Portlet API (First DRAFT)

Please send technical comments to

Stephan Hesmer (stephan.hesmer@de.ibm.com)

Stefan Hepper (sthepper@de.ibm.com)

Thomas Schaeck (schaeck@de.ibm.com)

Status

This document is not yet complete. It covers some important aspects of portlets, but major parts of the document are still missing.

Table of Contents

1 Overview

This chapter provides an overview of the Portlet API. It introduces the concept of portlets, gives some definitions and introduces certain entities related to portlets that are usually present in portals.

1.1 What is a Portlet ?

A portlet is a web component managed by a container, that generates dynamic content. Portlets are platform independent Java classes compiled to an architecture neutral bytecode that can be loaded dynamically into and run by a web server. While servlets usually interact directly with web clients, portlets interact with web clients indirectly through portals, via a request response paradigm implemented by the portlet container. This request-response model is based on the behavior of the Hypertext Transfer Protocol (HTTP).

Portlets are specialized servlets that plug into and run in portals. Portlets are designed to be aggregatable in the larger context of a portal page. They rely on the portal infrastructure to function, e.g. access to user profile information for the current user, access to the window object that represents the window in which the portlet is displayed, participation in the portal window and action event model, access to web client information, inter-portlet messaging and a standard way of storing and retrieving per-user or per-instance data persistently.

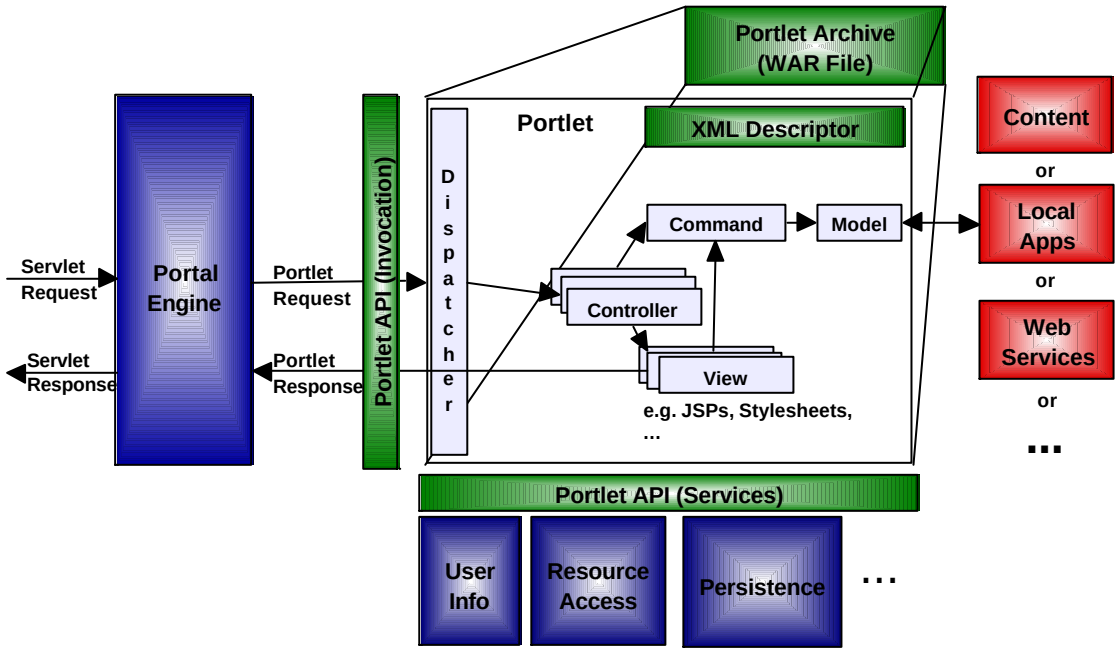

Figure 1: Portlet Principle

Usually, many portlets are invoked in the course of handling a single request, aggregating their respective produced content in one page by appending each individual portlet's output to the page. Portlets generate markup fragments that can be aggregated in the scope of a portal, containing links, actions and content suitable for aggregation within a portal. URL rewriting methods are provided that allow portlets to transparently create links, without needing to know how URLs are structured in the particular portal.

