Portlet Specification

Version 1.0 (20030314~~20030313~~)

Send comments about this document to: `jsr168-comments@jcp.org`

March 14, 2003~~March 13, 2003~~

Alejandro Abdelnur (alejandro.abdelnur@sun.com)
Stefan Hepper (sthepper@de.ibm.com)

Java^{TM} Portlet Specification ("Specification")

Version: 1.0

Status: Pre-FCS

Specification Lead: Sun Microsystems, Inc. and IBM Corporation ("Specification Lead")

Release: March 14, 2003~~March 13, 2003~~

Contents

PLT.1

Preface

This document is the Portlet Specification, v1.0. The standard for the Java portlet API is described here.

PLT.1.1 Additional Sources

The specification is intended to be a complete and clear explanation of Java portlets, but if questions remain the following may be consulted:

- A reference implementation (RI) has been made available which provides a behavioral benchmark for this specification. Where the specification leaves implementation of a particular feature open to interpretation, implementators may use the reference implementation as a model of how to carry out the intention of the specification
- A Technology Compatibility Kit (TCK) has been provided for assessing whether implementations meet the compatibility requirements of the Java portlet API standard. The test results have normative value for resolving questions about whether an implementation is standard
- If further clarification is required, the working group for the Java portlet API under the Java Community Process should be consulted, and is the final arbiter of such issues

Comments and feedback are welcomed, and will be used to improve future versions.

PLT.1.2 Who Should Read This Specification

The intended audience for this specification includes the following groups:

- Portal server vendors that want to provide portlet engines that conform to this standard
- Authoring tool developers that want to support web applications that conform to this specification
- Experienced portlet authors who want to understand the underlying mechanisms of portlet technology

We emphasize that this specification is not a user's guide for portlet developers and is not intended to be used as such.

PLT.1.3 API Reference

An accompanying javadoc™ , includes the full specifications of classes, interfaces, and method signatures.

PLT.1.4 Other Java™ Platform Specifications

The following Java API specifications are referenced throughout this specification:

- Java 2 Platform, Enterprise Edition, v1.3 (J2EE™)
- Java Servlet™, v2.3
- JavaServer Pages™, v1.1 (JSP™)

These specifications may be found at the Java 2 Platform,Enterprise Edition website: http://java.sun.com/j2ee/.

PLT.1.5 Other Important References

The following Internet specifications provide information relevant to the development and implementation of the portlet API and standard portlet engines:

- RFC 1630 Uniform Resource Identifiers (URI)
- RFC 1738 Uniform Resource Locators (URL)
- RFC 2396 Uniform Resource Identifiers (URI): Generic Syntax
- RFC 1808 Relative Uniform Resource Locators
- RFC 1945 Hypertext Transfer Protocol (HTTP/1.0)
- RFC 2045 MIME Part One: Format of Internet Message Bodies
- RFC 2046 MIME Part Two: Media Types
- RFC 2047 MIME Part Three: Message Header Extensions for non-ASCII text
- RFC 2048 MIME Part Four: Registration Procedures
- RFC 2049 MIME Part Five: Conformance Criteria and Examples
- RFC 2109 HTTP State Management Mechanism
- RFC 2145 Use and Interpretation of HTTP Version Numbers
- RFC 2616 Hypertext Transfer Protocol (HTTP/1.1)
- RFC 2617 HTTP Authentication: Basic and Digest Authentication
- ISO 639 Code for the representation of names of languages
- ISO 3166 Code (Country) list

Online versions of these RFC and ISO documents are at:

- `http://www.rfc-editor.org/`
- `http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt`

- `http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html`

The World Wide Web Consortium (http://www.w3.org/) is a definitive source of HTTP related information affecting this specification and its implementations.

The Extensible Markup Language (XML) is used for the specification of the Deployment Descriptors described in Chapter 13 of this specification. More information about XML can be found at the following websites:

```
http://java.sun.com/xml
http://www.xml.org/
```

PLT.1.6 Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in [RFC2119].

PLT.1.7 Providing Feedback

We welcome any and all feedback about this specification. Please e-mail your comments to jsr168-comments@sun.com.

Please note that due to the volume of feedback that we receive, you will not normally receive a reply from an engineer. However, each and every comment is read, evaluated, and archived by the specification team.

PLT.1.8 Acknowledgements

The formulation of this community draft is the result of the teamwork of the JSR168 expert group.

PLT.2

Overview

PLT.2.1 What is a Portal?

A portal is a web based application that –commonly- provides personalization, single sign on, content aggregation from different sources and hosts the presentation layer of Information Systems. Aggregation is the action of integrating content from different sources within a web page. A portal may have sophisticated personalization features to provide customized content to users. Portal pages may have different set of portlets creating content for different users.

PLT.2.2 What is a Portlet?

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

The content generated by a portlet is also called a fragment. A fragment is a piece of markup (e.g. HTML, XHTML, WML) adhering to certain rules and can be aggregated with other fragments to form a complete document. The content of a portlet is normally aggregated with the content of other portlets to form the portal page. The lifecycle of a portlet is managed by the portlet container.

Web clients interact with portlets via a request/response paradigm implemented by the portal. Normally, users interact with content produced by portlets, for example by following links or submitting forms, resulting in portlet actions being received by the portal, which are forwarded by it to the portlets targeted by the user's interactions.

The content generated by a portlet may vary from one user to another depending on the user configuration for the portlet.

PLT.2.3 What is a Portlet Container?

A portlet container runs portlets and provides them with the required runtime environment. A portlet container contains portlets and manages their lifecycle. It also provides persistent storage for portlet preferences. A portlet container receives requests from the portal to execute requests on the portlets hosted by it.

A portlet container is not responsible for aggregating the content produced by the portlets. It is the responsibility of the portal to handle the aggregation.

A portal and a portlet container can be built together as a single component of an application suite or as two separate components of a portal application.

PLT.2.4 An Example

The following is a typical sequence of events, initiated when users access their portal page:

- A client (e.g., a web browser) after being authenticated makes an HTTP request to the portal
- The request is received by the portal
- The portal determines if the request contains an action targeted to any of the portlets associated with the portal page
- If there is an action targeted to a portlet, the portal requests the portlet container to invoke the portlet to process the action
- A portal invokes portlets, through the portlet container, to obtain content fragments that can be included in the resulting portal page
- The portal aggregates the output of the portlets in the portal page and sends the portal page back to the client

PLT.2.5 Relationship with Java 2 Platform, Enterprise Edition

The Portlet API v1.0 is based on the Java 2 Platform, Enterprise Edition, v1.3. Portlet containers and portlets meet the requirements, described in the J2EE specification, for executing in a J2EE environment.

Due to the analogous functionality of servlets, concepts, names and behavior of the portlet will be similar to the ones defined in the *Servlet Specification 2.3* whenever applicable.

PLT.3

Relationship with the Servlet Specification

The *Servlet Specification v2.3* defines servlets as follows:

"A servlet is a Java technology based web component, managed by a container, that generates dynamic content. Like other Java-based components, servlets are platform independent Java classes that are compiled to platform neutral bytecode that can be loaded dynamically into and run by a Java enabled web server. Containers, sometimes called servlet engines, are web server extensions that provide servlet functionality. Servlets interact with web clients via a request/response paradigm implemented by the servlet container."

Portlets share many similarities with servlets:

- Portlets are Java technology based web components
- Portlets are managed by a specialized container
- Portlets generate dynamic content
- Portlets lifecycle is managed by a container
- Portlets interact with web client via a request/response paradigm

Portlets differ in the following aspects from servlets:

- Portlets only generate markup fragments, not complete documents. The Portal aggregates portlet markup fragments into a complete portal page
- Portlets are not directly bound to a URL
- Web clients interact with portlets through a portal system
- Portlets have a more refined request handling, action requests and render requests
- Portlets have predefined portlet modes and window states that indicate the function the portlet is performing and the amount of real state in the portal page
- Portlets can exist many times in a portal page

Portlets have access to the following extra functionality not provided by servlets:

- Portlets have means for accessing and storing persistent configuration and customization data
- Portlets have access to user profile information

- Portlets have URL rewriting functions for creating hyperlinks within their content, which allow portal server agnostic creation of links and actions in page fragments
- Portlets can store transient data in the portlet session in two different scopes: the application-wide scope and the portlet private scope

Portlets do not have access to the following functionality provided by servlets:

- Setting the character set encoding of the response
- Setting HTTP headers on the response
- The URL of the client request to the portal

Because of these differences, the Expert Group has decided that portlets needs to be a new component. Therefore, a portlet is not a servlet. This allows defining a clear interface and behavior for portlets.

In order to reuse as much as possible of the existing servlet infrastructure, the Portlet Specification leverages functionality provided by the Servlet Specification wherever possible. This includes deployment, classloading, web applications, web application lifecycle management, session management and request dispatching. Many concepts and parts of the portlet API have been modeled after the servlet API.

Portlets, servlets and JSPs are bundled in an extended web application called portlet application. Portlets, servlets and JSPs within the same portlet application share classloader, application context and session.

PLT.3.1 Bridging from Portlets to Servlets/JSPs

Portlets can leverage servlets, JSPs and JSP tag-libraries for generating content.

A portlet can call servlets and JSPs just like a servlet can invoke other servlets and JSPs using a request dispatcher (see ### *Dispatching Requests to Servlets and JSPs* Chapter). To enable a seamless integration between portlets and servlets the portlet specification leverages many of the servlet objects.

When a servlet or JSP is called from within a portlet, the servlet request given to the servlet or JSP is based on the portlet request and the servlet response given to the servlet or JSP is based on the portlet response. For example:

- Attributes set in the portlet request are available in the included servlet request (see ### *Dispatching Requests to Servlets and JSPs* Chapter),
- The portlet and the included servlet or JSP share the same output stream (see ### *Dispatching Requests to Servlets and JSPs* Chapter).
- Attributes set in the portlet session are accessible from the servlet session and vice versa (see ### *Portlet Session* Chapter).

PLT.3.2 Relationship Between the Servlet Container and the Portlet Container

The portlet container is an extension of the servlet container. As such, a portlet container can be built on top of an existing servlet container or it may implement all the functionality of a servlet container. Regardless of how a portlet container is implemented, its runtime environment is assumed to support Servlet Specification 2.3.

