is sent to all portlets in the boundary of the concrete portlet application on the same page.
 - `DefaultPortletMessage`: The message is sent to all portlets across all concrete portlet applications on the same page.

Message events are useful when changes in one portlet should be reflected in another one. An object with the type `PortletMessage` has to be implemented which is passed via the `MessageEvent`. The portlet receiving the message must implement the `MessageListener` interface and an object with the type `PortletMessage`.

Typically, when the user performs an action in one portlet, the action event is captured and processed. Based on the results of that action, the portlet can send a message to other portlets using the `send()` method of the `PortletContext` object.

```

...
public void actionPerformed (ActionEvent event) {
...
    if (action.getName().equals("browse")) {
        log.debug("BookmarkActionListener - browse action");
        String url = (String) action.getParameters().get("url");
        log.debug("BookmarkPortletActionListener - opening link: " + url);
...
        try {
            portlet.getConfig().getContext().send(null, new
                DefaultPortletMessage(url));
        }
        catch (AccessDeniedException ade) {
            log.error("BookmarkPortletActionListener - unable to send message.");
            log.error("URL = " + url + " - AccessDenied");
        }
    }
}
...
}

```

In the preceding sample, the "browse" action has been defined in the bookmark portlet:

```

DefaultPortletAction browseAction = new
    org.apache.jetspeed.portlets.DefaultPortletAction("browse");
browseAction.addParameter("url", url);
PortletURI portletURI = response.createReturnURI();
portletURI.addAction(browseAction);
String actionURI = portletURI.toString();

```

The send() method takes the following arguments :

portletName

The name of the portlet that receives this request. In the example above, the message is sent to `null`, which means it is broadcast to all portlet's in the same portlet application. To send the message to a specific portlet, specify the name of the portlet as it is defined by the `<portlet-name>` tag in the portlet deployment descriptor.

message

The message to be sent. The message must be a `PortletMessage` object or any subclass that implements that interface. In the example above, the message is instantiated in a `DefaultPortletMessage` object containing the `url` string from the action that was performed.

The receiving portlet has a message listener that retrieves the message using the `getMessage()` method of the message event.

```

public void messageReceived (MessageEvent event)
throws PortletException
{

```



```

PortletMessage msg = event.getMessage();

if (msg instanceof DefaultPortletMessage) {
    String url = ((DefaultPortletMessage)msg).getText();
    PortletAdapter portlet = (PortletAdapter)event.getPortlet();
    portlet.getPortletLog().debug("BookmarkPortletMessageListener
messageReceived");
    PortletRequest request = event.getRequest();
    PortletSession session = request.getSession();
    session.setAttribute("url",url);
}
}
}

```

Since the `MessageListener` set the received message as an attribute of the session, the receiving portlet must get the "url" parameter from the session.

5.2. PortletSettingsAttributesListener

This interface listens for changes to the attributes of the `PortletSettings`, like when an administrator configures a concrete portlet.

5.3. PortletApplicationSettingsAttributesListener

This interface listens for changes to the attributes of the `PortletApplicationSettings`, like when an administrator configures a concrete portlet application.

5.4. Portlet Caching

A portlet container can provide a per portlet cache which stores the content of a portlet. To configure the caching policy of a portlet, there are two properties in the portlet's deployment descriptor:

expires

indicates how long (in seconds) the content of a portlet should be cached until it expires.

- `<0`
The portlet provides static content and therefore does never expire
- `=0`
The portlet expires at once and is not cached at all
- `>0`
The time in seconds

shared

indicates if the portlet is user specific

- `YES`
This portlet provides the same content for all users -> one cache entry for all users
- `NO`
This portlet is user specific -> separate cache entries for each user

5.4.1. Refreshing the portlet cache

If servlet caching is enabled on the application server, the portlet cache holds the complete output of the portlet. As a result, the portal server does not call the portlet's `service()` or `doView()` methods when the user changes the portlet state. The `getLastModified()` method enables the portlet developer to inform the container when the current cache entry for the portlet should be invalidated, and therefore the portlet's content should be refreshed. You can use `WindowListeners` to set a new timestamp and then return the timestamp in `getLastModified()`.

The following examples show parts of a bookmark portlet that caches its output, but needs to change its content immediately if the window state changes to provide additional output.

First, the application server is configured to enable session caching. Check the **Enable Servlet Caching** option in the administrative console for the portal server. See the *WebSphere Application Server InfoCenter* for detailed instructions.

In the portlet deployment descriptor, the supported portlet states are declared and caching is enabled.

```
<cache>
  <expires>-1</expires>
  <shared>NO</shared>
</cache>
<allows>
  <maximized/>
  <minimized/>
</allows>
```

In the portlet's `getLastModified()` method, the timestamp is set to an attribute in the portlet's session.

```
public long getLastModified(PortletRequest request) {

    PortletSession session = request.getPortletSession(false);

    if(session == null) {
        return System.currentTimeMillis();
    }
    if (session != null) {
        Long lastModified = (Long) session.getAttribute(LAST_MODIFIED);
        if (lastModified != null) {
            return lastModified.longValue();
        }
    }
    return -1;
}
```

5.5. Portlet Services

Portlet services are discoverable extensions to the Portlet API which can be plugged into the portal server. That means a portlet can query the container for a specific service type and gets in return an implementation of the corresponding interface if

available. Thus specific portlet tasks can be encapsulated as portlet services and used by several portlets. The implementation of such a service can be exchanged / enhanced transparently to the portlet.

To enable portlets to use pluggable services via dynamic discovery, the Portlet API provides the `PortletService` interface. A `PortletService` is accessed from the `PortletContext.getService()` method that looks up the appropriate factory for the service, creates the service, and returns it to the portlet. Take for example a `CredentialVault` that enables portlets to access credentials for authentication. The `CredentialVaultService.class` would be available on the portal server but does not belong to the portlet application (that is, it is not packaged with the portlet's WAR file). That service can be retrieved as follows:

Example: Portlet using a Portlet Service

```
import org.apache.jetspeed.portlet.service.PortletService;
import com.ibm.wps.portletservice.credentialvault.CredentialVaultService;
...
CredentialVaultService vault =
    (CredentialVaultService) portletContext.getService(CredentialVaultService.class);
```

Various services may be implemented by different vendors, for example, a `SearchService`, `LocationService`, `ContentAccessService`, or a `MailService`. The Portlet API provides a [ContentAccessService](#).

5.5.1. Writing a portlet service

Writing a portlet service consists of four steps:

1. [Defining the interface](#)
2. [Writing the service implementation](#)
3. [Writing the service's factory](#)
4. [Registering the service](#)

Step 1 is not required if you want to implement your service against an existing interface. This is also the case for step 3 if you want to reuse an existing service factory.

5.5.1.1. Define the interface

Defining a portlet service interface requires the same careful considerations as defining any public API interface. A portlet service interface just must extend the `PortletService` interface defined in the `org.apache.jetspeed.portlet.service` package which serves as a flag like the `Serializable` interface of the Java API.

The `HelloWorldService` interface

```
package my.portlet.service;
import org.apache.jetspeed.portlet.service.*;
```

```
public interface HelloWorldService extends PortletService
{
    // my service's method
    public String sayIt();
}
```

5.5.1.2. Write the implementation

The service implementation must implement the `PortletServiceProvider` interface of the `org.apache.jetspeed.portlet.service.spi` package to be able to make use of the portlet service life cycle methods in addition to your service interface. The `PortletServiceConfig` parameter of the `init` method allows you e.g. to access the configuration of the service which will be discussed in [Configure and register the service](#). For further details see the JavaDoc of the Portlet API in the InfoCenter.