Portlets can have different modes. All portlets must implement a view mode that is responsible for displaying the portlets view, e.g. a list of stock symbols and their prices in a stocks quotes portlet. Optionally, portlets may implement edit, config and help modes in addition. A typically way how portals make the available modes of portlets accessible by displaying buttons for accessing those modes is in the title bars of portlets on a portal page.

In many aspects, the Portlet API is an extension of the Servlet API, while in other aspects, it restricts function provided by the Servlet API to the subset that is allowed for portlets running in the context of a portal. For example, the Portlet API's `PortletContext` does not allow portlets to obtain a request dispatcher to perform forward calls, it only provides portlets with an include call, the `PortletResponse` does not allow portlets to invoke the `sendRedirect` or `sendError` methods as these things may only be done by the portal that contains the portlets.

1.2 Definitions

In the rest of this paper we'll use the terms portlet class, portlet class instance, concrete portlet, and concrete portlet instance as defined below:

1.2.1.1 Portlet Class

The term portlet class denotes the code of a portlet. Portlet classes are derived from the the `PortletAdapter` base class. A portlet class is an implementation aspect that will never be visible to administrators or users.

Example: A class `sample.portlet.StockQuotePortlet` that is the implementation of a stock quote portlet.

1.2.1.2 Portlet Class Instance

A portlet class instance is an instance of a portlet class, parameterized by a `PortletConfig` object. One `PortletConfig` exists per portlet class instance. Portlet class instances are created when an administrator deploys a new Portlet Application WAR file or when the server restarts. Portlet class instances are never visible to administrators or users.

Example: An portlet class instance defined in a WAR file, named "StockQuotePortlet" of the class `sample.portlet.StockQuotePortlet`. Might have init parameters like JSPs to use for dispaying stock quotes set by the portlet provider.

1.2.1.3 Concrete Portlet

Concrete portlets are what is typically visible to administrators and users as “Portlets” in portal UIs. They usually are administrable in admin UIs, e.g. to associate access rights or change settings and they typically can be selected by users in page customizers and be put on pages which creates a concrete portlet instance.

A concrete portlet is a portlet instance parametrized with individual `PortletSettings` on a per-request basis. For each single portlet instance, there may be many concrete portlets resulting from different parametrizations. There is one `PortletSettings` object per concrete portlet. Concrete portlets are created when an administrator deploys a new portlet application WAR file that has definitions of concrete portlets in it or when an administrator uses the admin UI of the portal to create new concrete portlets. The same concrete portlet can be shared across many users.

Example: Concrete portlet named “XYZ Stock Quotes” of the stocks portlet instance named “StockQuotePortlet” that is parametrized with a unique source URL provided by a stock quote service to obtain stock quotes.

1.2.1.4 Concrete Portlet Instances

Concrete Portlet Instances are what is visible as a portlet on a user’s page. They are concrete portlets, additionally parameterized by a `PortletData` object on a per-request basis. There can be many concrete portlet instances per concrete portlet.

Example: A user selects a concrete portlet named “XYZ Stock Quotes” of the portlet instance “StockQuotePortlet” of the class `sample.portlet.StockQuotePortlet` and puts it on a personal page, creating a concrete portlet instance and a reference from the page to the new concrete portlet instance. The concrete portlet instance can be parametrized by the user via the edit mode, defining the particular stocks to track.

1.3 Entities and Relations in Portals

Figure 2 shows an example of how these entities can be related to each other. We assume that the portal implementation has the notion of users and groups, where a user can be in 0..n groups. Users and groups have access to 0..n pages, e.g. determined by a combination of access rights and explicit selection of the desired subset of accessible pages. The relations between users, pages, page groups etc are irrelevant for a portlet container, they can be different in different portals.