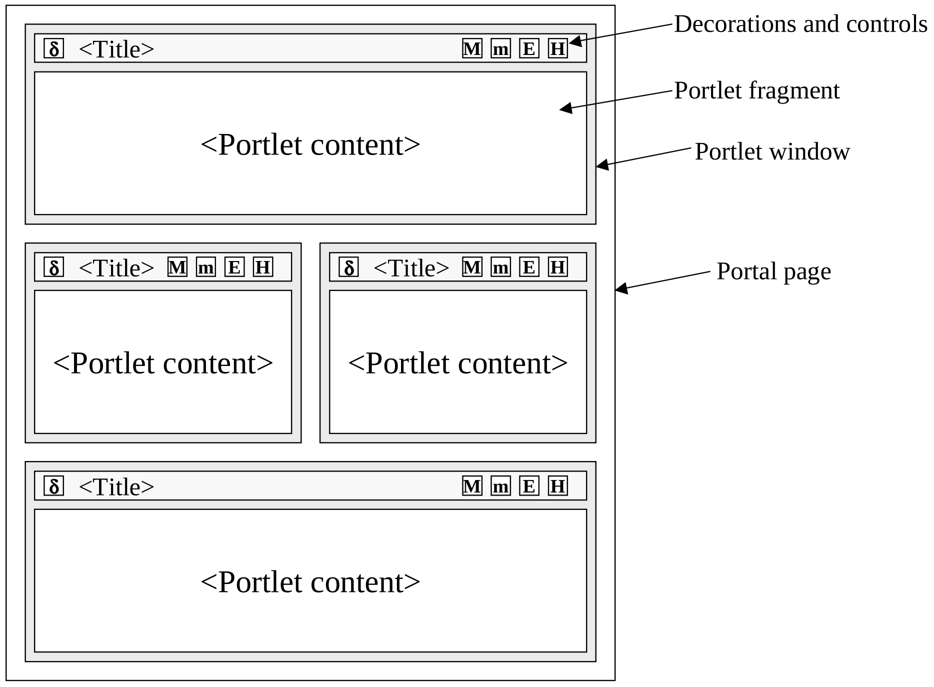
PLT.4

Concepts

PLT.4.1 Elements of a Portal Page

A portlet generates markup fragments. A portal normally adds a title, control buttons and other decorations to the markup fragment generated by the portlet, this new fragment is called a portlet window. Then the portal aggregates portlet windows into a complete document, the portal page.

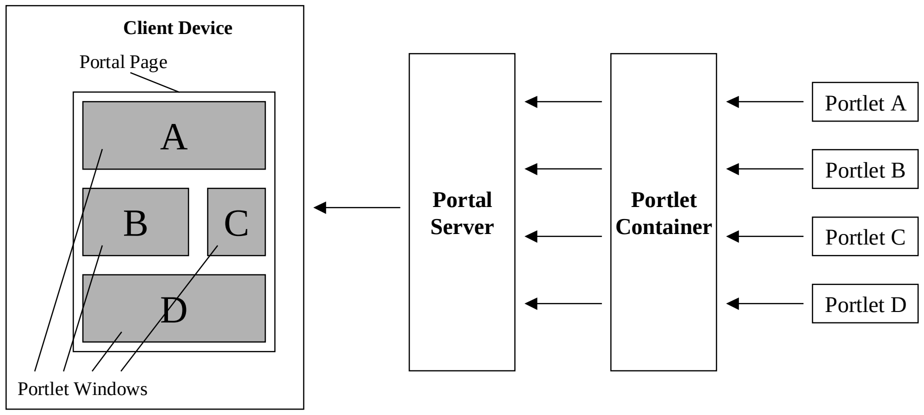
Figure 4-1 Elements of a Portal Page



PLT.4.2 Portal Page Creation

Portlets run within a portlet container. The portlet container receives the content generated by the portlets. Typically, the portlet container hands the portlet content to a portal. The portal server creates the portal page with the content generated by the portlets and sends it to the client device (i.e. a browser) where it is displayed to the user.

FIGURE 4-2 Portal Page Creation



PLT.4.3 Portal Page Request Sequence

Users access a portal by using a client device such as an HTML browser or a web-enabled phone. Upon receiving the request, the portal determines the list of portlets that need to be executed to satisfy the request. The portal, through the portlet container, invokes the portlets. The portal creates the portal page with the fragments generated by the portlets and the page is returned to the client where it is presented to the user.

PLT.5

The Portlet Interface

The `Portlet` interface is the main abstraction of the portlet API. All portlets implement this interface either directly or, more commonly, by extending a class that implements the interface.

The portlet API includes a `GenericPortlet` class that implements the `Portlet` interface and provides default functionality. Developers should extend, directly or indirectly, the `GenericPortlet` class to implement their portlets.

PLT.5.1 Number of Portlet Instances

The portlet definition sections in the deployment descriptor of a portlet application control how the portlet container creates portlet instances.

For a portlet, not hosted in a distributed environment (the default), the portlet container must[i] use only one portlet object per portlet definition.

In the case where a portlet is deployed as part of a portlet application marked as distributable, in the `web.xml` deployment descriptor, a portlet container may instantiate only one portlet object per portlet definition-in the deployment descriptor- per virtual machine (VM). [ii]

PLT.5.2 Portlet Life Cycle

A portlet is managed through a well defined life cycle that defines how it is loaded, instantiated and initialized, how it handles requests from clients, and how it is taken out of service. This life cycle of a portlet is expressed through the `init`, `processAction`, `render` and `destroy` methods of the `Portlet` interface.

PLT.5.2.1 Loading and Instantiation

The portlet container is responsible for loading and instantiating portlets. The loading and instantiation can occur when the portlet container starts the portlet application, or delayed until the portlet container determines the portlet is needed to service a request.

The portlet container must load the portlet class using the same ClassLoader the servlet container uses for the web application part of the portlet application.[iii] After loading the portlet classes, the portlet container instantiates them for use.

PLT.5.2.2 Initialization

After the portlet object is instantiated, the portlet container must initialize the portlet before invoking it to handle requests.[iv] Initialization is provided so that portlets can initialize costly resources (such as backend connections), and perform other one-time activities. The portlet container must initialize the portlet object by calling the init method of the `Portlet` interface with a unique (per portlet definition) object implementing the `PortletConfig` interface. This configuration object provides access to the initialization parameters and the `ResourceBundle` defined in the portlet definition in the deployment descriptor. The configuration object also gives the portlet access to a context object that describes the portlet's runtime environment. Refer to *### Portlet Context* Chapter for information about the `PortletContext` interface.

PLT.5.2.2.1 Error Conditions on Initialization

During initialization, the portlet object may throw an `UnavailableException` or a `PortletException`. In this case, the portlet container must not place the portlet object into active service and it must release the portlet object.[v] The `destroy` method must not be called because the initialization is considered unsuccessful.[vi]

The portlet container may attempt to instantiate and initialize the portlets at any time after a failure. The exception to this rule is when an `UnavailableException` indicates a minimum time of unavailability. When this happens the portlet container must wait for the specified time to pass before creating and initializing a new portlet object.[vii]

A `RuntimeException` thrown during initialization must be handled as a `PortletException`.[viii]

PLT.5.2.2.2 Tools Considerations

The triggering of static initialization methods when a tool loads and introspects a portlet application is to be distinguished from the calling of the `init` method. Developers should not assume that a portlet is in an active portlet container runtime until the `init` method of the `Portlet` interface is called. For example, a portlet should not try to establish connections to databases or Enterprise JavaBeans™ containers when static (class) initialization happens.

PLT.5.2.3 Portlet Window

The portlet definition may include a set of preference attributes with their default values. They are used to create a preferences objects (see ### Portlet Preferences Chapter).

At runtime, when serving requests, a portlet object is associated with a preferences object. Normally, a portlet customizes its behavior and the content it produces based on the attributes of the associated preference object. The portlet may read, modify and add preference attributes.

By default, a preferences object is built using the initial preferences values defined in the portlet deployment descriptor. A portal/portlet-container implementation may provide administrative means to create new preferences objects based on existing ones. Portal/portlet-container created preferences objects may have their attributes further customized.

When a portlet is placed in a portal page, a preferences object is also associated with it. The occurrence of a portlet and preferences-object in a portal page is called a portlet window. The portal/portlet-container implementation manages this association.

A portal page may contain more than one portlet window that references the same portlet and preferences-object.

Administration, management and configuration of preferences objects and creation of portlet windows is left to the portal/portlet-container implementation. It is also left to the implementation to provide advanced features, such as hierarchical management of preferences objects or cascading changes on preference attributes.

PLT.5.2.4 Request Handling

After a portlet object is properly initialized, the portlet container may invoke the portlet to handle client requests.

The `Portlet` interface defines two methods for handling requests, the `processAction` method and the `render` method.

When a portal/portlet-container invokes the `processAction` method of a portlet, the portlet request is referred to as an action request. When a portal/portlet-container invokes the `render` method of a portlet, the portlet request is referred to as a render request.

Commonly, client requests are triggered by URLs created by portlets. These URLs are called portlet URLs. A portlet URL is targeted to a particular portlet. Portlet URLs may be of two types, action URLs or render URLs. Refer to *### `Portlet URLs`* Chapter for details on portlet URLs.

Normally, a client request triggered by an action URL translates into one action request and many render requests, one per portlet in the portal page. A client request triggered by a render URL translates into many render requests, one per portlet in the portal page.

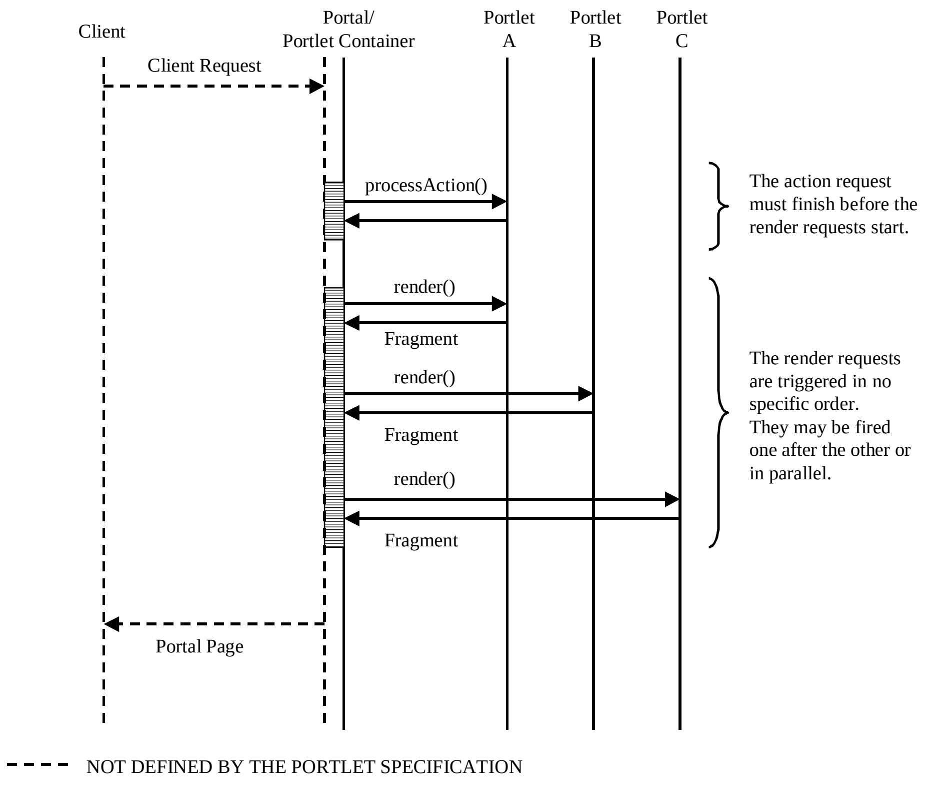
If the client request is triggered by an action URL, the portal/portlet-container must first trigger the action request by invoking the `processAction` method of the targeted portlet.[ix] The portal/portlet-container must wait until the action request finishes. Then, the portal/portlet-container must trigger the render request by invoking the `render` method for all the portlets in the portal page with the possible exception of portlets for which their content is being cached.[x] The render requests may be executed sequentially or in parallel without any guaranteed order.