The HelloWorldServiceImpl class

```
package my.portlet.service.impl;

import org.apache.jetspeed.portlet.service.*;
import org.apache.jetspeed.portlet.service.spi.*;

public class HelloWorldServiceImpl implements HelloWorldService, PortletServiceProvider
{
    private String it = null;

    public void destroy()
    {
        // do nothing - no resources in use
    }

    public void init(PortletServiceConfig config)
    {
        it = config.getInitParameter("MY MESSAGE");

        if (it == null)
            it = "Hello world!!!";
    }

    public String sayIt()
    {
        return it;
    }
}
```

5.5.1.3. Write the factory

Usually you will not have to write a factory of your own as there are two generic factories being shipped with the portal server in the `org.apache.jetspeed.portletcontainer.service` package. The `PortletServiceDefaultFactory` always returns a new instance of a given service and the `PortletServiceCacheFactory` always returns the same instance of a given service so you do not have to implement your service as a singleton but just use this factory. Factories have to implement the `PortletServiceFactory` interface of the `org.apache.jetspeed.portlet.service.spi` package.

The HelloWorldServiceFactory class

```
package my.portlet.service.factory;

import org.apache.jetspeed.portlet.service.*;
import org.apache.jetspeed.portlet.service.spi.*;

public class HelloWorldServiceFactory implements PortletServiceFactory
{
```

```

PortletServiceProvider psp = null;

public PortletService createPortletService(Class service,
                                           Properties serviceProperties,
                                           ServletConfig servletConfig)
    throws PortletServiceUnavailableException
{
    if (psp != null) {
        return psp;
    }
    else {
        psp = new HelloWorldServiceImpl();

        psp.init(new PortletServiceConfigImpl(
            service,
            serviceProperties,
            servletConfig));

        return psp;
    }
}
}

```

5.5.1.4. *Configure and register the service*

To plug the service into the portal server the corresponding class files must be copied to the `.../lib/app` directory of the WebSphere Application Server installation.

Then the `PortletServiceRegistryService.properties` file in the `...app\wps.ear\wps.war\WEB-INF\conf` directory of the WebSphere Portal installation must be changed:

1. register the implementation as the corresponding service type
2. register the factory for the implementation
3. provide configuration parameters for the implementation

The PortletServices.properties file

```

# PortletServiceRegistryService.properties

org.apache.jetspeed.portlet.service.default.factory =
    org.apache.jetspeed.portletcontainer.service.PortletServiceDefaultFactory

...

my.portlet.service.HelloWorldService = my.portlet.service.impl.HelloWorldServiceImpl
my.portlet.service.impl.HelloWorldServiceImpl.factory =
    my.portlet.service.factory.HelloWorldServiceFactory
my.portlet.service.impl.HelloWorldServiceImpl.MY MESSAGE = Hello World (properties)!

...

```

5.5.2. ContentAccessService

To retrieve content from sources outside the intranet, usually a proxy must be used. To enable portlets to transparently make use of a proxy to access remote content, WPS provides a `ContentAccessService`. This service allows portlets to access content either from remote web sites or from JSPs and servlets of the whole portal web application. (In contrast, the `PortletContext.include` method only includes JSPs local to the portlet application.)

Hint: The disadvantage when accessing internet resources directly is that the content is simply taken and written to the portlet's output. This means that all relative links to other pages or images will be broken. This can be solved by parsing the content or use some enhanced browser portlet.

Example: Portlet using ContentAccessService

```
import org.apache.jetspeed.portlet.service.ContentAccessService;
...
ContentAccessService service =
    (ContentAccessService)portletContext.getService(ContentAccessService.class);

//get an URL
URL url = service.getURL("http://www.ibm.com", request, response);

//include the content of a web site into the portlet
service.include("http://www.ibm.com", request, response);

//include a JSP or servlet belonging to the portal web application
service.include("/templates/weather.jsp", request, response);
```

6. Deployment Descriptors

The WAR file for a portlet application must contain two descriptor documents: the Web application deployment descriptor and the portlet deployment descriptor.

6.1. Web application deployment descriptor

As with other servlets following the J2EE model, portlets are packaged as WAR files with a Web application deployment descriptor (web.xml). This descriptor defines each portlet as a servlet within the Web application, including unique identifiers for each portlet, the portlet class, and initialization parameters.

For more information about the Web application deployment descriptor, see the [Java Servlet Specification Version 2.3](#).

6.2. Portlet deployment descriptor

The following shows the structure of the portlet deployment descriptor (portlet.xml). Click any tag to get more information about its use.

Structure of the portlet deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE portlet-app-def PUBLIC "-//IBM//DTD Portlet Application 1.1//EN"
    "portlet_1.1.dtd">
<portlet-app-def>
  <portlet-app uid="uid">
    <portlet-app-name>portlet_application_name</portlet-app-name>
    <portlet id="portlet_id" href="WEB-INF/web.xml#servlet_id">
      <portlet-name>portlet_name</portlet-name>
      <cache>
        <expires>number</expires>
        <shared>yes | no</shared>
      </cache>
      <allows>
        <maximized/>
        <minimized/>
      </allows>
      <supports>
        <markup name="html | wml | chtml">
          <view output="fragment"/>
          <edit output="fragment"/>
          <help output="fragment"/>
          <configure output="fragment"/>
        </markup>
      </supports>
    </portlet>
  </portlet-app>

  <concrete-portlet-app uid="uid">
    <portlet-app-name>portlet_application_name</portlet-app-name>
    <context-param>
      <param-name>name</param-name>
      <param-value>value</param-value>
    </concrete-param>
    <concrete-portlet href="#portlet_id">
      <portlet-name>portlet_name</portlet-name>
      <default-locale>locale</default-locale>
      <language locale="locale">
```

```

<title>title</title>
<title-short>short title</title-short>
<description>description</description>
<keywords>keyword1, keyword2</keywords>
</language>
<config-param>
  <param-name>name</param-name>
  <param-value>value</param-value>
</config-param>
</concrete-portlet>
</concrete-portlet-app>
</portlet-app-def>

```

The following describes the elements that make up the portlet deployment descriptor:

<portlet-app-def>

Required. Top level element that contains information about the portlet application. This element includes exactly one <portlet-app> element and one or more <concrete-portlet-app> elements.

<portlet-app uid="uid">

Required. Contains information about the portlet application. The uid for each portlet must be unique within the portlet application. See [Guidelines for portlet UIDs](#). The following are subelements of <portlet-app>.

<portlet-app-name>

Exactly one is required. Indicates the name of the portlet application.

<portlet id="id" href="href" major-version="version" minor-version="version">

At least one is required. Contains elements describing a portlet that belongs to this portlet application. *id* and *href* is required. The *id* must be unique within the portlet application. The *href* attribute points to the identifier of the servlet, as in `WEB-INF/web.xml#servlet_id`.

Use the version attributes for maintaining and supporting multiple versions of the portlet.

The following are subelements of <portlet>:

<portlet-name>

Required. The name of the portlet. This name does not appear on the portal page.

<cache>

Optional. Describes how the portal handles caching the output of this portlet. Servlet caching must be enabled on the application server to support caching of portlet output. The following are subelements of <cache>:

<expires>

Required if <cache> is specified. Indicates the number of seconds after which the portlet's output is refreshed.

- 0 indicates that the portlet's output always expires. Each request causes the portlet to refresh its output. This is the default setting if <cache> is not present.
- Any number greater than 0 indicates the number of seconds the portlet output is cached. After the cache time expires, subsequent requests cause the portlet to refresh its output.
- -1 indicates that the portlets output never expires. After the portlet is initialized, its content is no longer refreshed.

<shared>

Required if <cache> is specified. Indicates if the portlet output is cached and shared with all users or for each individual user. Specify "yes" or "no".

<allows>

Optional. Indicates the states supported by the portlet. No more than one <allows> element can be specified for a portlet. These settings only affect the rendering of portlets in HTML. The following are subelements of <allows>:

<maximized/>

Optional. Indicates whether the portlet can be maximized. When maximized, the portlet replaces all other portlets on the portal page. If this element is present, the maximize button is rendered on the portlet's title bar.

<minimized/>

Optional. Indicates whether the portlet can be minimized. When minimized, the portlet is rendered only as a title bar. If this element is present, the minimize button is rendered on the portlet's title bar.

<supports>

Required. Indicates the modes and markup types supported by the portlet. All portlets must support the view mode. Other modes must be provided only if they are supported by the portlet. The following are subelements of <supports>:

<markup name="name">

At least one is required. Indicates the type of markup this portlet supports. *name* is one of the following values:

- html

- wml
- chtml

Indicate one <markup> tag for each markup type, for example:

```
<markup name="html">
    ...
</markup>
<markup name="wml">
    ...
</markup>
```

The following are subelements of <markup>:

<view/>

Required. Indicates that this portlet supports view mode.

<edit/>

Optional. Indicates that this portlet supports edit mode. This element is optional. If edit mode is supported, the portlet must supply methods that users can invoke to customize the portlet for their own use.

<help/>

Optional. Indicates that this portlet supports help mode. This element is optional. If help mode is supported, the portlet must supply help output that will display in place of the portlet when the user clicks the help icon.

<configure/>

Optional. Indicates that this portlet supports configure mode. This element is optional.

<concrete-portlet-app uid="uid">

At least one is required. Contains information about the concrete portlet application. The following are subelements of <concrete-portlet-app>.

<portlet-app-name>

Exactly one is required. Indicates the name of the portlet application.

<context-param>

Optional. Contains a pair of <param-name> and <param-value> elements that this concrete portlet application can accept as input parameters. A concrete portlet application can accept any number of context parameters.

Administrators can change the context parameters when they configure the concrete portlet application. Provide help information using XML comments to explain what values the portlet application can accept. The unique configuration settings for a concrete portlet application make up its [PortletApplicationSettings](#).

<concrete-portlet id="*id*" href="*href*">

At least one is required. Contains elements describing the concrete portlet that belongs to this concrete portlet application. *id* and *href* are required. The *id* must be unique within the portlet application. The *href* attribute points to the identifier of the servlet, as in `WEB-INF/web.xml#servlet_id`. The following are subelements of `<concrete-portlet>`:

<portlet-name>

Required. The name of the portlet. This name does not appear on the portal page.

<default-locale>

Required. Indicate the locale that will be used if the client locale cannot be determined.

<language locale="*locale*">

At least one is required. Provide one `<language>` element for each **locale** that your portlet supports. *locale* can have one of the following values:

- **en** English
- **de** German
- **fr** French
- **es** Spanish
- **ja** Japanese
- **ko** Korean
- **zh** Simplified Chinese
- **zh_TW** Traditional Chinese
- **pt** Brazilian Portuguese
- **it** Italian
- **iw** Hebrew
- **cs** Czechoslovakian
- **tr** Turkish
- **pl** Polish

The following are subelements of `<language>`:

<title>

Exactly one <title> is required for each <language> element. Specify a portlet title translated for the given *locale*.

<title-short>

Optional. Indicates a translated short title.

<description>

Optional. Provides a translated description of the portlet.

<keywords>

Optional. Provides any translated keywords associated with your portlet.

<config-param>

Contains a pair of <param-name> and <param-value> elements that this portlet can accept as input parameters. A portlet can accept any number of configuration parameters. Administrators can change the configuration parameters when they configure the concrete portlet. Provide help information using XML comments to explain what values the portlet application can accept. The unique configuration settings for a concrete portlet make up its [PortletSettings](#).

6.3. Linking the servlet, portlet, and concrete portlet

The definition of the servlets in the `web.xml` must be in the same order as the definition of portlets in the `portlet.xml`. The servlet identifier must be referenced by the portlet deployment descriptor, using the `href` attribute of the <portlet> tag. As shown in the following table, the `href` attribute indicates the path of the Web application descriptor in the WAR file appended by the servlet ID as the anchor.

portlet.xml	web.xml
<pre><portlet id="Portlet_1" href="WEB-INF/web.xml#Servlet_1"> <portlet-name>Mail</portlet-name> ... </portlet></pre>	<pre><servlet id="Servlet_1"> <servlet-name> MailPortlet </servlet-name> ... </servlet></pre>
<pre><portlet id="Portlet_2" href="WEB-INF/web.xml#Servlet_2"> <portlet-name>Calendar</portlet-name> ... </portlet></pre>	<pre><servlet id="Servlet_2"> <servlet-name> CalendarPortlet </servlet-name> ... </servlet></pre>

Each concrete portlet definition indicates its parent portlet using the **href** attribute of the `<concrete-portlet>` tag. As shown in the following table, the **href** attribute indicates the portlet ID as an anchor.

Portlet tag	Concrete portlet tag
<pre><portlet id="Portlet_1" href="WEB-INF/web.xml#Servlet_1"> <portlet-name>Mail</portlet-name> ... </portlet></pre>	<pre><concrete-portlet href="#Portlet_1"> <portlet-name> Mail Box </portlet-name> ... </concrete-portlet></pre>
<pre><portlet id="Portlet_2" href="WEB-INF/web.xml#Servlet_2"> <portlet-name> Calendar </portlet-name> ... </portlet></pre>	<pre><concrete-portlet href="#Portlet_2"> <portlet-name> Group calendar </portlet-name> ... </concrete-portlet></pre>

6.4. Guidelines for portlet application UIDs

The UIDs of portlet applications and concrete portlet applications must identify them unambiguously in the area of their usage, which could be world-wide. To make this possible, it is strongly recommended to follow these guidelines.

- Include the portlet's namespace in the UID, using the same format that is used for Java packages
- Add some portlet application specific description
- Add some arbitrary characters to guarantee uniqueness within the namespace:
com.ibm.wps.sample.mail.4969
- Add postfixes for the corresponding concrete portlet applications, like:
com.ibm.wps.sample.mail.4969.1

Portlet IDs must be unique within the application.

7. Compiling and testing portlets

The following steps show how to compile and test portlets, using the BookmarkPortlet as an example:

[Set up the portlet development environment](#)

[Compile the Java source](#)

[Create the deployment descriptor](#)

[Setup the WAR file directory structure](#)

[Package the portlet into a WAR file](#)

[Test the portlet](#)

7.1. Setting up a portlet development environment

Before trying any of the classes and samples in this document, you should setup an environment that makes the tasks of writing, compiling, and testing portlets easier. The following tools are recommended for developing portlets:

- WebSphere Studio Application Developer
- Portal Toolkit
- WebSphere Application Server
- WebSphere Portal developer install

The Portal Toolkit plugs in to WebSphere Studio and provides wizards for creating portlet application projects and for creating basic portlets that extend the PortletAdapter class, as well as portlet templates for ServletInvoker, JSPPortlet, and XSL (RSS) Portlet.

For better performance, you can setup WebSphere Portal and its prerequisites on one computer and setup your development tools and environment on a workstation with FTP or share access to the publish directory on the server. In this setup, you will also need the IBM Agent Controller running on the server. This tool is provided by WebSphere Studio.

Make sure that your system's PATH environment variable is setup to use JDK 1.3.0, which is the JDK used by WebSphere Application Server. You will need to use this JDK level to compile class files for use in your portlets.

7.2. Compiling Java source

Use the JDK provided by WebSphere Application Server to compile your Java source files. *Before* you compile your Java source, set the CLASSPATH for the compiler to find the JAR files for any portlet packages that your portlet uses. The following JAR files should always be set in the CLASSPATH to compile:

```
was_root/lib/app/portlet-api.jar;  
was_root/lib/app/wps.jar;  
was_root/lib/app/wpsportlets.jar
```

where `was_root` is the directory where WebSphere Application Server is installed.

If your portlet requires any servlet functionality, add the following to your CLASSPATH:

```
was_root/lib/j2ee.jar;  
was_root/lib/websphere.jar;
```

You might also need to add the following to your CLASSPATH:

```
was_root/lib/app/xalan-2.2.jar      - Include for XSL support  
was_root/lib/app/xerces-1.4.2.jar  - Include for XML support
```

Then, compile the portlet using the fully-qualified path to the Java portlet source.

```
javac -classpath %CLASSPATH% com/mycompany/portlets/BookmarkPortlet.java
```

Before packaging the portlet, the portlet class file must be packaged in the JAR file format. To create a JAR file with the name `BookmarkPortlet.jar`, enter the following command:

```
jar -cf BookmarkPortlet.jar BookmarkPortlet.class
```

7.3. Creating the deployment descriptors

In addition to the portlet code and resources, the WAR file contains the deployment descriptors, `web.xml` and `portlet.xml`, that contain the information necessary for the portal server to install and configure the portlet. If you are working with the [BookmarkPortlet samples](#) in this document, use the descriptors provided.

For complete information about the structure of the deployment descriptors, see [Deployment descriptors](#).

7.4. Setting up the WAR file directory structure

The WAR file format contains the Java classes and resources for a single portlet application. The resources can be images, JSP files, or property files containing translated text. In addition to the portlet code and resources, the WAR file contains a Web application deployment descriptor (`web.xml`) and portlet deployment descriptor, (`portlet.xml`), that contains the information necessary for the portal server to install and configure the portlets. Packaging portlet classes, resources, and descriptive information in a single file makes distribution and deployment of portlets easier.

Before you package your portlet, the class files and resources must be arranged in the directory structure described here. A portlet application exists as a structured hierarchy of directories.

/

The root directory of the portlet file structure serves as a document root for serving unprotected files for the portlet.

/images

The `/images` directory is an example of a directory name in which all images may be located that are needed by the portlet. Any directory name or number of nested directories may be used in the unprotected space of the package.

/WEB-INF

Location for all protected resources. The `/WEB-INF` directory stores the deployment descriptors and all of the runtime executable JAR files and classes that the packaged portlet requires. The portlet information directory is not part of the public document tree of the application. Files that reside in `/WEB-INF` are not served directly to a client.

/WEB-INF/lib

JAR files should be stored in the `/WEB-INF/lib` directory.

/jsp

Location for JSP files. This is a suggested path name. Your JSPs can be packaged in any location outside of the `/WEB-INF` directory.

/WEB-INF/classes

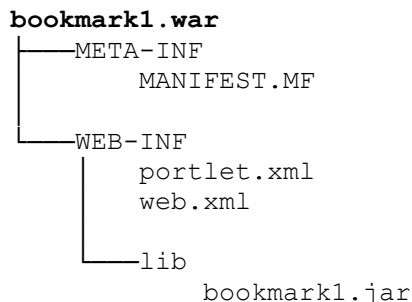
Individual class files should be stored in a directory structure within `/WEB-INF` that reflects the class package. For example, the class `BookmarkPortlet.class`, in package `com.mycompany.portlets.bookmark`, would be stored in

```
/WEB-INF/classes/  
  com/mycompany/portlets/bookmark/BookmarkPortlet.class.
```

/META-INF

The manifest for the WAR file, `manifest.mf`, resides in this directory. The manifest is in the standard JAR file format as defined by the Java 1.3 specification. The contents of the `/META-INF` directory is not served to clients.

The directory structure for `BookmarkPortlet version 1` is as follows:



7.4.1. Support for multiple markups and locales

Portal aggregation allows you to package JSPs to support multiple markups, clients, and locales. JSPs that contain mostly text, such as help JSPs, can be translated directly rather than storing strings in a resource bundle. JSPs that do not use resource bundles, such as help JSPs, should be stored in the appropriate localized location. When a portlet uses a JSP for rendering of the portlet's content, the portal searches for and selects the proper JSP based on the client type (including browser), markup language, and locale indicated in the request. To include a JSP in a portlet, you must use the function `PortletContext.include()`:

```
getPortletConfig().getContext().include(jsp_path/jspname.jsp,  
portletRequest, portletResponse);
```

To support multiple markup types and locales, the portlet's JSP's must be packaged in the WAR file using the following directory structure:

```
jsp_path/markup_type /language _region/client/jspname.jsp
```

Where

jsp_path

a path defined by the developer. For example, JSPs can be located in the root of the WAR file or in a `/jsp` directory. However, this path must not include *mime-type/language_region_variant* . The `include()` method already locates the correct JSP also in those directories.

markup_type

is either `html`, `wml`, or `html` .

language

is the language for this JSP, for example, `en`, `ja`, or `de` .

region

is the country of the JSP, for example, `US`, `UK`, or `CA`.

client

is the type of device. For example, it could indicate a JSP with browser-specific markup, such as `ie` or `ns4`. Manage Clients help describes how clients are identified by the portal server.

For example, if the client is using Internet Explorer 5 with language properties are set to English (United States), the method `include(/mypath/mytemplate.jsp, portletRequest, portletResponse)` would cause the portal server to look for the JSP in the following order.

1. `/html/mypath/ie5/en_US/mytemplate.jsp`
2. `/html/mypath/ie5/en/mytemplate.jsp`
3. `/html/mypath/ie5/mytemplate.jsp`

4. /html/mypath/en_US/mytemplate.jsp
5. /html/mypath/en/mytemplate.jsp
6. /html/mypath/mytemplate.jsp
7. /html/en_US/mytemplate.jsp
8. /html/en/mytemplate.jsp
9. /html/mytemplate.jsp
10. /mytemplate.jsp

Content accessed by a portlet JSP (for example, images or HTML pages) using `ServletResponse.encodeURL()` are not found by portal aggregation. To provide different versions of these resources based on client type, markup, and locale, the portlet must use `PortletContext.include()`.

7.5. Packaging a portlet and resources into a WAR file

Any JAR utility may be used to build a WAR file. Below are examples of how to use the JAR utility provided by WebSphere Application Server.

- To create a WAR file with the name `BookmarkPortlet.war` and include all of the files in the `/WEB-INF` and `/images` directories:

```
jar -cf BookmarkPortlet.war images WEB-INF
```

- To update an existing WAR file, `BookmarkPortlet.war` with a revised portlet descriptor:

```
jar -uf BookmarkPortlet.war WEB-INF/portlet.xml
```

- To extract the portlet descriptor from the WAR file, `BookmarkPortlet.war`:

```
jar -xf MailPortlet.war PORTLET-INF/portlet.xml
```

- To extract all files from an existing WAR file, `BookmarkPortlet.war`:

```
jar -xf BookmarkPortlet.war
```

7.5.1. Testing the sample

If you have WebSphere Studio, you can test your portlet during development, without the manual WAR file packaging and deployment to the server. To do this, you must first setup the server configuration. WebSphere Application Server Advanced Single Server Edition is required for this enablement.

If you do not have WebSphere Studio, log into the portal as an administrator, install the portlet WAR file, and then place the portlet on a portal page.

8. Credential Vault Service

Portlets running on WebSphere Portal may need to access remote applications that require some form of authentication by using appropriate credentials. Examples for credentials are user id/password, SSL client certificates or private keys. In order to provide a single sign-on user experience, portlets may not ask the user for the credentials of individual applications each time the user starts a new portal session. Instead they must be able to store and retrieve user credentials for their particular associated application and use those credentials to log in on behalf of the user.

The credential vault provides exactly this functionality. Portlets can use it through the Credential Vault PortletService.

8.1. Credential Vault organization

The portal server's credential vault is organized as follows:

- The portal administrator can partition the vault into several *vault segments*. Vault segments can be created and configured only by portal administrators.
- A vault segment contains one or more *vault slots*. Vault slots are the “drawers” where portlets store and retrieve a user's credentials. Each slot holds one credential.
- A vault slot is linked to a *resource* in a *vault implementation*, the place where the credential secrets are actually stored.
 - A vault implementation is the place where users credentials are actually stored. Examples of vault implementations include the WebSphere Portal's default database vault or the IBM Tivoli Access Manager lock box.
 - The resource within the vault implementation corresponds to an application or backend system that requires its own authentication. Examples of resources include Lotus Notes, personnel records, or bank accounts.

8.2. Vault Segments

A vault segment is flagged to be either administrator-managed or user-managed. While portlets (on behalf of a portal user) can set and retrieve credentials in both types of segments, they are permitted to create vault slots only in user managed vault segments. The following figure shows how administrator-managed vault segments can be distributed among different vault implementations. There is only one user-managed vault segment, and it resides in the vault provided by WebSphere Portal.

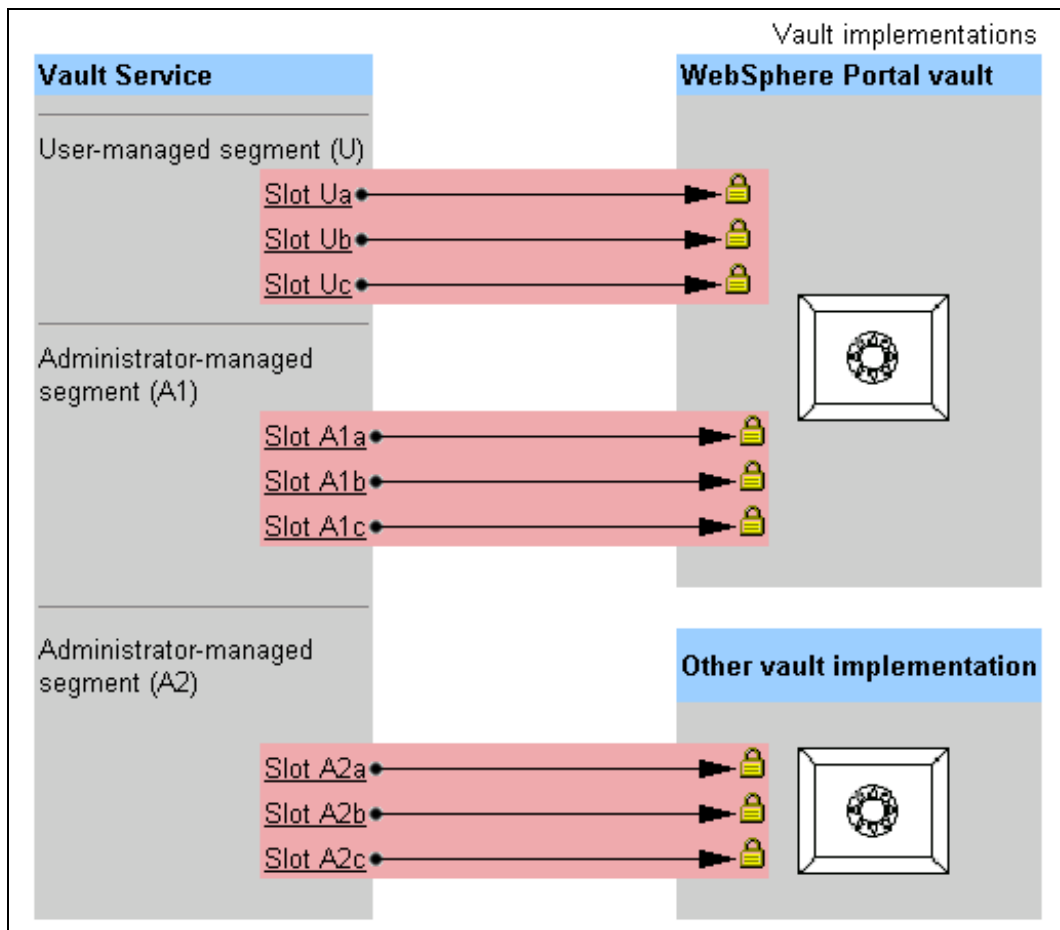


Figure 4: Vault segments and vault implementations

8.3. Vault slots

The credential vault provided by WebSphere Portal distinguishes between four different types of vault slots:

A *system* slot stores system credentials where the actual secret is shared among all users and portlets.

A *shared* slot stores user credentials that are shared among the user's portlets.

A *portlet private* slot stores user credentials that are not shared among portlets.

An *administrative slot* allows each user to store a secret for an administrator-defined resource (for example, Lotus Notes).

8.4. Credential objects

The CredentialVault PortletService returns credentials in form of credential objects:

```
com.ibm.wps.portletservice.credentialvault.credentials.Credential
```

WebSphere Portal differentiates between passive and active credential objects:

- Passive credential objects are containers for the credential's secret. Portlets that use passive credentials need to extract the secret out of the credential and do all the authentication communication with the backend itself.

Passive credential object use (pseudo code)

```
Object userSecret = credential.getUserSecret();

< portlet connects to back-end system authenticates using the user's secret >

< portlet can use the connection to communicate with the backend application >

< portlet takes care of logging at the back-end >
```

- Active credential objects hide the credential's secret from the portlet, there is no way of extracting it out of the credential. In return, active credential objects offer business methods that take care of all the authentication.

Active credential object use (pseudo code)

```
// log into the backend system
credential.login()

// get an authenticated connection to work with
URLConnection = credential.getAuthenticatedConnection();