Portal pages have references to concrete portlets instances, more than one page may have a reference to a particular concrete portlet instance. Many concrete portlet instances may be instantiated from a single concrete portlet. Many concrete portlets may exist that are based on the same portlet instance. Many portlet instances may be instantiated from a single portlet class.

The relations that are important for the portlet container are those between concrete portlet instances, concrete portlets, portlet (class) instances and portlet classes.

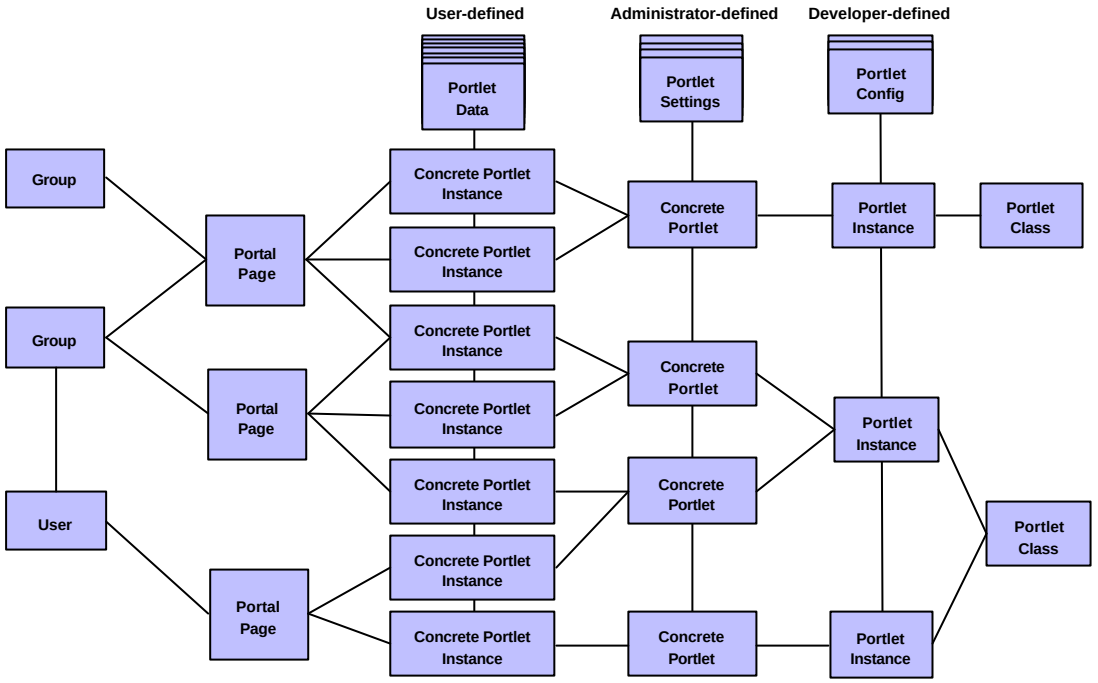

Figure 2: Portlet related entities and their relations

A portlet can work with the following data:

Portlet Configuration – the unchangeable configuration of a portlet. The portlet configuration is accessible through the `PortletConfig` object. The portlet configuration is read only and persistent. Objects associated with data from the portlet configuration may be held in portlet instance variables of portlets since there is one instance of a portlet class per configuration. The portlet configuration data is defined by portlet developers at design time and cannot be changed by portal administrators or user.

Portlet Settings – per-concrete portlet configuration settings that can be changed by portal administrators at any time. Usually portlet settings are used by the portlet's edit and view modes and updated by the config mode. The portlet settings are accessible through the `PortletSettings` object associated with the request. Portlet settings are read/write accessible and persistent. Objects associated with data from portlet settings may not be held in portlet instance variables, since the same instance is usually parameterized with many different `PortletSettings` objects during its lifetime.