If the client request is triggered by a render URL, the portal/portlet-container must invoke the `render` method for all the portlets in the portal page with the possible exception of portlets for which their content is being cached.[xi] The portal/portlet-container must not invoke the `processAction` of any of the portlets in the portal page for that client request.

If a portlet has caching enabled, the portal/portlet-container may not invoke the render method. The portal/portlet-container may instead use the portlet's cached content. Refer to *### Caching* Chapter for details on caching.

A portlet object placed into service by a portlet container may end up not handling any request during its lifetime.

Figure 5-1 Request Handling Sequence


Client
Portal/
Portlet Container
Portlet
A
Portlet
B
Portlet
C
Client Request
processAction()
The action request must finish before the render requests start.
render()
Fragment
render()
Fragment
render()
Fragment
The render requests are triggered in no specific order.
They may be fired one after the other or in parallel.
Portal Page
NOT DEFINED BY THE PORTLET SPECIFICATION

PLT.5.2.4.1 Action Request

Typically, in response to an action request, a portlet updates state based on the information sent in the action request parameters.

The `processAction` method of the `Portlet` interface receives two parameters, `ActionRequest` and `ActionResponse`.

The `ActionRequest` object provides access to information such as the parameters of the action request, the window state, the portlet mode, the portal context, the portlet session and the portlet preferences data.

While processing an action request, the portlet may instruct the portal/portlet-container to redirect the user to a specific URL. If the portlet issues a redirection, when the `processAction` method concludes, the portal/portlet-container must send the redirection back to the user agent[xii] and it must finalize the processing of the client request.

A portlet may change its portlet mode and its window state during an action request. This is done, using the `ActionResponse` object. The change of portlet mode or window state must be effective for the following render request the portlet receives.[xiii]

The portlet may also set, in the `ActionResponse` object, render parameters during the processing of an action request. Refer to *### Request Parameters* Section for details on render parameters.

PLT.5.2.4.2 Render Request

Commonly, during a render request, portlets generate content based on their current state.

The `render` method of the `Portlet` interface receives two parameters, `RenderRequest` and `RenderResponse`.

The `RenderRequest` object provides access to information such as the parameters of the render request, the window state, the portlet mode, the portal context, the portlet session and the portlet preferences data.

The portlet can produce content using the `RenderResponse` writer or it may delegate the generation of content to a servlet or a JSP. Refer to *### Dispatching Requests to Servlets and JSPs* Chapter for details on this.

PLT.5.2.4.2.1 GenericPortlet

The `GenericPortlet` abstract class provides default functionality and convenience methods for handling render requests.

The render method in the GenericPortlet class sets the title specified in the portlet definition in the deployment descriptor and invokes the `doDispatch` method.

The `doDispatch` method in the `GenericPortlet` class implements functionality to aid in the processing of requests based on the portlet mode the portlet is currently in (see *### Portlet Modes* Chapter). These methods are:

- `doView` for handling VIEW requests[xiv]
- `doEdit` for handling EDIT requests[xv]
- `doHelp` for handling HELP requests[xvi]

If the window state of the portlet (see *### Window States* Chapter) is `MINIMIZED`, the `render` method of the `GenericPortlet` does not invoke any of the portlet mode rendering methods.[xvii]

Typically, portlets will extend the `GenericPortlet` class directly or indirectly and they will override the `doView`, `doEdit`, `doHelp`, `doCustom` and `getTitle` methods instead of the `render` and `doDispatch` methods.

PLT.5.2.4.3 Multithreading Issues During Request Handling

The portlet container handles concurrent requests to the same portlet by concurrent execution of the request handling methods on different threads. Portlet developers must design their portlets to handle concurrent execution from multiple threads from within the `processAction` and `render` methods at any particular time.

PLT.5.2.4.4 Exceptions During Request Handling

A portlet may throw either a `PortletException`, a `PortletSecurityException` or an `UnavailableException` during the processing of a request.

A `PortletException` signals that an error has occurred during the processing of the request and that the portlet container should take appropriate measures to clean up the request. If a portlet throws an exception in the `processAction` method, all operations on the ActionResponse must be ignored and the render method must not be invoked within the current client request.[xviii] The portal/portlet-container should continue processing the other portlets visible in the portlet page.

A PortletSecurityException indicates that the request has been aborted because the user does not have sufficient rights. Upon receiving a PortletSecurityException, the portlet-container should handle this exception in an appropiate maner.

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 immediately, call the portlet's `destroy` method, and release the portlet object.[xix] A portlet that throws a permanent `UnavailableException` must be considered unavailable until the portlet application containing the portlet is restarted.

When temporary unavailability is indicated by the `UnavailableException`, then the portlet container may choose to not route any requests to 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 object that throws any `UnavailableException` from service.

A `RuntimeException` thrown during the request handling must be handled as a `PortletException`.[xx]

When a portlet throws an exception, or when a portlet becomes unavailable, the portal/portlet-container may include a proper error message in the portal page returned to the user.

PLT.5.2.4.5 Thread Safety

Implementations of the request and response objects are not guaranteed to be thread safe. This means that they must only be used within the scope of the thread invoking the `processAction` and `render` methods.

To remain portable, portlet applications should not give references of the request and response objects to objects executing in other threads as the resulting behavior may be non-deterministic.

PLT.5.2.5 End of Service

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

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

Before the portlet container calls the `destroy` method, it should allow any threads that are currently processing requests within the portlet object to complete execution.To avoid waiting forever, the portlet container can optionally wait for a predefined time before destroying the portlet object.

Once the `destroy` method is called on a portlet object, the portlet container must not route any requests to that portlet object.[xxi] If the portlet container needs to enable the portlet again, it must do so with a new portlet object, which is a new instance of the portlet's class.[xxii]

If the portlet object throws a `RuntimeException` within the execution of the `destroy` method the portlet container must consider the portlet object successfully destroyed.[xxiii]

After the `destroy` method completes, the portlet container must release the portlet object so that it is eligible for garbage collection.[xxiv] Portlet implementations should not use finalizers.

PLT.6

Portlet Config

The `PortletConfig` object provides the portlet object with information to be used during initialization. It also provides access to the portlet context and the resource bundle that provides title-bar resources.

PLT.6.1 Initialization Parameters

The `getInitParameterNames` and `getInitParameter` methods of the `PortletConfig` interface return the initialization parameter names and values found in the portlet definition in the deployment descriptor.

PLT.6.2 Portlet Resource Bundle

Portlets must specify, in their deployment descriptor definition, some basic information that can be used for the portlet title-bar and for the portal's categorization of the portlet. The specification defines a few resource elements for these purposes, title, short-title, description and keywords (see the *### Resource Bundles* Section).

These resources can be defined in one of two ways in the deployment descriptor. They can be directly included in the portlet definition in the deployment descriptor, or they can be placed in a resource bundle with a reference to the resource bundle in the deployment portlet definition in the descriptor.

If the resources are included inline the deployment descriptor, the portlet container must use this information for all Locales.[xxv]. An example of a deployment descriptor defining portlet information inline could be:

```
<portlet>
  ...
  <portlet-info>
    ~~<locale>en</locale>~~
    <title>Stock Quote Portlet</title>
    <short-title>Stock</short-title>
    <keywords>finance,stock market</keywords>
  </portlet-info>
  ...
</portlet>
```

If the resources are in a resource bundle, the portlet must provide the name of the resource bundle containing the localized values. An example of a deployment descriptor defining portlet information in resource bundles could be:

```
<portlet>
  ...
  <portlet-info>
    <resource-bundle>QuotePortlet</resource-bundle>
  </portlet-info>
  ...
</portlet>
```

Regardless of what mechanism is used for providing this information, the portlet access this information using the `getResourceBundle` method of the `PortletConfig` interface. If the information is defined inline in the deployment descriptor, the portlet container must create a ResourceBundle and populate it, with the inline values, using the keys defined in the *### Resource Bundles* Section.[xxvi]

The `render` method of the `GenericPortlet` uses the `ResourceBundle` object of the `PortletConfig` to retrieve the title of the portlet from the portlet definition.

PLT.7

Portlet URLs

As part of its content, a portlet may need to create URLs that reference the portlet itself. For example, when a user acts on a URL that references a portlet (i.e., by clicking a link or submitting a form) the result is a new client request to the portal targeted to the portlet. Those URLs are called portlet URLs.

PLT.7.1 PortletURL

The portlet API defines the `PortletURL` interface. Portlets must create portlet URLs using `PortletURL` objects. A portlet creates `PortletURL` objects invoking the `createActionURL` and the `createRenderURL` methods of the `RenderResponse` interface. The `createActionURL` method creates action URLs. The `createRenderURL` method creates render URLs.

Portlet developers should be aware that the use of forms using the method GET may lead to non-deterministic behavior in some portal/portlet-containers implementations that encode internal state as part of the URL query string.

A render URL is an optimization for a special type of action URLs. The portal/portlet-container must not invoke the processAction method of the targeted portlet.[xxvii] The portal/portlet-container must ensure that all the parameters set when constructing the render URL become parameters of the subsequent render requests for the portlet.[xxviii]

Render URLs should not be used for tasks that are not idempotent from the portlet perspective. Error conditions, cache expirations and changes of external data may affect the content generated by a portlet as result of a request triggered by a render URL. Render URLs should not be used within forms as the portal/portlet-container may ignore form parameters.

Portlets can add application specific parameters to the `PortletURL` objects using the `addParameter` method. All the parameters a portlet adds to a `PortletURL` object must be made available to the portlet as request parameters.[xxix] If a portal/portlet-container encodes additional information as parameters, it must encode them properly to avoid collisions with the parameters set and used by the portlet.[xxx]

Using the `toString` method, a portlet can obtain the string representation of the `PortletURL` for its inclusion in the portlet content.

An example of creating a portlet URI would be:

```
        ...
        PortletURL url = response.createRenderURL();
        url.addParameter(“customer”,”foo.com”);
        url.addParameter(“show”,”summary”);
        writer.print(“<A HREF=”+url.toString()+”>Summary</A>”);
        ...
```

Portlet developers should be aware that the string representation of a PortletURL may not be a well formed URL but special tokens at the time the portlet is generating its content. Portal servers often use a technique called URL rewriting that post-processes the content resolving tokens into real URLs.

PLT.7.1.1 Including a Portlet Mode or a Window State

A portlet URL can include a specific portlet mode (see *### Portlet Modes* Chapter) or window state (see *### Window States* Chapter). If a portlet URL containing a portlet mode or a window state is requested by the user, the portal/portlet-container must invoke the portlet using the specified portlet mode and window state. The `PortletURL` interface has the `setWindowState` and `setPortletMode` methods for setting the portlet mode and window state in the portlet URL. For example:

```
        ...
        PortletURL url = response.createActionURL();
        url.addParameter(“paymentMethod”,”creditCardInProfile”);
        url.setWindowState(WindowState.MAXIMIZED);
        writer.print(“<FORM METHOD=POST ACTION=”+ url.toString()+”>”);
        ...
```

A portlet cannot create a portlet URL using a portlet mode that is not defined as supported by the portlet or that the user it is not allowed to use. The `setPortletMode` methods must throw a `PortletModeException` in that situation.[xxxi].