// log out at the back end
credential.logout();
```

The second case allows portlets to trigger authentication to remote servers using standard mechanisms such as basic authentication, HTTP form-based authentication, or POP3 authentication, without even knowing the credential secrets. They can ask the portal to authenticate on their behalf and then use already authenticated connections. From a security point of view the portlets never get in touch with the credential secrets and thus there is no risk a portlet could violate any security rules like, for example, storing the secret on the portlet session. While there might not always be an appropriate active credential class available, it is the preferred type of credential object to use.

All credential types that are available within the portal are registered in a *credential type registry*. WebSphere Portal provides a small set of credential types, but additional credential objects can easily be registered in this registry. The credential classes that are shipped with the current release are:

Passive Credential Objects:

SimplePassive

This credential object stores secrets in form of serializable Java objects. As the vault service does currently not support binary large object (BLOB) secrets, this is intended for future use only.

UserPasswordPassive

Credential object that stores secrets in the form of userid/password pairs.

JaasSubjectPassive

Credential object that stores secrets in form of `javax.security.auth.Subject` objects. Again, this kind of secret cannot currently be stored by the vault service.

Active Credential Objects:

HttpBasicAuth

This credential object stores userid/password secrets and supports HTTP Basic Authentication.

HttpFormBasedAuth

This credential object stores userid/password secrets and supports HTTP Form-Based Authentication.

JavaMail

A credential object that stores userid/password pairs and leverages the authentication functionality of the `javax.mail` API.

LtpaToken

A credential object for authenticating with a backend system that is within the same WebSphere Application Server single sign-on domain as the portal.

SiteMinderToken

A credential object for authenticating with a backend system that is within the same SiteMinder single sign-on domain as the portal. This credential should be used when SiteMinder is used as an authentication proxy for the portal.

WebSealToken

A credential object for authenticating with a backend system that is within the same WebSEAL single sign-on domain as the portal. This credential should be used when a WebSEAL authentication proxy is used by the portal.

8.4.1. Storing credential objects in the PortletSession

Credential objects do not implement `java.io.Serializable` - they cannot simply be stored in the `PortletSession`. This is for security reasons. Because the credential classes store the actual credential secret as one of their private attributes, the secret could be found by anyone who has access to the application server session database.

However, you can store a credential object in the `PortletSession` as long as you ensure sure that it is not serialized in a cluster setup. One way of doing this would be to define a credential container class that stores the actual credential object as a `transient` member. This container object can then be stored in the `PortletSession` without any problems, you only have to make sure to check whether the credential object got lost during serialization and in this case retrieve it from the vault again.

Credential object session container

```
import com.ibm.wps.portletservice.credentialvault.credentials.Credential;

public class CredentialSessionContainer implements java.io.Serializable
{
    private transient Credential credential = null;

    public void setCredential(Credential cred) {this.credential = cred;}

    public Credential getCredential() {return credential;}

    public boolean hasCredential() {return credential != null;}
}
```

8.5. Credential Vault usage scenarios

Portlets that need a credential to complete their service have two options:

1. Use an existing vault slot that has been defined by the portal administrator in an administrator-managed vault segment.
2. Create a vault slot in the user-managed vault segment.

The option you choose depends on how the portlet will be used. Generally, the best solution hides the technical details of the credential vault from users. The following are some example scenarios for the use of slots.

8.5.1. Intranet Lotus Notes mail portlet

Scenario A:

A company has an intranet employee portal. Each portal user has a Lotus Notes mail server account and a Lotus Notes mail portlet will be deployed and pre-configured on one of the employee's default portal pages.

Design solution:

The Notes mail portlet needs to store the user's Notes password. As most users will actually use this portlet, the administrator needs to create a "Lotus Notes Vault Slot" for it. Using the portlet's configure mode, the administrator sets the vault slot ID for all concrete portlet instances. The portlet allows users to set their personal Notes password in edit mode. The portlet can store each user's password in the credential vault.

8.5.2. Stock of inventory portlet

Scenario B:

A company's buying department runs a portal that integrates different legacy applications. One of these applications is a mainframe ordering application that connects directly to systems of the suppliers. Several employees use the ordering portlet. However the mainframe application is secured by a simple system ID; it doesn't support several user accounts.

Design solution:

The ordering portlet needs to access the ordering application under the system ID. The administrator configures the vault slot ID during portlet deployment. The portal administrator therefore creates a vault slot in an administrator-managed vault segment and marks it as a system credential. The administrator uses the **Credential vault** portlet to store the ordering system ID and password in this slot. The buying department's employees do not have to be concerned with credentials at all.

8.5.3. Internet Mail federating POP3 portlet

Scenario C:

An internet community portal offers, among other features, a mail-federating portlet that can be used by a portal user to collect mail from a number of POP3 mail accounts.

Design solution:

The mail federating portlet is just another feature of the community portal and thus it is likely that it will be used only by some of the portal users. Furthermore, it is not clear from the outset, how many mail accounts a user wants to federate. Therefore, it does not make sense for the portal administrator to create a vault slot for this portlet. Instead, the portlet provides the user with a comfortable configuration in its edit mode. Users can add as many POP3 mailboxes as necessary. The portlet creates an additional vault slot in the user managed vault segment.

Theoretically, a user could configure two instances of the portlet on the page, one for business accounts and one for private mail accounts. Therefore, and because it most likely does not make sense to share the user's mail credentials with other portlets, the portlet created vault slots are better marked as portlet private.

8.6. Methods of the CredentialVaultService

This section provides a brief overview of the methods that a portlet can use through the CredentialVaultService.

getCredentialTypes

Description

This method returns a list of all Credential Types that are registered in the Credential Type Registry.

Return value

The Credential Types are returned in form of an `Iterator` of `String` objects.

getSlotDescription

Description

This method returns the description of a certain vault slot in a specified locale.

Parameters

- `slotId` [`String`]
The slot's identifier.
- `locale` [`Locale`]
The description locale. If set to `null`, the default locale will be used.

Return value

The description is returned in form of a `String` object.

Exceptions

- `PortletServiceException`
Thrown if the description could not be retrieved.

getAccessibleSlots

Description

This method returns a list of all vault slots that a portlet is authorized to use.

Parameters

- `request` [`PortletRequest`]
The portlet request is needed by the `CredentialVault` service in order to determine the portlet id, portlet application id and alike.

Return value

The description is returned in form of an Iterator of `CredentialSlotConfig` objects.

Exceptions

- `PortletServiceException`
Thrown if the list of slots could not be retrieved.

setCredentialSecretBinary

Description

This method sets the secret of a credential that holds binary credential data.

Parameters

- `slotId` [`String`]
The slot's identifier.
- `secret` [`byte[]`]
The credential secret data in binary form.
- `request` [`PortletRequest`]
The portlet request is needed by the `CredentialVault` service in order to determine the portlet id, portlet application id and alike.

Exceptions

- `PortletServiceException`
Thrown if the credential secret is not of the type binary or if the secret could not be set.

setCredentialSecretUserPassword

Description

This method sets the secret of a credential that holds credential data in form of a userid/password pair.

Parameters

- `slotId` [`String`]
The slot's identifier.
- `userId` [`String`]
The credential's user Id.
- `password` [`char[]`]
The credential's password.

- `request [PortletRequest]`
The portlet request is needed by the CredentialVault service in order to determine the portlet id, portlet application id and alike.

Exceptions

- `PortletServiceException`
Thrown if the credential secret is not of the type binary or if the secret could not be set.

createSlot

Description

This method creates a new vault slot.

Parameters

- `resourceName [String]`
The name of the resource.
- `segmentId [ObjectID]`
The object id of the Vault Segment in which the slot is to be created.
- `descriptions [Map]`
Human understandable descriptions of the slot, keyed by the description's `Locale`. One description - at least in the default locale - should always be set, only for portlet private slots the user may never see this description at all and in this case it can be left empty.
- `keywords [Map]`
A list of keywords that characterize the slot and can be used for searching.
- `secretType [int]`
The secret type as defined in the constants of the CredentialVaultService.
- `active [boolean]`
A map of configuration parameters that are needed to initialize the credential object. Most credential classes do not have such parameters (set to `null`). Please see the credential class' documentation for detailed information.
- `portletPrivate [boolean]`
Flag indicating whether this slot is for exclusive use of the portlet or whether it may be shared with other portlets.
- `request [PortletRequest]`
The portlet request is needed by the CredentialVault service in order to determine the portlet id, portlet application id and alike.

Return value

This method returns the `CredentialSlotConfig` object of the newly created slot.

Exceptions

- `PortletServiceException`
Thrown if the credential secret is not of the type binary or if the secret could not be set.

getDefaultVaultSegmentId

Description

Returns the `ObjectID` of the default user managed vault segment. Currently there is only one user-managed slot, so this returns the ID of *the* user-managed segment.

Return value

This method returns the `ObjectID` of the default user managed vault segment, `null` if no default user managed vault segment can be found.

Exceptions

- `PortletServiceException`
Thrown if no Vault Segment information could be retrieved.

getAllVaultSegments

Description

Returns a `List` of all Vault Segments.

Return value

This method returns all Vault Segments in form of a `List` of `VaultSegmentConfig` objects.

Exceptions

- `PortletServiceException`
Thrown if no Vault Segment information could be retrieved.

getCredential

Description

This method retrieves a certain credential from the Credential Vault.

Parameters

- `slotId` [`String`]
The slot's identifier.
- `secretType` [`int`]
The secret type as defined in the constants of the `CredentialVaultService`.
- `config` [`Map`]
A map of configuration parameters that are needed to initialize the credential object. Most credential classes do not have such parameters (set to `null`). Please see the credential class' documentation for detailed information.
- `request` [`PortletRequest`]
The portlet request is needed by the `CredentialVault` service in order to determine the portlet id, portlet application id and alike.

Return value

This method returns the `CredentialSlotConfig` object of the newly created slot.

Exceptions

- `PortletServiceException`
Thrown if the credential secret is not of the type binary or if the secret could not be set.
- `CredentialSecretNotSetException`
Thrown if the respective credential secret has not been set. If the credential in question is a user credential, the portlet should react on this exception by prompting the user for the credential data in order to call the respective `setCredential...` method.

getUserSubject

Description

This method returns the user's JAAS subject. This is a special case of the `getCredential` call.

Parameters

- `request` [`PortletRequest`]
The portlet request is needed by the `CredentialVault` service in order to determine the portlet id, portlet application id and alike.

Return value

This method returns the `CredentialSlotConfig` object of the newly created slot.

Exceptions

- `PortletServiceException`
Thrown if the credential secret is not of the type binary or if the secret could not be set.

8.7. Programming Example

The following is an example of a portlet that uses a `userid/password` credential to log into a web application that is secured with an HTTP form-based login page. The example shows how the `CredentialVault PortletService` is used to read a credential from the vault and how to use this credential. It does not show how in the portlet's edit mode the portlet queries the user for the secret and stores it in the vault.

Example: Portlet using the CredentialVault PortletService with a HttpFormBasedAuthCredential