Portlet Data – per concrete portlet instance data, e.g. user preferences for the portlet that usually are changed by the portlet's edit mode. Portlet data are accessible through the `PortletData` object associated with the request. Portlet data are read/writable and persistent. Objects associated with data from portlet data may not be held in portlet instance variables, since the same instance is usually parameterized with many different `PortletData` objects during its lifetime.

User Profile – per user data shared across all portlet instances across all portlet applications. The user profile is accessible via the `User` object read only.

Portlet Session – non-persistent session data scoped by portlet application. The portlet session is read/writable and accessible via the `PortletSession` object.

1.4 What is a Portlet Container?

The portlet container, in conjunction with a portal server, provides the environment over which requests and responses are set, decodes MIME based requests, and formats MIME based responses. A portlet container also contains and manages portlets through their lifecycle. A portlet container can be built based on an application server's servlet container.

1.5 An Example

A client program, such as a web browser, accesses a web server and makes an HTTP request. This request is processed by the web server and is handed off to the servlet container. The servlet container invokes the portal server servlet that in turn calls the portlet container, most likely several times in order to aggregate multiple portlets on the result page.

For each invocation, the portlet container determines which portlet to invoke based on a passed ID and its internal configuration and calls it with objects representing the request and response. The portlet container can be implemented as an extension of the servlet container.

The portlet uses the request object to find out who the remote user is, what the attributes of the remote user are, what kind of client the user has, what HTML form parameters may have been sent as part of this request, the data associated with the user and invoked concrete portlet instance, and other relevant data. The portlet can then perform whatever logic it was programmed with and can generate data to send back to the portal server. It sends this data back to the portal server via the response object. Once the portlet is done with the request, the portlet container ensures that the response is properly flushed and returns control back to the portal server. The portal server usually aggregates output from many portlets in composite pages.

1.6 Comparing Portlets with Other Technologies

Portlets are based on the Servlet technology, but are designed particularly for use in portals. In many aspects, the Portlet API is an extension of the Servlet API, while in other aspects, it restricts function provided by the Servlet API to the subset that is allowed and appropriate for portlets running in the context of a portal.

For use in portals, Portlets have the following advantages over servlets:

- Portlets can exploit an event mechanism for window events, action events and message events
- Portlets are invoked with portlet request, portlet response and portlet session objects wrapping the original servlet request, servlet response and HTTP session

to prevent portlets from disturbing the portal and that provide access to portlet or user related objects.

- Portlets can exist multiple times on a single page, i.e. there can me more than one concrete portlet instance of the same portlet on a single portal page which can be invoked with a single request.

2 The PortletAdapter Class

Portlets are designed to run in a portal environment, producing markup fragments that can be aggregated in pages and reacting on certain portal-related events. All portlets must inherit from the `Portlet` base class directly of indirectly. The recommended way of writing portlets is to inherit from the `PortletAdapter` class.

2.1 Request Handling Methods

The `service` method of a portlet is called for each request that the portlet container routes to an instance of a portlet. The container invokes the portlet's service method with `PortletRequest` (see Chapter 0) and `PortletResponse` (see Chapter 5) objects as arguments.

As portlets are specialized servlets, multiple request threads may be executing within the service method of a portlet at any time. Therefore, a portlet may not use instance variables to store any information associated with incoming requests.

2.1.1 Conditional Rendering Support

The `Portlet` class defines the `getLastModified` method to support conditional get operations. A conditional get operation is one in which the portal requests portlet content indicating that the content body should only be sent if it has been modified since a specified time.

Portlets that implement the `service` method and that provide content that does not necessarily change from request to request should implement this method to aid in efficient utilization of server resources.

2.2 Number of Instances

The portlet container must use only one instance of a portlet class per portlet definition.

2.3 Portlet Life Cycle

A portlet is managed through a well defined life cycle that defines how it is loaded, instantiated and initialized, handles requests from portals, and how it is taken out of service. This life cycle is expressed in the API by the `init`, `service`, and `destroy` methods.

2.3.1 Loading and Instantiation

The portlet container is responsible for loading and instantiating a portlet. The instantiation and loading can occur when the engine is started or it can be delayed until the container determines that it needs the portlet to service a request.