A portlet cannot create a portlet URL using a window state that is not supported by the portlet container. The `setWindowState` method must throw a `WindowStateException` if that is the case.[xxxii]

PLT.7.1.2 Portlet URL security

The `setSecure` method of the `PortletURL` interface allows a portlet to indicate if the portlet URL has to be a secure URL or not (i.e. HTTPS or HTTP). If the `setSecure` method is not used, the portlet URL must be of the same security level of the current request.[xxxiii]

PLT.8

Portlet Modes

A portlet mode indicates the function a portlet is performing. Normally, portlets perform different tasks and create different content depending on the function they are currently performing. A portlet mode advises the portlet what task it should perform and what content it should generate. When invoking a portlet, the portlet container provides the current portlet mode to the portlet. Portlets can programmatically change their portlet mode when processing an action request.

The Portlet Specification defines three portlet modes, `VIEW`, `EDIT`, and `HELP`. The `PortletMode` class defines constants for these portlet modes.

The availability of the portlet modes, for a portlet, may be restricted to specific user roles by the portal. For example, anonymous users could be allowed to use the `VIEW` and `HELP` portlet modes but only authenticated users could use the `EDIT` portlet mode.

PLT.8.1 `VIEW` Portlet Mode

The expected functionality for a portlet in `VIEW` portlet mode is to generate markup reflecting the current state of the portlet. For example, the `VIEW` portlet mode of a portlet may include one or more screens that the user can navigate and interact with, or it may consist of static content that does not require any user interaction.

Portlet developers should implement the `VIEW` portlet mode functionality by overriding the `doView` method of the `GenericPortlet` class.

Portlets must support the `VIEW` portlet mode.

PLT.8.2 `EDIT` Portlet Mode

Within the `EDIT` portlet mode, a portlet should provide content and logic that lets a user customize the behavior of the portlet. The `EDIT` portlet mode may include one or more screens among which users can navigate to enter their customization data.

Typically, portlets in `EDIT` portlet mode will set or update portlet preferences. Refer to *### Portlet Preferences* Chapter for details on portlet preferences.

Portlet developers should implement the EDIT portlet mode functionality by overriding the `doEdit` method of the `GenericPortlet` class.

PLT.8.3 HELP Portlet Mode

When in `HELP` portlet mode, a portlet should provide help information about the portlet. This help information could be a simple help screen explaining the entire portlet in coherent text or it could be context-sensitive help.

Portlet developers should implement the HELP portlet mode functionality by overriding the `doHelp` method of the `GenericPortlet` class.

PLT.8.4 Custom Portlet Modes

Portal vendors may define custom portlet modes for vendor specific functionality.

Portlets can only use portlet modes that are defined by the portal. Portlets must define the custom portlet modes they intend to use in the deployment descriptor using the `custom-portlet-mode` element. At deployment time, the custom portlet modes defined in the deployment descriptors should be mapped to custom portlet modes supported by the portal implementation.

If a custom portlet mode defined in the deployment descriptor is not mapped to a custom portlet mode provided by the portal, portlets must not be invoked in that portlet mode.

For example, the deployment descriptor for a portlet application containing portlets that support clipboard and config custom portlet modes would have the following definition:

```
<portlet-app>
  ...
  <custom-portlet-mode>
    <description>Creates content for Cut and Paste</description>
    <name>clipboard</name>
  </custom-portlet-mode>

  <custom-portlet-mode>
    <description>Provides administration functions</description>
    <name>config</name>
  </custom-portlet-mode>
  ...
</portlet-app>
```

The *### Extended Portlet Modes* appendix defines a list of portlet mode names and their suggested utilization. Portals implementing these predefined custom portlet modes could do an automatic mapping when custom portlet modes with those names are defined in the deployment descriptor.

PLT.8.5 GenericPortlet Render Handling

The `GenericPortlet` class implementation of the `render` method dispatches requests to the `doView`, `doEdit` or `doHelp` method depending on the portlet mode indicated in the request using the `doDispatch` method.[xxxiv] If the portlet provides support for custom portlet modes, the portlet should override the `doDispatch` method of the `GenericPortlet`.

PLT.8.6 Defining Portlet Modes Support

Portlets must describe within their definition, in the deployment descriptor, the portlet modes they can handle for each markup type they support. As all portlets must support the `VIEW` portlet mode, `VIEW` does not have to be indicated.[xxxv] The portlet-container must not invoke a portlet in a portlet mode that has not been declared as supported for a given markup type.[xxxvi]

The following example shows a snippet of the portlet modes a portlet defines as supporting in its deployment descriptor definition:

```
...
<supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>edit</portlet-mode>
    <portlet-mode>help</portlet-mode>
    ...
</supports>
<supports>
    <mime-type>text/vnd.wap.wml</mime-type>
    <portlet-mode>help</portlet-mode>
    ...
</supports>
...
```

For HTML markup, this portlet supports the `EDIT` and `HELP` portlet modes in addition to the required `VIEW` portlet mode. For WML markup, it supports the `VIEW` and `HELP` portlet modes.

The portlet container must ignore all references to custom portlet modes that are not supported by the portal implementation, or that have no mapping to portlet modes supported by the portal.[xxxvii]

PLT.9

Window States

A window state is an indicator of the amount of portal page space that will be assigned to the content generated by a portlet. When invoking a portlet, the portlet-container provides the current window state to the portlet. The portlet may use the window state to decide how much information it should render. Portlets can programmatically change their window state when processing an action request.

The Portlet Specification defines three window states, `NORMAL`, `MAXIMIZED` and `MINIMIZED`. The `WindowState` class defines constants for these window states.

PLT.9.1 `NORMAL` Window State

The `NORMAL` window state indicates that a portlet may be sharing the page with other portlets. It may also indicate that the target device has limited display capabilities. Therefore, a portlet should restrict the size of its rendered output in this window state.

PLT.9.2 `MAXIMIZED` Window State

The `MAXIMIZED` window state is an indication that a portlet may be the only portlet being rendered in the portal page, or that the portlet has more space compared to other portlets in the portal page. A portlet may generate richer content when its window state is `MAXIMIZED`.

PLT.9.3 `MINIMIZED` Window State

When a portlet is in `MINIMIZED` window state, the portlet should only render minimal output or no output at all.

PLT.9.4 Custom Window States

Portal vendors may define custom window states.

Portlets can only use window states that are defined by the portal. Portlets must define the custom window states they intend to use in the deployment descriptor using the `custom-window-state` element. At deployment time, the custom window states defined in the deployment descriptors should be mapped to custom window states supported by the portal implementation.

If a custom window state defined in the deployment descriptor is not mapped to a custom window state provided by the portal, portlets must not be invoked in that window state.[xxxviii]

For example, the deployment descriptor for a portlet application containing portlets that use a custom `half_page` window state would have the following definition:

```
<portlet-app>
  ...
  <custom-window-state>
    <description>Occupies 50% of the portal page</description>
    <name>half_page</name>
  </custom-window-state>
  ...
</portlet-app>
```

PLT.10

Portlet Context

The `PortletContext` interface defines a portlet's view of the portlet application within which the portlet is running. Using the `PortletContext` object, a portlet can log events, obtain portlet application resources, and set and store attributes that other portlets and servlets in the portlet application can access.

PLT.10.1 Scope of the Portlet Context

There is one instance of the `PortletContext` interface associated with each portlet application deployed into a portlet container.[xxxix] In cases where the container is distributed over many virtual machines, a portlet application will have an instance of the `PortletContext` interface for each VM.[xl]

PLT.10.2 Portlet Context functionality

Through the PortletContext interface, it is possible to access context initialization parameters, retrieve and store context attributes, obtain static resources from the portlet application and obtain a request dispatcher to include servlets and JSPs.

PLT.10.3 Relationship with the Servlet Context

A portlet application is an extended web application. As a web application, a portlet application also has a servlet context. The portlet context leverages most of its functionality from the servlet context of the portlet application.

The initialization parameters are the same initialization parameters of the servlet context and the context attributes are shared with the servlet context. Therefore, they must be defined in the web application deployment descriptor (the `web.xml` file). The initialization parameters accessible through the `PortletContext` must be same that are accessible through the `ServletContext` of the portlet application.[xli]

Context attributes set using the `PortletContext` must be stored in the `ServletContext` of the portlet application. A direct consequence of this is that data stored in the `ServletContext` by servlets or JSPs is accessible to portlets through the `PortletContext` and vice versa.[xlii]

The `PortletContext` must offer access to the same set of resources the `ServletContext` exposes.[xliii]

The PortletContext must handle the same temporary working directory the ServletContext handles. It must be accessible as a context attribute using the same constant defined in the *Servlet Specification 2.3 SVR 3 Servlet Context* Chapter, `javax.servlet.context.tempdir`.[xliv] The portlet context must follow the same behavior and functionality that the servlet context has for virtual hosting and reloading considerations. (see *Servlet Specification 2.3 SVR 3 Servlet Context* Chapter)[xlv]:

PLT.10.3.1 Correspondence between ServletContext and PortletContext methods

The following methods of the `PortletContext` should be based on the methods of the `ServletContext` of similar name: `getAttribute`, `getAttributeNames`, `getInitParameter`, `getInitParameterNames`, `getMimeType`, `getRealPath`, `getResource`, `getResourcePaths`, `getResourceAsStream`, `log`, `removeAttribute` and `setAttribute`.

PLT.11

Portlet Requests

The request objects encapsulate all information about the client request, parameters, request content data, portlet mode, window state, etc. A request object is passed to `processAction` and `render` methods of the portlet.

PLT.11.1 PortletRequest Interface

The `PortletRequest` interface defines the common functionality for the `ActionRequest` and `RenderRequest` interfaces.

PLT.11.1.1 Request Parameters

If a portlet receives a request from a client request targeted to the portlet itself, the parameters must be the string parameters sent by the client to the portlet as part of the client request.[xlvi]

The portlet-container must not propagate parameters received in an action request to subsequent render requests of the portlet.[xlvii] If a portlet wants to do that, it can use render URLs or it must use the `setRenderParameter` or `setRenderParameters` methods of the `ActionResponse` object within the `processAction` call.