```
import org.apache.jetspeed.portlet.service.PortletServiceException;
import com.ibm.wps.portletservice.credentialvault.CredentialVaultService;
import com.ibm.wps.portletservice.credentialvault.CredentialSecretNotSetException;
import com.ibm.wps.portletservice.credentialvault.credentials.HttpFormBasedAuthCredential;
...

public void doView (PortletRequest request, PortletResponse response)
    throws PortletException, IOException
{
    // get output stream and write out the results...
    PrintWriter writer = response.getWriter();

    // get the CredentialVault PortletService
    PortletContext context = this.getPortletConfig().getContext();
    CredentialVaultService service =
        (CredentialVaultService) context.getService(CredentialVaultService.class);

    // retrieve slotId from persistent portlet data
```

```

String slotId = (String) request.getData().getAttribute("VaultTestSlotID");
if (slotId == null) {
    writer.println("<h2>Credential not found. Please set it in the edit mode! </h2>");
    return;
}

// bundle the credential config data
HashMap config = new HashMap();
config.put( HttpFormBasedAuthCredential.KEY_USERID_ATTRIBUTE_NAME, "userid");
config.put( HttpFormBasedAuthCredential.KEY_PASSWORD_ATTRIBUTE_NAME, "password");
config.put( HttpFormBasedAuthCredential.KEY_LOGIN_URL, "EAI.yourco.com/login.jsp");
config.put( HttpFormBasedAuthCredential.KEY_LOGOUT_URL, "EAI.yourco.com /quit.jsp");
config.put( HttpFormBasedAuthCredential.KEY_USE_AUTH_COOKIES, Boolean.TRUE);

// get the actual credential from the credential vault
HttpFormBasedAuthCredential credential;
try {
    credential = (HttpFormBasedAuthCredential)
        service.getCredential(slotId, "HttpFormBasedAuth", config, request);
} catch (PortletServiceException serviceExc) {
    writer.println("<h2>Credential vault error, please contact your admin! </h2>");
    return;
} catch (CredentialSecretNotSetException userExc) {
    writer.println("<h2>Credential not set. Please set it in the edit mode! </h2>");
    return;
}

try {
    // use credential object to log in at the backend server
    credential.login();

    // get an authenticated connection to the backend server and send the actual request
    connection = credential.getAuthenticatedConnection("EAI.yourco.com/request.jsp");

    // Work with the connection: send an HTTP GET or POST and evaluate the response
    [...] // your business code

    // use credential object to log out at the backend server
    credential.logout();
} catch (IOException exc) {
    writer.println("<h2>Single-sign-on error, login at back-end failed! </h2>");
    return;
}

// get output stream and write out the results...
PrintWriter writer = response.getWriter();
}

```

8.8. Using JAAS to retrieve the user's credentials

Single sign-on support in WebSphere Portal allows users to communicate through the portal server, via portlets, with many different back end systems without being prompted multiple times for a username and password by each individual portlet's back end server. WebSphere Portal internally makes use of the [Java Authentication and Authorization Service \(JAAS\)](#) API to provide a framework for integrating single sign-on into portlets and via these portlets to back end applications.

After a user is authenticated, WebSphere Application Server generates a security context for the user. WebSphere Portal uses this security context to set up (internally to the portal) a [JAAS Subject](#) containing several Principals and Private Credentials. When the portal first handles the HTTP request, it will set up the JAAS Subject and make it available as an attribute of the User, which the portlet accesses from the PortletSession object. From the Subject, a portlet can use one of the following methods to extract the Principal and Credentials to pass to its back end application.

getPrincipals(*java.lang.Class*)

Class is optional. If no argument is specified, this method returns a set containing all Principals. Use the `Class.forName(name)` method to generate a class argument for `getPrincipals`. See the following example.

```
// Retrieve the Subject from the CredentialVaultService
PortletContext context = this.getPortletConfig().getContext();
CredentialVaultService service = (CredentialVaultService)
    context.getService(CredentialVaultService.class);
Subject subject = service.getUserSubject(portletRequest);

// Get the User DN
Set set = subject.getPrincipals(Class.forName("com.ibm.wps.sso.UserDNPrincipal"));
UserDNPrincipal userDn = null;

if (set.hasNext()) {
    userDn = (UserDNPrincipal) set.next();