First, a class of the portlet's type must be located by the portlet container. If needed, the portlet container loads a portlet using normal Java class loading facilities from a local file system, a remote file system, or other network services.

After the container has loaded the portlet class, it instantiates an object instance of that class for use. It is important to note that there can be more than one instance of a given portlet class in the portlet container. For example, this can occur where there was more than one portlet definition that utilized a specific portlet class with different initialization parameters.

2.3.2 Initialization

After the portlet object is loaded and instantiated, the container must initialize the portlet before it can handle requests from the portal. Initialization is provided so that a portlet can read any persistent configuration data, initialize costly resources (such as JDBC™ based connection), and perform any other one-time activities. The container initializes the portlet by calling the `init(PortletConfig)` method with an object implementing the `PortletConfig` interface. This configuration object allows the portlet to access name-value initialization parameters from the portlet container's configuration information. The configuration object also gives the portlet access to an object implementing the `PortletContext` interface which describes the runtime environment that the portlet is running within. See Chapter 3, "Portlet Context" for more information about the `PortletContext` interface.

2.3.2.1 Error Conditions on Initialization

During initialization, the portlet instance can signal that it is not to be placed into active service by throwing an `UnavailableException` or `PortletException`. If a portlet instance throws an exception of this type, it must not be placed into active service and the instance must be immediately released by the portlet container. The `destroy` method is not called in this case as initialization was not considered to be successful.

After the instance of the failed portlet is released, a new instance may be instantiated and initialized by the container at any time. The only exception to this rule is if the `UnavailableException` thrown by the failed portlet which indicates the minimum time of unavailability. In this case, the container must wait for the minimum time of unavailability to pass before creating and initializing a new portlet instance.

2.3.2.2 Tool Considerations

When a tool loads and introspects a portlet application, it may load and introspect member classes of the web application. This will trigger static initialization methods to be executed.

Because of this behavior, a Developer should not assume that a portlet is in an active container runtime unless the `init` method of the `Portlet` interface is called. For example, this means that a portlet should not try to establish connections to databases or Enterprise JavaBeans™ compenent architecture containers when its static (class) initialization methods are invoked.

2.3.3 Request Handling

After the portlet is properly initialized, the portlet container may use it to handle requests. Each request is represented by a request object of type `PortletRequest` and the portlet can create a response to the request by using the provided object of type `PortletResponse`. These objects are passed as parameters to the `service` method portlets. The default implementation of the `service` method provided in the `PortletAdapter` base class detects the requested mode and dispatches the request to the `doView`, `doEdit`, `doHelp`, or `doConfig` methods.

It is important to note that a portlet instance may be created and placed into service by a portlet container but may handle no requests during its lifetime.

2.3.3.1 Multithreading Issues

During the course of servicing requests from clients, a portlet container may send multiple requests from multiple clients through the `service` method of the portlet at any one time. This means that the Developer must take care to make sure that the portlet is properly programmed for concurrency.

If a Developer defines a `service` method with the `synchronized` keyword, the portlet container will, by necessity of the underlying Java runtime, serialize requests through it. It is strongly recommended that developers not synchronize the `service` method nor any of the `doView`, `doEdit`, `doConfig`, or `doHelp` methods.

2.3.3.2 Exceptions During Request Handling

A portlet may throw either a `PortletException` or an `UnavailableException` during the service of a request. A `PortletException` signals that some error occurred during the processing of the request and that the container should take appropriate measures to clean up the request. An `UnavailableException` signals that the portlet is unable to handle requests either temporarily or permanently.

If a permanent unavailability is indicated by the `UnavailableException`, the portlet container must remove the portlet from service, call its `destroy` method, and release the portlet instance.

If temporary unavailability is indicated by the `UnavailableException`, then the container may choose to not route any requests through the portlet during the time period of the temporary unavailability. The portlet container may choose to ignore the distinction between a permanent and temporary unavailability and treat all `UnavailableExceptions` as permanent, thereby removing a portlet that throws any `UnavailableException` from service.