If a portlet receives a render request that is the result of a client request targeted to another portlet in the portal page, the parameters must be the same parameters of the previous render request.[xlviii]

If a portlet receives a render request following an action request as part of the same client request, the parameters received with render request must be the render parameters set during the action request.[xlix]

Commonly, portals provide controls to change the portlet mode and the window state of portlets. The URLs these controls use are generated by the portal. Client requests triggered by those URLs must be treated as render URLs and the existing render parameters must be preserved.[l]

A portlet must not see any parameter targeted to other portlets.[li]

The parameters are stored as a set of name-value pairs. Multiple parameter values can exist for any given parameter name. The following methods of the `PortletRequest` interface are available to access parameters:

- `getParameter`
- `getParameterNames`
- `getParameterValues`
- `getParameterMap`

The `getParameterValues` method returns an array of `String` objects containing all the parameter values associated with a parameter name. The value returned from the `getParameter` method must be the first value in the array of `String` objects returned by `getParameterValues` [lii]. If there is a single parameter value associated with a parameter name the method returns must return an array of size one containing the parameter value.[liii]. The `getParameterMap` method must return an unmodifiable `Map` object. If the request does not have any parameter, the `getParameterMap` must return an empty `Map` object.

PLT.11.1.2 Extra Request Parameters

The portal/portlet-container implementation may add extra parameters to portlet URLs to help the portal/portlet-container route and process client requests.

Extra parameters used by the portal/portlet-container must be invisible to the portlets receiving the request. [liv]

It is the responsibility of the portal/portlet-container to properly encode these extra parameters to avoid name collisions with parameters the portlets define.

PLT.11.1.3 Request Attributes

Request attributes are objects associated with a portlet during a single request. Request Attributes may be set by the portlet or the portlet container to express information that otherwise could not be expressed via the API. . Request attributes can be used to share information with a servlet or JSP being included via the `PortletRequestDispatcher`.

Attributes are set, obtained and removed using the following methods of the `PortletRequest` interface:

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

Only one attribute value may be associated with an attribute name.

Attribute names beginning with the `"javax.portlet."` prefix are reserved for definition by this specification. It is suggested that all attributes placed into the attribute set be named in accordance with the reverse domain name convention suggested by the *Java Programming Language Specification 1* for package naming.

PLT.11.1.4 Request Properties

A portlet can access portal/portlet-container specific properties and, if available, the headers of the HTTP client request through the following methods of the methods of the `PortletRequest` interface:

- `getProperty`
- `getProperties`
- `getPropertyNames`

There can be multiple properties with the same name. If there are multiple properties with the same name, the `getProperty` method returns the first property value. The `getProperties` method allows access to all the property values associated with a particular property name, returning an `Enumeration` of `String` objects.

Depending on the underlying web-server/servlet-container and the portal/portlet-container implementation, client request HTTP headers may not be always available. Portlets should not rely on the presence of headers to function properly. The `PortletRequest` interface provides specific methods to access information normally available as HTTP headers: character-encoding, content-length, content-type, locale. Portlets should use the specific methods for retrieving those values as the portal/portlet-container implementation may use other means to determine that information.

PLT.11.1.5 Request Context Path

The context path of a request is exposed via the request object. The context path is the path prefix associated with the portlet context that this portlet is a part of. If this context is the "default" context rooted at the base of the web server URL namespace, this path will be an empty string.[lv] Otherwise, if the context is not rooted at the root of the server's namespace, the path starts with a'/' character but does not end with a'/' character.[lvi]

PLT.11.1.6 Security Attributes

The PortletRequest interface offers a set of methods that provide security information about the user and the connection between the user and the portal. These methods are:

- `getAuthType`
- `getRemoteUser`
- `getUserPrincipal`
- `isUserInRole`
- `isSecure`

The getAuthType indicates the authentication scheme being used between the user and the portal. It may return one of the defined constants (BASIC_AUTH, DIGEST_AUTH, CERT_AUTH and FORM_AUTH) or another String value that represents a vendor provided authentication type. If the user is not authenticated the getAuthType method must return null.[lvii]

The getRemoteUser method returns the login name of the user making this request.

The getUserPrincipal method returns a java.security.Principal object containing the name of the authenticated user.

The isUserInRole method indicates if an authenticated user is included in the specified logical role.

The isSecure method indicates if the request has been transmitted over a secure protocol such as HTTPS.

PLT.11.1.7 Response Content Types

Portlet developers may code portlets to support multiple content types. A portlet can obtain, using the getResponseContentType method of the request object, a string representing the default content type the portlet container assumes for the output.

If the portlet container supports additional content types for the portlet's output, it must declare the additional content types through the getResponseContentTypes method of the request object. The returned Enumeration of strings should contain the content types the portlet container supports in order of preference. The first element of the enumeration must be the same content type returned by the getResponseContentType method.[lviii]

If a portlet defines support for all content types using a wildcard and the portlet container supports all content types, the getContentType may return the wildcard or the portlet container preferred content type.

PLT.11.1.8 Internationalization

The portal/portlet-container decides what locale will be used for creating the response for a user. The portal/portlet-container may use information that the client sends with the request. For example the Accept-Language header along with other mechanisms described in the HTTP/ 1.1 specification. The getLocale method is provided in the PortletRequest interface to inform the portlet about the locale of user the portal/portlet-container has chosen.

PLT.11.1.9 Portlet Mode

The `getPortletMode` and `getPreviousPortletMode` methods of the `PortletRequest` interface allow a portlet to find out its current and previous portlet mode. The `getPreviousPortletMode` method returns the portlet mode the portlet was in before the current portlet mode.[lix] If there is not previous portlet mode, the `getPreviousPortletMode` must return `null`.[lx]

A portlet may be restricted to work with a subset of the portlet modes supported by the portal/portlet-container. A portlet can use the `isPortletModeAllowed` method of the `PortletRequest` interface to find out if the portlet is allowed to use a portlet mode. A portlet mode is not allowed if the portlet mode is not in the portlet definition or portlet or user has been constrained further.

PLT.11.1.10 Window State

The `getWindowState` and `getPreviousWindowState` methods of the `PortletRequest` interface allows a portlet to find out its current and previous window state. The `getPreviousWindowState` method returns the window state the portlet was in before the current window state.[lxi] If there is not previous window state, the `getPreviousWindowState` must return `null`.[lxii]

A portlet may be restricted to work with a subset of the window states supported by the portal/portlet-container. A portlet can use the `isWindowStateAllowed` method of the `PortletRequest` interface to find out if the portlet is allowed to use a window state.

PLT.11.2 ActionRequest Interface

The `ActionRequest` interface extends the `PortletRequest` interface and is used in the `processAction` method of the `Portlet` interface. In addition to the functionality provided by the `PortletRequest` interface, the `ActionRequest` interface gives access to the input stream of the request.

PLT.11.2.1 Retrieving Uploaded Data

The input stream is useful when the client request contains HTTP POST data of type other than `application/x-www-form-urlencoded.` For example, when a file is uploaded to the portlet as part of a user interaction.

As a convenience to the portlet developer, the Action`Request` interface also provides a `getReader` method that retrieves the HTTP POST data as character data according to the character encoding defined in the user request.

Only one of the two methods, `getPortletInputStream` or `getReader`, can be used during an action request. If the input stream is obtained, a call to the `getReader` must throw an `IllegalStateExcepion`. Similarly, if the reader is obtained, a call to the `getPortletInputStream` must throw an `IllegalStateException`.[lxiii]

To help manage the input stream, the `ActionRequest` interface also provides the following methods:

- `getContentType`
- `getCharacterEncoding`
- `getContentLength`

If the user request HTTP POST data is of type `application/x-www-form-urlencoded`, this data has been already processed by the portal/portlet-container and is available as request parameters. The `getPortletInputStream` and `getReader` methods must throw an `IllegalStateException` if called.[lxiv]

PLT.11.3 RenderRequest Interface

The `RenderRequest` interface extends the `PortletRequest` interface and is used in the `render` method of the `Portlet` interface.

PLT.11.4 Lifetime of the Request Objects

Each request object is valid only within the scope of a particular `processAction` or `render` method call. Containers commonly recycle request objects in order to avoid the performance overhead of request object creation. The developer must be aware that maintaining references to request objects outside the scope described above may lead to non-deterministic behavior.

PLT.12

Portlet Responses

The response objects encapsulate all information to be returned from the portlet to the portlet container during a request: a redirection, a portlet mode change, title, content, etc. The portal/portlet-container will use this information to construct the response –usually a portal page- to be returned to the client. A response object is passed to `processAction` and `render` methods of the portlet.

PLT.12.1 PortletResponse Interface

The `PortletResponse` interface defines the common functionality for the `ActionResponse` and `RenderResponse` interfaces.

PLT.12.1.1 Response Properties

Properties can be used by portlets to send vendor specific information to the portal/portlet-container.

A portlet can set properties using the following methods of the `PortletResponse` interface:

- `setProperty`
- `addProperty`

The `setProperty` method sets a property with a given name and value. A previous property is replaced by the new property. Where a set of property values exist for the name, the values are cleared and replaced with the new value. The `addProperty` method adds a property value to the set with a given name. If there are no property values already associated with the name, a new set is created.

PLT.12.1.2 URLs encoding

Portlets may generate content with URLs referring to other resources within the portal, such as servlets, JSPs, images and other static files. Some portal/portlet-container implementation may require those URLs to contain implementation specific data encoded in it. Because of that, portlets should use the `encodeURL` method to create such URLs. The `encodeURL` method may include the session ID and other portal/portlet-container specific information into the URL. If encoding is not needed, it returns the URL unchanged.

~~Namespace encoding~~

~~Within their content, portlets may include elements that must be unique within the whole portal page. JavaScript functions and variables are an example of this.~~

~~The `encodeNamespace` method provides the portlet with a mechanism that ensures the uniqueness of the returned string in the whole portal page. For example, the `encodeNamespace` method could append a unique ID to the received string.~~

~~If the string passed to the `encodeNamespace` method is a valid identifier as defined in the *3.8 Identifier* Section of the *Java Language Specification Second Edition*, the returned string must also be a valid identifier.~~~~[lxv]~~

PLT.12.2 ActionResponse Interface

The `ActionResponse` interface extends the `PortletResponse` interface and it is used in the `processAction` method of the `Portlet` interface. This interface allows a portlet to redirect the user to another URL, set render parameters, change the window state of the portlet and change the portlet mode of the portlet.

PLT.12.2.1 Redirections

The `sendRedirect` method instructs the portal/portlet-container to set the appropriate headers and content body to redirect the user to a different URL. A fully qualified URL or a full path URL must be specified. If a relative path URL is given, an `IllegalArgumentException` must be thrown.[lxvi]

If the `sendRedirect` method is called after the `setPortletMode`, `setWindowState`, `setRenderParameter` or `setRenderParameters` methods of the `ActionResponse` interface, an `IllegalStateException` must be thrown and the redirection must not be executed.[lxvii]

PLT.12.2.2 Portlet Modes and Window State Changes

The `setPortletMode` method allows a portlet to change its current portlet mode. The new portlet mode will be effective in the following render request. If a portlet attempts to set a portlet mode that is not allowed to switch to, a `PortletModeException` must be thrown.[lxviii]

The `setWindowState` method allows a portlet to change its current window state. The new window state will be effective in the following render request. If a portlet attempts to set a window state that it is not allowed to switch to, a `WindowStateException` must be thrown.[lxix]

If the `setPortletMode` or `setWindowState` methods are called after the `sendRedirect` method has been called and `IllegalStateException` must be thrown.[lxx] If the exception is caught by the portlet, the redirection must be executed.[lxxi] If the exception is propagated back to the portlet-container, the redirection must not be executed.[lxxii]

PLT.12.2.3 Render Parameters

Using the `setRenderParameter` and `setRenderParameters` methods of the `ActionResponse` interface portlets may set render parameters during an action request. These parameters will be used in all subsequent render requests untils a new client request targets the portlet. If no render parameters are set during the `processAction` invocation, the render request must not contain any request parameters.[lxxiii]

PLT.12.3 RenderResponse Interface

The `RenderResponse` interface extends the `PortletResponse` interface and it is used in the `render` method of the `Portlet` interface. This interface allows a portlet to set its title and generate content.