    // Got it, print out what it contains.
    System.out.println("The User DN is " + userDn.getName());
}
```

The following are some of the class names that can be used with `Class.forName()` on this method

com.ibm.wps.sso.UserDNPrincipal

returns the user's distinguished name from LDAP

com.ibm.wps.sso.GroupDNPrincipal

returns the distinguished name of an LDAP group of which the user is a member. There are as many instances of `GroupDNPrincipals` as the number of LDAP groups to which the user belongs.

com.ibm.wps.sso.UserIdPrincipal

returns the ID with which the user logged on to the portal only if an authentication proxy was not used

com.ibm.wps.sso.PasswordCredential

returns the password with which the user logged on to the portal. Available only if an authentication proxy was not used.

All Principals descend from `java.security.Principal` from the standard JDK. The portlet calls `getName()` to extract the information contained in these Principals. Before the portlet can extract any information from the Subject, it must first get the `PortletSession`, then the `User`, and finally the `Subject`.

getPrivateCredentials(*java.lang.Class*)

Class is optional. If no argument is specified, this method returns a set containing all `PrivateCredentials`. Also use `Class.forName(name)` to get a Class object from a classname string for this method. To obtain the user's CORBA credential, specify `org.omg.SecurityLevel2.Credentials` as the argument for `Class.forName()`.

8.8.1. Basic authentication sample

The following example demonstrates how to extract the Subject from the PortletSession and then how to extract the user name and password from the Subject. The portlet then creates a Basic Authentication Header and attempts to connect to a URL protected by basic authentication. The URL is specified as a configuration parameter in the portlet deployment descriptor (`portlet.xml`). For output, it displays the user name, password, and the results of the connection attempt.

In the `service()` method, the JAAS Subject is extracted from the `CredentialVaultService`.

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.security.*;
import javax.security.auth.*; // for the JAAS Subject

import com.ibm.wps.sso.*; // for the Principals contained in the Subject
import com.ibm.wps.portletservice.credentialvault.CredentialVaultService; // for the
                                                                    // Credential Service
import com.ibm.ejs.security.util.Base64Coder;

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

public class URLSSOSamplePortlet extends AbstractPortlet {
    /** URL To Access */
    private String urlSpec = null;

    /**
     * Initialization Routine
     *
     * @param PortletConfig Portlet Configuration
     * @throws UnavailableException
     */
    public void init(PortletConfig portletConfig)
        throws UnavailableException {
        super.init(portletConfig);

        // Get the URL
        urlSpec = portletConfig.getAttribute("url");
    }

    /**
     * Retrieves the specified Principal from the provided Subject
     *
     * @param Subject subject
     * @param String Class Name
     * @return The values of the Principals for the given class name, null if
     *         nothing was returned
     * @throws PortletException An error occurred retrieving the Principal
     */
    private String[] getPrincipalsFromSubject(Subject subject,
                                              String className)
        throws PortletException {

        try {
            // Get the Set of Principals
            Object[] principals =
                subject.getPrincipals(Class.forName(className)).toArray();

            // Do we have any?
            if ((null == principals) || (0 == principals.length)) {
                // No principals of this class name
                return(null);
            }

            // Create the String Array to hold the values
            String[] values = new String[principals.length];

```



```

        // Populate the values Array
        for (int current = 0; current < values.length; current++) {
            values[current] = ((Principal) principals[current]).getName();
        }

        return(values);

    } catch (ClassNotFoundException cnfe) {
        throw(new PortletException("Class " + className + " could not be found",
            cnfe));
    }
}

/**
 * Extracts the UserID and Password from the JAAS Subject, creates a Basic
 * Auth Header from it, and then connects to the resource.
 *
 * @param PortletRequest Request Object
 * @param PortletResponse Response Object
 * @throws PortletException
 * @throws IOException
 */
public void service(PortletRequest portletRequest,
    PortletResponse portletResponse)
    throws PortletException,
        IOException {
    // Get the Writer from the Response
    PrintWriter writer = portletResponse.getWriter();

    PortletContext context = this.getPortletConfig().getContext();
    CredentialVaultService service = (CredentialVaultService)
        context.getService(CredentialVaultService.class);
    Subject subject = service.getUserSubject(portletRequest);

    // Grab the UserID and Password. There will only be one of each
    // contained in the Subject.
    String[] userID = getPrincipalsFromSubject(subject,
        "com.ibm.wps.sso.UserIdPrincipal");
    String[] password = getPrincipalsFromSubject(subject,
        "com.ibm.wps.sso.PasswordCredential");

    // The URL
    writer.println("<TABLE>");
    writer.println("<TR><TD><B>URL:</B></TD><TD>" + urlSpec +
        "</TD></TR>");

    // Show the UserID and Password
    writer.println("<TR><TD><B>UserID:</B></TD><TD>" + userID[0] +
        "</TD></TR>");
    writer.println("<TR><TD><B>Password:</B></TD><TD>" + password[0] +
        "</TD></TR>");

    // Create the Basic Auth Header
    String basicAuth = Base64Coder.base64Encode(userID[0] +
        ":" +
        password[0]);

    basicAuth = "Basic " + basicAuth;
    writer.println("<TR><TD><B>Basic Auth Header:</B></TD><TD>" +
        basicAuth +
        "</TD></TR>");
    writer.println("</TABLE><BR>");

    // Create the URL for our protected resource, and get a URLConnection
    try {
        URL url = new URL(urlSpec);
        HttpURLConnection con = (HttpURLConnection) url.openConnection();

        // Set our Basic Auth Header
        con.setRequestProperty("authorization", basicAuth);

        // Connect
        con.connect();

        // Get the Response Code
        String responseMessage = con.getResponseMessage();

```

```

int responseCode = con.getResponseCode();

// Were we successful?
if (URLConnection.HTTP_OK == responseCode) {
    writer.println("<P>Successfully connected to " +
        urlSpec + "!</P>");
} else {
    writer.println("<P>Unable to successfully connect to " +
        urlSpec + ", HTTP Response Code = " +
        responseCode +
        ", HTTP Response Message = \"" +
        responseMessage + "\"</P>");
}

} catch (Exception e) {
    writer.println("<P>Exception caught in HTTP Connection:</P><TT>");
    writer.println(e.getMessage());
    e.printStackTrace(writer);
    writer.println("</TT>");
}
}
}

```

8.8.2. LTPA example

Lightweight third-party authentication (LTPA) is the mechanism for providing single sign-on between WebSphere Application Server and Domino servers that reside in the same security domain. The LTPA token is carried in the HTTP request as a cookie. The example in this section assumes that:

1. The portlet is accessing a protected resource on the same WebSphere Application Server that WebSphere Portal is installed on.
2. If the resource resides on another WebSphere Application Server or Domino Server, then single sign-on is enabled between them.

Extracting the LTPA token

```

Object[] temp =
subject.getPrivateCredentials(LTPATokenCredential.class).toArray();
LTPATokenCredential ltpaToken = (LTPATokenCredential) temp[0];

// Create the LTPA Cookie Header
String cookie = "LtpaToken=" + ltpaToken.getTokenString();

// Create the URL for our protected resource, and get a URLConnection
URL url = new URL("http://myserver.example.com/ltpa protected resource");
URLConnection con = (URLConnection) url.openConnection();

// Set our LTPA token Cookie
con.setRequestProperty("cookie", cookie);

// Connect
con.connect();

```

9. References

WebSphere Portal InfoCenter

Authors: John Waller, Reinhard Brosche, Jennifer Heins, Mary Carbonara, Michelle Wallace, Ann Roy, Bill Polomchak, Richard Spinks, Ted Buckner, Laura Feldberg, Margaret MacMahon, Maggie Litton

Describes the concepts and tasks for setting up a portal site using IBM WebSphere Portal. This includes planning, installation, administration, portal design topics, and troubleshooting. The InfoCenter also includes many of the portlet development topics discussed in this document.

Portlet Development Best Practices and Coding Guidelines

Author: Marshall Lamb

Provides guidelines for how to develop well-designed portlets that operate in the portal environment without clashing with other portlets or causing performance problems.

WebSphere Portal Zone

<http://www.ibm.com/websphere/portal/developer>

Technical information for developers and administrators of WebSphere Portal. This site contains many step-by-step articles for developing portlets.

10. Trademarks

IBM, DB2, and WebSphere are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.

Windows and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

[IBM copyright and trademark information](#)

End of document