2.3.3.3 Thread Safety

A Developer should note that implementations of the request and response objects are not guaranteed to be thread safe. This means that they should only be used in the scope of the request handling thread. References to the request and response objects should not be given to objects executing in other threads as the behavior may be nondeterministic.

2.3.4 End of Service

The portlet container is not required to keep a portlet loaded for any period of time. A portlet instance may be kept active in a portlet container for a period of only milliseconds, for the lifetime of the portlet container (which could be measured in days, months, or years), or any amount of time in between.

When the portlet container determines that a portlet should be removed from service (for example, when a container wants to conserve memory resources, or when it itself is being shut down), it must allow the portlet to release any resources it is using and save any persistent state. To do this the portlet container calls the `destroy` method of the `Portlet` interface.

Before the portlet container can call the `destroy(PortletConfig)` method, it must allow any threads that are currently running in the `service` method of the portlet to either complete, or exceed a server defined time limit, before the container can proceed with calling the `destroy` method.

Once the `destroy` method is called on a portlet instance, the container may not route any more requests to that particular instance of the portlet. If the container needs to enable the portlet again, it must do so with a new instance of the portlet's class.

After the `destroy` method completes, the portlet container must release the portlet instance so that it is eligible for garbage collection.

2.4 Portlet Modes

The portlet API defines four portlet modes: view, edit, configure, and help. The `PortletAdapter` class defines a default implementation of the `service` method for handling client requests that dispatches requests to the `doView`, `doEdit`, `doHelp`, and `doConfigure` methods which correspond to the four modes.

2.4.1 View Mode

The view mode displays the normal view of a portlet, e.g. a list of stock symbols and current prices for a stock quote portlet. The view mode may include one or more views between which the user can navigate or it may just consist of one single view without any user interaction. The view mode of a portlet must be implemented within the `doView` method.

2.4.2 Edit Mode

The edit mode of a portlet displays one or more edit views that let the user change portlet data, e.g. the list of stock symbols he is interested in for the example of a stock quote portlet. Typcally, the edit mode will update attributes of the `PortletData` that belongs to the current concrete portlet instance to make changes persistent. If present, the edit mode of a portlet must be implemented in the `doEdit` method of a portlet.

2.4.3 Config Mode

The config mode of a portlet displays one or more configuration views that let administrators configure portlet settings, e.g. the source for stock quotes to use in the example of a stock quote portlet. Typically, the config mode will update attributes of the `PortletSettings` that belong to the current concrete portlet. If present, the config mode must be implemented in the `doConfig` method of a portlet.

2.4.4 Help Mode

The help mode of a portlet displays one or more help views that help users or administrators to understand how the portlet works. Portlets may provide simple help views that explain the entire portlet in coherent text or provide context-sensitive help. If present, the help mode of a portlet must be implemented in the `doHelp` method.

3 Portlet Context

The `PortletContext` defines a portlet's view of the portlet application within which the portlet is running. The `PortletContext` also allows a portlet to access resources available to it. Using such an object, a portlet can log events, obtain URL references to resources, and set and store attributes that other portlets in the context can use. The Container Provider is responsible for providing an implementation of the `PortletContext` interface in the portlet container.

3.1 Scope of a PortletContext

There is one instance of the `PortletContext` interface associated with each portlet application deployed into a container. In cases where the container is distributed over many virtual machines, there is one instance per portlet application per VM.

3.2 Initialization Parameters

A set of context initialization parameters can be associated with a portlet application and are made available by the following methods of the `PortletContext` interface:

- `getInitParameter`
- `getInitParameterNames`

Initialization parameters can be used by an application developer to convey setup information, such as a webmaster's e-mail address or the name of a system that holds critical data.