PLT.12.3.1 Content Type

The `getContentType` method of the `RenderResponse` interface returns the portal/portlet-container default content type.

A portlet may change the content type of the response using the `setContentType` method of the `RenderResponse` interface.~~If the default response content type for the portlet is not a wildcard and the portlet does not set a~~~~If the content type is not set~~ ~~in the response object, the portlet container must assume the default response content type returned by the request object.~~[lxxiv]

~~If the default content type returned by the request object is a wildcard, the portlet must set a concrete content type in the response object, if this is not done the portlet-container must throw an `IllegalStateException` when the `getWriter` or `getOutputStream` methods are called.~~

~~If a supported response content type (other than the default response content type) is set by the portlet, the portal/portlet-container has the responsibility of performing any necessary content type conversion to generate the portal page using the portal page content type.~~

~~If a portal/portlet-container chooses not to do content type conversions on a portlet basis, the `getResponseContentTypes` method of the request object should always return a single value: the content type that will be returned to the user.~~

The `setContentType` method must throw an `IllegalArgumentException` if the content type set does not match (including wildcard matching) any of ~~is not one~~ the content types returned~~declared~~ by the `getResponseContentType` method of the `PortleRequest` object[lxxv]. The portlet container must ignore any character encoding specified as part of the content type.[lxxvi]

The `setContentType` method must be called before the `getWriter` or `getOutputStream` methods, if called after it should be ignored.

If the portlet has set a content type, the getContentType method must return it. If the portlet has not set any content type, the getContentType method returns default content type the portal/portlet-container will use for the response. The default content type must not be a wildcard and it must match (including wildcard matching) one of the content types declared in the portlet definition.

PLT.12.3.2 Output Stream and Writer Objects

A portlet may generate its content by writing to the `OutputStream` or to the `Writer` of the `RenderResponse` object.

The raw OutputStream is available because of some servlet container implementations requirements and for portlets that do not generate markup fragments. If a portlet utilizes the `OutputStream`, the portlet is responsible of using the proper character encoding.A portlet must use only one of these objects. The portlet container must throw an `IllegalStateException` if a portlet attempts to use both.[lxxvii]

The termination of the `render` method of the portlet indicates that the portlet has satisfied the request and that the output object is to be closed.

PLT.12.3.3 Buffering

A portlet container is allowed, but not required, to buffer output going to the client for efficiency purposes. Typically servers that do buffering make it the default, but allow portlets to specify buffering parameters.

The following methods in the `RenderResponse` interface allow a portlet to access and set buffering information:

- `getBufferSize`
- `setBufferSize`
- `isCommitted`
- `reset`
- `resetBuffer`
- `flushBuffer`

These methods are provided on the `RenderResponse` interface to allow buffering operations to be performed whether the portlet is using an `OutputStream` or a `Writer`.

The `getBufferSize` method returns the size of the underlying buffer being used. If no buffering is being used, this method must return the int value of `0` (zero).[lxxviii]

The portlet can request a preferred buffer size by using the `setBufferSize` method. The buffer assigned is not required to be the size requested by the portlet, but must be at least as large as the size requested.[lxxix] This allows the container to reuse a set of fixed size buffers, providing a larger buffer than requested if appropriate. The method must be called before any content is written using a `OutputStream` or `Writer`. If any content has been written, this method must throw an `IllegalStateException`.[lxxx]

The `isCommitted` method returns a `boolean` value indicating whether any response bytes have been returned to the client. The `flushBuffer` method forces content in the buffer to be written to the client.

The `reset` method clears data in the buffer when the response is not committed. Properties set by the portlet prior to the reset call must be cleared as well.[lxxxi] The `resetBuffer` method clears content in the buffer if the response is not committed without clearing the properties.

If the response is committed and the reset or resetBuffer method is called, an `IllegalStateException` must be thrown.[lxxxii] The response and its associated buffer must be unchanged.[lxxxiii]

When using a buffer, the container must immediately flush the contents of a filled buffer to the client.[lxxxiv] If this is the first data is sent to the client, the response must be considered as committed.

PLT.12.3.4 Namespace encoding

Within their content, portlets may include elements that must be unique within the whole portal page. JavaScript functions and variables are an example of this.

The `encodeNamespace` method provides the portlet with a mechanism that ensures the uniqueness of the returned string in the whole portal page. For example, the `encodeNamespace` method could append a unique ID to the received string.

If the string passed to the `encodeNamespace` method is a valid identifier as defined in the *3.8 Identifier* Section of the *Java Language Specification Second Edition*, the returned string must also be a valid identifier.[lxxxv]

PLT.12.3.5 Portlet Title

A portlet may indicate to the portal/portlet-container its preferred title. It is up to the portal/portlet-container to use the preferred title set by the portlet.

The `setTitle` method must be called before the output of the portlet has been commited, if called after it should be ignored.[lxxxvi]

PLT.12.4 Lifetime of Response Objects

Each response object is valid only within the scope of a particular `processAction` or `render` method call. Containers commonly recycle request objects in order to avoid the performance overhead of response object creation. The developer must be aware that maintaining references to response objects outside the scope described above may lead to non-deterministic behavior.

PLT.13 Portal Context

The `PortalContext` interface provides information about the portal that is invoking the portlet.

The `getPortalInfo` method returns information such as the portal vendor and portal version. The returned string should start with the vendor and version information, `'vendorname.majorversion.minorversion.'`.

The `getProperty` and `getPropertyNames` methods return portal properties.

The `getSupportedPortletModes` method returns the portlet modes supported by the portal.

The `getSupportedWindowState` method returns the window states supported by the portal.

A portlet obtains a `PortalContext` object from the request object using `getPortalContext` method.

PLT.14

Portlet Preferences

Portlets are commonly configured to provide a customized view or behavior for different users. This configuration is represented as a persistent set of name-value pairs and it is referred as portlet preferences. The portlet container is responsible for the details of retrieving and storing these preferences.

Portlet preferences are intended to store basic configuration data for portlets. It is not the purpose of the portlet preferences to replace general purpose databases.

PLT.14.1 PortletPreferences Interface

Portlets have access to their preferences attributes through the `PortletPreferences` interface. Portlets have access to the associated `PortletPreferences` object when it is processing requests. A portlet may only modify preferences attributes during a `processAction` invocation.

Preference attributes are `String` array objects.

To access and manipulate preference attributes, the `PortletPreferences` interface provides the following methods:

- `getNames`
- `getValue`
- `setValue`
- `getValues`
- `setValues`
- `isModifiable`
- `reset`
- `store`

The `getValue` and `setValue methods` are convenience methods for dealing with single values. If a preference attribute has multiple values, the `getValue` method returns the first value. The `setValue` method sets a single value into a preferences attribute.

The following code sample demonstrates how a stock quote portlet would retrieve from its preferences object, the preferred stock symbols, the URL of the backend quoting services and the quote refresh frequency.

```
PortletPreferences prefs = req.getPreferences();
String[] symbols =
    prefs.getValues("preferredStockSymbols",
                    new String[]{"ACME","FOO"});
String url = prefs.getValue("quotesFeedURL",null);
int refreshInterval =
    Integer.parseInt(prefs.getValue("refresh","10"));
```

The `reset` method must reset a preference attribute to its default value. If there is no default value, the preference attribute must be deleted.[lxxxvii] It is left to the vendor to specify how and from where the default value is obtained.

If a preference attribute is not modifiable, the `setValue`, `setValues` and `reset` methods must throw an `UnmodifiableException when the portlet is in any of the standard modes.`[lxxxviii]

The `store` method must persist all the changes made to the `PortletPreferences` object in the persistent store.[lxxxix] If the call returns successfully, it is safe to assume the changes are permanent. The `store` method must be conducted as an atomic transaction regardless of how many preference attributes have been modified.[xc] The portlet container implementation is responsible for handling concurrent writes to avoid inconsistency in portlet preference attributes. All changes made to `PortletPreferences` object not followed by a call to the `store` method must be discarded when the portlet finishes the `processAction` method. [xci] If the `store` method is invoked within the scope of a `render` method invocation, it must throw an `UnsupportedOperationException`.[xcii]

The `PortletPreferences` object must reflect the current values of the persistent store when the portlet container invokes the `processAction` and `render` methods of the portlet. [xciii]

PLT.14.2 Preference Attributes Scopes

Portlet Specification assumes preference attributes are user specific, it does not make any provision at API level or at semantic level for sharing preference attributes among users. If a portal/portlet-container implementation provides an extension mechanism for sharing preference attributes, it should be well documented how the sharing of preference attributes works. Sharing preference attributes may have significant impact on the behavior of a portlet. In many circumstances it could be inappropriate sharing attributes that are meant to be private or confidential to the user.

PLT.14.3 Preference Attributes definition

The portlet definition may define the preference attributes a portlet uses.

A preference attribute definition may include initial default values. A preference attribute definition may also indicate if the attribute is non-modifiable.

An example of a fragment of preferences attributes definition in the deployment descriptor would be:

```
<portlet>
...
  <!—- Portlet Preferences -->
  <portlet-preferences>
    <preference>
      <description>Preferred stock symbols</description>
      <name>PreferredStockSymbols</name>
      <value>FOO</value>
      <value>XYZ</value>
      <~~non-~~modifiable>0~~1~~</~~non-~~modifiable>
    </preference>
    <preference>
      <description>Quotes Service URL</description>
      <name>quotesFeedURL</name>
      <value>http://www.foomarket.com/quotes</value>
    </preference>
  </portlet-preferences>
</portlet>
```

If a preference attribute definition does not contain the ~~non-~~`modifiable` element set to `0`, the preference attribute is modifiable when the portlet is processing a request in any of the standard portlet modes (`VIEW`, `EDIT` or `HELP`).[xciv] Portlets may change the value of modifiable preference attributes using the `setValue`, `setValues` and `reset` methods of the `PortletPreferences` interface. Deployers may use the ~~non-~~`modifiable element` set to `0` to fix certain preference values at deployment time. Portal/portlet-containers may allow changing non-modifiable preference attributes while performing administration tasks.

Portlets are not restricted to use preference attributes defined in the deployment descriptor. They can programmatically add preference attributes using names not defined in the deployment descriptor. These preferences attributes must be treated as modifiable attributes. [xcv]

Portal administration and configuration tools may use and change, default preference attributes when creating a new portlet preferences objects.

PLT.14.3.1 Localizing Preference Attributes

The Portlet Specification does not define a specific mechanism for localizing preference attributes. It leverages the JDK ResourceBundle classes.