3.3 Context Attributes

A portlet can bind an object attribute into the context by name. Any object bound into a context is available to any other portlet that is part of the same portlet application. The following methods of `PortletContext` interface allow access to this functionality:

- `setAttribute`
- `getAttribute`
- `getAttributeNames`
- `removeAttribute`

3.3.1 Context Attributes in a Distributed Container

Context attributes exist locally to the VM in which they were created and placed. This prevents the `PortletContext` from being used as a distributed shared memory store. If information needs to be shared between portlets running in a distributed environment, that information should be placed into a session (See Chapter 8, "Sessions"), a database or set in an Enterprise JavaBean.

3.4 Resources

The `PortletContext` interface allows direct access to the static document hierarchy of content documents, such as HTML, GIF, and JPEG files that are part of the web application via the following methods of the `PortletContext` interface:

- `getResourceAsStream`

The `getResourceAsStream` methods either takes just a `String` argument giving the path of the resource relative to the root of the context or additionally a `Client` and `Locale` object. It is important to note that these methods give access to static resources from whatever repository the server uses. This hierarchy of documents may exist in a file system, in a web application archive file, on a remote server, or some other location. These methods are not used to obtain dynamic content.

3.5 Container Information

The `PortletContext` interface provides these methods to obtain information about the portlet container:

- `getContainerInfo`
- `getMajorVersion`
- `getMinorVersion`

3.6 Logging

To allow portlets to log messages, the `PortletContext` interface provides the method `getLog`.

3.7 Access to Portlet Services

To allow portlets to discover and use portlet services, the `PortletContext` interface provides the `getService` method.

3.8 National Language Support

The `getText` method of the `PortletContext` returns the localized text resource with the given key and using the given locale.

3.9 Sending Events

Portlets can send messages to other portlets in the same portlet application using the `send` method of the `PortletContext`.

3.10 Including Servlet URIs

Portlets can include servlets or JSPs in the rendering process by calling the `include` method of the `PortletContext`.

4 The Portlet Request

The `PortletRequest` interface extends the `HttpServletRequest` interface and adds portal-specific methods:

- `getClient`: returns a `Client` object that encapsulates information about the client.
- `getData`: returns a `PortletData` object that provides access to the per-concrete portlet instance data, typically user preferences in the form of name-value pairs.
- `getSettings`: returns a `PortletSettings` object that provides access to the per-concrete portlet data, typically settings defined by administrators or defaults provided by developers in the form of name value pairs.
- `getMode`: returns the current mode of concrete portlet instance. Possible modes are `Portlet.Mode.VIEW`, `Portlet.Mode.EDIT`, `Portlet.Mode.HELP`, and `Portlet.Mode.CONFIGURE`.
- `getUser`: returns the user who originated the request

- `getWindow`: returns a `PortletWindow` object that represents the window in which the concrete portlet instance is displayed
- `getPortletSession`: returns a `PortletSession` object that represents session state scoped to the portlet application instance to which the portlet belongs.

5 The Portlet Response

The `PortletResponse` interface extends the `HttpServletResponse` interface. It provides namespacing functionality in order to establish isolation between multiple concrete portlet instances.

The following methods can be used to create URIs to be embedded in the generated portlet markup so that clicking on the links associated with these URIs will result in action events being sent to the portlet in the context of the correct concrete portlet instance.

The `createURI` method creates a `PortletURI` object to which `PortletActions` may be attached. It can either be called with no parameter or with the desired new window state.

As portlets must create markup fragments rather than entire pages like servlets, the portlet container imposes restrictions on the usage of certain methods that the `PortletResponse` inherits from the `HttpServletResponse`.

For example, certain types of headers may not be set by portlets, portlets may not send HTTP errors, etc.

6 The User

The `User` interface gives access to information about the current user that portlets may use for personalization purposes. A `User` object can be obtained from the `PortletRequest` by calling the `getUser` method.

The `User` interface gives read-only access to user attributes through the `getAttributeNames` and `getAttribute` methods.

7 Portlet Events

The Portlet API supports action events, message events, window events, and page events.