To enable localization support of preference attributes for administration and configuration tools, developers should adhere to the following naming convention for entries in the portlet's ResourceBundle.

Entries for preference attribute names should be constructed as `'javax.portlet.preference.<attribute-name>'`, where `<attribute-name>` is the preference attribute name.

Entries for preference attribute values that require localization should be constructed as `'<javax.portlet.preference.<attribute-name>.<attribute-value>'`, where `<attribute-name>` is the preference attribute name and `<attribute-value>` is the localized preference attribute value.

If using Properties based resource bundles, attribute names and attribute value must not contain white-space characters or the = (equal) sign.

PLT.14.4 Validating Preference values

A class implementing the `PreferencesValidator` interface can be associated with the preferences definition in the deployment descriptor, as shown in the following example:

```
<!—- Portlet Preferences -->
<portlet-preferences>
    ...
    <validator>com.foo.portlets.XYZValidator</validator>
</portlet-preferences>
```

`A PreferencesValidator` implementation must be coded in a thread safe manner as the portlet container may invoke concurrently from several requests. If a portlet definition includes a validator, the portlet container must create a single validator instance per portlet definition .[xcvi] If the application is a distributed application, the portlet container must create an instance per VM.[xcvii]

When a validator is associated with the preferences of a portlet definition, the `store` method of the `PortletPreferences` implementation must invoke the `validate` method of the validator before writing the changes to the persistent store.[xcviii] If the validation fails, the `PreferencesValidator` implementation must throw a `ValidatorException`. If a `ValidatorException` is thrown, the `portlet container` must cancel the store operation and it must propagate the exception to the portlet.[xcix] If the validation is successful, the store operation must be completed.[c]

When creating a `ValidatorException,` portlet developers may include the set of preference attributes that caused the validator to fail. It is left to the developers to indicate the first preference attribute that failed or the name of all the invalid preference attributes.

PLT.15

Sessions

To build effective portlet applications, it is imperative that requests from a particular client be associated with each other. There are many session tracking approaches such as HTTP Cookies, SSL Sessions or URL rewriting. To free the programmer from having to deal with session tracking directly, this specification defines a `PortletSession` interface that allows a portal/portlet-container to use any of the approaches to track a user's session without involving the developers in the nuances of any one approach.

PLT.15.1 Creating a Session

A session is considered "new" when it is only a prospective session and has not been established. Because the portlet specification is designed around a request-response based protocol (HTTP would be an example of this type of protocol) a session is considered to be new until a client "joins" it. A client joins a session when session tracking information has been returned to the server indicating that a session has been established. Until the client joins a session, it cannot be assumed that the next request from the client will be recognized as part of a session.

The session is considered to be "new" if either of the following is true:

- The client does not yet know about the session
- The client chooses not to join a session

These conditions define the situation where the portlet container has no mechanism by which to associate a request with a previous request. A portlet developer must design the application to handle a situation where a client has not, cannot, or will not join a session.

For portlets within the same portlet application, a portlet container must ensure that every portlet request generated as result of a group of requests originated from the portal to complete a single client request receive or acquire the same session.[ci] In addition, if within these portlet requests more than one portlet creates a session, the session object must be the same for all the portlets in the same portlet application.[cii]

PLT.15.2 Session Scope

`PortletSession` objects must be scoped at the portlet application context level.[ciii]

Each portlet application has its own distinct `PortletSession` object per user session. The portlet container must not share the `PortletSession` object or the attributes stored in it among different portlet applications or among different user sessions.[civ]

To illustrate this requirement with an example: if a portlet sets an object in the session, a portlet from another portlet application running in the same portlet container must not have access to the object.

PLT.15.3 Binding Attributes into a Session

A portlet can bind an object attribute into a `PortletSession` by name.

The `PortletSession` interface defines two scopes for storing objects, `APPLICATION_SCOPE` and `PORTLET_SCOPE`.

Any object stored in the session using the `APPLICATION_SCOPE` is available to any other portlet that belongs to the same portlet application and that handles a request identified as being a part of the same session.[cv]

Objects stored in the session using the `PORTLET_SCOPE` must be available to the portlet during requests for the same portlet window that the objects where stored from.[cvi] The object must be stored in the `PORTLET_SCOPE` with the following fabricated attribute name `‘javax.portlet.p.<ID>?<ATTRIBUTE_NAME>’`. `<ID>` is a unique identification for the portlet window (assigned by the portal/portlet-container) that must not contain a ‘?’ character.[cvii] `<ATTRIBUTE_NAME>` is the attribute name used to set the object in the `PORTLET_SCOPE` of the portlet session.

Attributes stored in the PORTLET_SCOPE are not protected from other web components of the portlet application. They are just conveniently namespaced.

The `setAttribute` method of the `PortletSession` interface binds an object to the session into the specified scope. For example:

```
PortletSession session = request.getSession(true);
URL url = new URL(“http://www.foo.com”);
session.setAttribute(“home.url”,url,session.APPLICATION_SCOPE);
session.setAttribute(“bkg.color”,”RED”,session.PORTLET_SCOPE);
```

The `getAttribute` method from the `PortletSession` interface is used to retrieve attributes stored in the session.

To remove objects from the session, the `removeAttribute` method is provided by the `PortletSession` interface.

Objects that need to know when they are placed into a session, or removed from a session must implement the `HttpSessionBindingListener` of the servlet API (see *Servlet Specification 2.3, SRV.7.4* Section). The `PortletSessionUtils` class provides utility methods to help determine the scope of the object in the `PortletSession`. If the object was stored in the `PORTLET_SCOPE`, the `PortletSessionUtils` allows retrieving the attribute name without any portlet-container fabricated prefix. Portlet developers should always use the `PortletSessionUtils` class to deal with attributes in the `PORTLET_SCOPE` when accessing them through the servlet API.

PLT.15.4 Relationship with the Web Application HttpSession

A Portlet Application is also a Web Application. The Portlet Application may contain servlets and JSPs in addition to portlets. Portlets, servlets and JSPs may share information through their session.

The `PortletSession` must store all attributes in the `HttpSession` of the portlet application. A direct consequence of this is that data stored in the `HttpSession` by servlets or JSPs is accessible to portlets through the `PortletSession` in the portlet application scope.[cviii] Conversely, data stored by portlets in the `PortletSession` in the portlet application scope is accessible to servlets and JSPs through the `HttpSession`. [cix]

If the HttpSession object is invalidated, the PortletSession object must also be invalidated by the portlet container.[cx] If the PortletSession object is invalidated by a portlet, the portlet container must invalidate the associated HttpSession object.[cxi]

PLT.15.4.1 HttpSession Method Mapping

The following methods of the `PortletSession` interface must be based on the methods of the `HttpSession` interface of identical names: `getCreationTime`, `getId`, `getLastAccessedTime`, `getMaxInactiveInterval`, `invalidate`, `isNew` and `setMaxInactiveInterval`.

The `getAttribute`, `setAttribute`, `removeAttribute` and `getAttributeNames` methods of the `PortletSession` interface must be based on the `HttpSession` interface methods of identical names adhering to the following rules:

- The attribute names must be the same if APPLICATION_SCOPE scope is used.[cxii]
- The attribute name has to conform with the specified prefixing if PORTLET_SCOPE is used.[cxiii]
- The variant of these methods that does not receive a scope must be treated as PORTLET_SCOPE.[cxiv]

PLT.15.5 Reserved HttpSession Attribute Names

Session attribute names starting with "`javax.portlet.`" are reserved for usage by the Portlet Specification and for Portlet Container vendors. A Portlet Container vendor may use this reserved namespace to store implementation specific components. Application Developers must not use attribute names starting with this prefix.

PLT.15.6 Session Timeouts

The portlet session follows the timeout behavior of the servlet session as defined in the *Servlet Specification 2.3, SRV.7.5* Section.

PLT.15.7 Last Accessed Times

The portlet session follows the last accessed times behavior of the servlet session as defined in the *Servlet Specification 2.3, SRV.7.6* Section.

PLT.15.8 Important Session Semantics

The portlet session follows the same semantic considerations as the servlet session as defined in the *Servlet Specification 2.3, SRV.7.7.3* Section.

These considerations include *Threading Issues*, *Distributed Environments* and *Client Semantics*.[cxv]

PLT.16

Dispatching Requests to Servlets and JSPs

Portlets can delegate the creation of content to servlets and JSPs. The `PortletRequestDispatcher` interface provides a mechanism to accomplish this.

PLT.16.1 Obtaining a PortletRequestDispatcher

A portlet may use a `PortletRequestDispatcher` object only when executing the `render` method of the `Portlet` interface. `PortletRequestDispatcher` objects may be obtained using one of the following methods of the `PortletContext` object:

- `getRequestDispatcher`
- `getNamedDispatcher`

The `getRequestDispatcher` method takes a String argument describing a path within the scope of the `PortletContext` of a portlet application. This path must begin with a '/' and it is relative to the `PortletContext root`.[cxvi]

The `getNamedDispatcher` method takes a String argument indicating the name of a servlet known to the `PortletContext` of the portlet application.

If no resource can be resolved based on the given path or name the methods must return `null`.[cxvii]

PLT.16.1.1 Query Strings in Request Dispatcher Paths

Query strings used in the path for obtaining a `PortletRequestDispatcher` object follows the same rules as defined in the request dispatcher paths in *Servlet Specification 2.3, SRV.8.1.1* Section.[cxviii]

PLT.16.2 Using a Request Dispatcher

To include a servlet or a JSP, a portlet calls the `include` method of the `PortletRequestDispatcher` interface. The parameters to these methods must be the request and response arguments that were passed in via the `render` method of the `Portlet` interface.[cxix]

The portlet container must ensure that the servlet or JSP called through a PortletRequestDispatcher is called in the same thread as the PortletRequestDispatcher include invocation.[cxx]

PLT.16.3 The Include Method

The include method of the `PortletRequestDispatcher` interface may be called at any time and multiple times within the `render` method of the `Portlet` interface. The servlet or JSP being included can make a limited use of the received `HttpServletRequest` and `HttpServletResponse` objects.

Servlets and JSPs included from portlets should not use the servlet `RequestDispatcher` `forward` method as it behavior may be non-deterministic.

PLT.16.3.1 Included Request Parameters

Except for servlets obtained by using the `getNamedDispatcher` method, a servlet or JSP being used from within an `include` call has access to the path used to obtain the `PortletRequestDispatcher`. The following request attributes must be set[cxxi]:

```
      javax.servlet.include.request_uri
      javax.servlet.include.context_path
      javax.servlet.include.servlet_path
      javax.servlet.include.path_info
      javax.servlet.include.query_string
```

These attributes are accessible from the included servlet via the `getAttribute` method on the request object.

If the included servlet was obtained by using the `getNamedDispatcher` method these attributes are not set.