7.1 Action events

Action events are triggered by the portlet container when a user clicks on an action reference embedded in markup previously created by the listening portlet using the `encodeURL` method.

Portlets that want to receive action events must implement the `ActionListener` interface which defines the method `actionPerformed(ActionEvent)`.

The `ActionEvent` interface defines the methods `getAction`, `getPortlet`, and `getRequest` to obtain the `PortletAction` object, the target portlet and the `PortletRequest` object respectively.

Action listeners may use the `PortletRequest` object to access any form parameters associated with the action event.

7.2 Message events

Message events are triggered by the portlet container when another portlet sends a message to the listening portlet.

Portlets that want to receive message events must implement the `MessageListener` interface which defines the method `messageReceived(MessageEvent)`.

The `MessageEvent` interface defines the methods `getMessage`, `getPortlet`, and `getRequest`. The `getMessage` method returns the `PortletMessage` object that represents the received message.

7.3 Window events

Window events are triggered by the portlet container when the window that displays the portlet is modified (e.g. maximized, minimized, detached, …).

Portlets that want to receive window events must implement the `WindowListener` interface that defines the `getPortlet` and `getRequest` methods.

7.4 Portlet Settings Attributes events

Portlet settings attribute events are triggered by the portlet container when attributes of a `PortletSettings` object associated with a portlet change.

Portlets that want to receive such events must implements the `PortletSettingsAttributesListener` interface that defines the methods `attributeAdded`, `attributeReplaced`, and `attributeRemoved`. When the portlet container invokes these methods, it provides an object that implements the `PortletSettingsAttributeEvent` interface which provides the methods `getName`, `getValue`, and `getPortletSettings`. The `getName` method returns the name of the affected attribute, the `getValue` method returns its value and the `getPortletSettings` method returns all current settings.

7.5 Portlet Application Settings Attributes events

Portlet application settings attribute events are triggered by the portlet container when attributes of a `PortletApplicationSettings` object associated with a portlet change.

Portlets that want to receive such events must implements the `PortletApplicationSettingsAttributesListener` interface that defines the methods `attributeAdded`, `attributeReplaced`, and `attributeRemoved`. When the portlet container invokes these methods, it provides an object that implements the `PortletApplicationSettingsAttributeEvent` interface which provides the methods `getName`, `getValue`, and `getPortletApplicationSettings`. The `getName` method returns the name of the affected attribute, the `getValue` method returns its value and the `getPortletApplicationSettings` method returns all current settings.

7.6 Page Events

Page events are triggered by the portlet container when a page referencing a portlet that implements the `PortletPageListener` interface. The page listener interface defines the methods `beginPage` and `endPage`. In these methods, the portlet can do things that need to happen before or after page aggregation, e.g. set cookies on the response.

8 Portlet Window

The `PortletWindow` interface defines an abstraction for windows in which portlets are displayed. A portlet can obtain a `PortletWindow` object with information about the current portlet window from the `PortletRequest`.

9 Portlet Applications

Tbd

9.1 Relationshi p to PortletContext

Tbd

9.2 Elements of a Portlet Application

- Portlets
- JavaServer Pages
- Utility Classes
- Static documents (html, images, sounds, etc.)

- Client side applets, beans, and classes
- Descriptive meta information which ties all of the above elements together.

9.3 Web Application Archive File for Portlet Applications

Web Application Archive Files are WAR files as described in the Servlet Specification with additional portlet-related information in the portlet.xml file.

10 Tag Library Support

In order to allow JSPs invoked by portlets to access portal related data, several JSP tags are defined that give access to the `User` object, the `PortletData` object, and the `PortletSettings` object as well as for URL encoding.

The request, response, and session can be accessed through the `<jsp:useBean>` and related tags.

10.1 The User Tag

Tbd

10.2 The PortletData Tag

Tbd

10.3 The PortletSettings Tag

Tbd

10.4 The Encode Namespace Tag

Tbd

11 Portlet Services

Tbd