PLT.16.3.2 Included Request Attributes

In addition to the request attributes specified in *Servlet Specification 2.3, SRV.8.3.1* Section, the included servlet or JSP must have the following request attributes set:

Request Attribute	Type
`javax.portlet.config`	`javax.portlet.PortletConfig`
`javax.portlet.request`	`javax.portlet.RenderRequest`
`javax.portlet.response`	`javax.portlet.RenderResponse`

These attributes must be the same portlet API objects accessible to the portlet doing the include call.[cxxii] They are accessible from the included servlet or JSP via the `getAttribute` method on the `HttpServletRequest` object.

PLT.16.3.3 Request and Response objects for Included Servlets/JSPs

The target servlet or JSP of portlet request dispatcher has access to a limited set of methods of the request and the response objects.

The following methods of the `HttpServletRequest` must return `null`: `getProtocol`, `getRemoteAddr`, `getRemoteHost`, `getRealPath`, and `getRequestURL`.[cxxiii]

The following methods of the `HttpServletRequest` must return the path and query string information used to obtain the `PortletRequestDispatcher` object: `getPathInfo`, `getPathTranslated`, `getQueryString`, `getRequestURI` and `getServletPath`.[cxxiv]

The following methods of the `HttpServletRequest` must be equivalent to the methods of the `PortletRequest` of similar name: `getScheme`, `getServerName`, `getServerPort`, `getAttribute`, `getAttributeNames`, `getParameter`, `getParameterNames`, `getParameterValues`, `getParameterMap`, ~~`getReader`~~, `setAttribute`, `removeAttribute`, `getLocale`, `isSecure`, `getAuthType`, `getContextPath`, `getRemoteUser`, `getUserPrincipal`, `getRequestedSessionId`, `isRequestedSessionIdValid`.[cxxv]

The following methods of the `HttpServletRequest` must do no operations and return `null`: `getCharacterEncoding`, `setCharacterEncoding`, , `getContentType`, `getInputStream` and `getReader`. The `getContentLength` method of the `HttpServletRequest` must return `0`.

The `getLocales` method of `HttpServletRequest` must return an `Enumeration` of one element containing the same `Locale` returned by the `getLocale` method of the `PortletRequest`.[cxxvi]

The following methods of the `HttpServletRequest` must be based on the properties provided by the `getProperties` method of the `PortletRequest` interface: `getHeader`, `getHeaders`, `getHeaderNames`, `getCookies`, `getDateHeaders` and `getIntHeaders`.[cxxvii].

The following methods of the `HttpServletRequest` must provide the functionality defined by the *Servlet Specification 2.3*: `getRequestDispatcher`, `getMethod`, `isUserInRole`, `getSession`, `isRequestedSessionIdFromCookie`, `isRequestedSessionIdFromURL` and `isRequestedSessionIdFromUrl`.[cxxviii]

The following methods of the `HttpServletResponse` must return `null`: `encodeRedirectURL` and `encodeRedirectUrl`.[cxxix]

The following methods of the `HttpServletResponse` must be equivalent to the methods of the `RenderResponse` of similar name: `getCharacterEncoding`, `setBufferSize`, `flushBuffer`, `resetBuffer`, `reset`, getBufferSize, `isCommitted`, `getOutputStream`, `getWriter`, `encodeURL` and `encodeUrl`.[cxxx]

The following methods of the `HttpServletResponse` must perform no operations: `setContentType`, `setContentLength`, `setLocale`, `addCookie`, `sendError`, `sendRedirect`, `setDateHeader`, `addDateHeader`, `setHeader`, `addHeader`, `setIntHeader`, `addIntHeader` and `setStatus`.[cxxxi] The `containsHeader` method of the `HttpServletResponse` must return `false`.

The `getLocale` method of the `HttpServletResponse` must be based on the `getLocale` method of the `PortletRequest`.[cxxxii]

PLT.16.3.4 Error Handling

If the servlet or JSP that is the target of a request dispatcher throws a runtime exception or a checked exception of type `IOException`, it must be propagated to the calling portlet.[cxxxiii] All other exceptions, including a `ServletException`, must be wrapped with a `PortletException`. The root cause of the exception must be set to the original exception before being propagated.[cxxxiv]

PLT.17

User Information

Commonly, portlets provide content personalized to the user making the request. To do this effectively they may require access to user attributes such as the name, email, phone or address of the user. Portlet containers provide a mechanism to expose available user information to portlets.

PLT.17.1 Defining User Attributes

The deployment descriptor of a portlet application must define the user attribute names the portlets use. The following example shows a section of a deployment descriptor defining a few user attributes:

```
<portlet-app>
  …
  <user-attribute>
    <description>User Given Name</description>
    <name>user.name.given</name>
  </user-attribute>
  <user-attribute>
    <description>User Last Name</description>
    <name>user.name.family</name>
  </user-attribute>
  <user-attribute>
    <description>User eMail</description>
    <name>user.home-info.online.email</name>
  </user-attribute>
  <user-attribute>
    <description>Company Organization</description>
    <name>user.business-info.postal.organization</name>
  </user-attribute>
  …
<portlet-app>
```

A deployer must map the portlet application's logical user attributes to the corresponding user attributes offered by the runtime environment. At runtime, the portlet container uses this mapping to expose user attributes to the portlets of the portlet application. User attributes of the runtime environment not mapped as part of the deployment process must not be exposed to portlets.[cxxxv]

Refer to *PLT.## User Information Attribute Names* Appendix for a list of recommended names.

PLT.17.2 Accessing User Attributes

Portlets can obtain an unmodifiable `Map` object containing the user attributes, of user associated with the current request, from the request attributes. The `Map` object can be retrieved using the `USER_INFO` constant defined in the `PortletRequest` interface. If the request is done in the context of an un-authenticated user, calls to the `getAttribute` method of the request using the `USER_INFO` constant must return `null`.[cxxxvi]. If the user is authenticated and there are no user attributes available, the `Map` must be an empty `Map`.

The Map object must contain a String name value pair for each available user attribute. The Map object should only contain user attributes that have been mapped during deployment..[cxxxvii]

An example of a portlet retrieving user attributes would be:

```
...
Map userInfo = (Map) request.getAttribute(request.USER_INFO);
String givenName = (userInfo!=null)
      ? (String) userInfo.get(“user.name.given”) : “”;
String lastName  = (userInfo!=null)
      ? (String) userInfo.get(“user.name.family”) : “”;
...
```

PLT.17.3 Important Note on User Information

The Portlet Specification expert group is aware of the fact that user information is outside of the scope of this specification. As there is not standard Java standard to access user information, and until such Java standard is defined, the Portlet specification will provide this mechanism that is considered to be the least intrusive from the portlet API perspective. At a latter time, when the Java standard for user information is defined, the current mechanism will be deprecated in favor of it.

PLT.18 Caching

Caching content helps improve the Portal response time for clients. It also helps to reduce the load on servers.

The Portlet Specification defines an expiration based caching mechanism. This caching mechanism is per portlet per client. Cached content must not be shared across different client displaying the same portlet.

Portlet containers are not required to implement expiration caching. Portlet containers implementing this caching mechanism may disable it, partially or completely, at any time to free memory resources.

PLT.18.1 Expiration Cache

Portlets that want their content to be cached using expiration cache must define the duration (in seconds) of the expiration cache in the deployment descriptor.

The following is an example of a portlet definition where the portlet defines that its content should be cache for 5 minutes (300 seconds).

```
...
<portlet>
  ...
    <expiration-cache>300</expiration-cache>
  ...
</portlet>
...
```

A portlet that has defined an expiration cache in its portlet definition may programmatically alter the expiration time by setting the `expiration-cache` property in the `RenderResponse` object. If the expiration value is set to 0, caching is disabled for the portlet. If the `expiration-cache` property is set to –1, the cache does not expire. If during a `render` invocation the `expiration-cache property` is not set, the expiration time defined in the deployment descriptor must be used. For a portlet that has not defined expiration cache in the deployment descriptor, if the `expiration-cache` property is set it must be ignored by the portlet-container.

If the content of a portlet is cached, the cache has not expired and the portlet is not the target of the client request, then the request handling methods of the portlet should not be invoked as part of the client request. Instead, the portlet-container should use the data from the cache.

If the content of a portlet is cached and a client request is targeted to the portlet, the portlet container must discard the cache and invoke the request handling methods of the portlet.

PLT.19

Portlet Applications

A portlet application is a web application, as defined in *Servlet Specification 2.3, SRV.9* Chapter, containing portlets and a portlet deployment descriptor in addition to servlets, JSPs, HTML pages, classes and other resources normally found in a web application. A bundled portlet application can run in multiple portlet containers implementations.

PLT.19.1 Relationship with Web Applications

All the portlet application components and resources other than portlets are managed by the servlet container the portlet container is built upon.

PLT.19.2 Relationship to PortletContext

The portlet container must enforce a one to one correspondence between a portlet application and a `PortletContext`.[cxxxviii] If the application is a distributed application, the portlet container must create an instance per VM.[cxxxix] A `PortletContext` object provides a portlet with its view of the application.

PLT.19.3 Elements of a Portlet Application

A portlet application may consist of portlets plus other elements that may be included in web applications, such as servlets, JSP™ pages, classes, static documents.

Besides the web application specific meta information, the portlet application must include descriptive meta information about the portlets it contains.

PLT.19.4 Directory Structure

A portlet application follows the same directory hierarchy structure as web applications.

In addition it must contain a `/WEB-INF/portlet.xml` deployment descriptor file.

Portlet classes, utility classes and other resources accessed through the portlet application classloader must reside within the `/WEB-INF/classes` directory or within a JAR file in the `/WEB-INF/lib/` directory.

PLT.19.5 Portlet Application Classloader

The portlet container must use the same classloader the servlet container uses for the web application resources for loading the portlets and related resources within the portlet application.[cxl]

The portlet container must ensure that requirements defined in the *Servlet Specification 2.3 SRV.9.7.1* and *SRV.9.7.2* Sections are fulfilled.[cxli]

PLT.19.6 Portlet Application Archive File

Portlet applications are packaged as web application archives (WAR) as defined in the *Servlet Specification 2.3 SRV.9.6* Chapter.

PLT.19.7 Portlet Application Deployment Descriptor

In addition to a web application deployment descriptor, a portlet application contains a portlet application deployment descriptor. The portlet deployment descriptor contains configuration information for the portlets contained in the application.

Refer to *### Packaging and Deployment Descriptor* Chapter for more details on the portlet application deployment descriptor.

PLT.19.8 Replacing a Portlet Application

A portlet container should be able to replace a portlet application with a new version without restarting the container. In addition, the portlet container should provide a robust method for preserving session data within that portlet application.

PLT.19.9 Error Handling

It is left to the portal/portlet-container implementation how to react when a portlet throws an exception while processing a request. For example, the portal/portlet-container could render an error page instead of the portal page, render an error message in the portlet window of the portlet that threw the exception or remove the portlet from the portal page and log an error message for the administrator.

PLT.19.10 Portlet Application Environment

The portlet specification leverages the provisions made by the *Servlet Specification 2.3 SRV.9.11* Section.

PLT.20

Security

Portlet applications are created by Application Developers who license the application to a Deployer for installation into a runtime environment. Application Developers need to communicate to Deployers how the security is to be set up for the deployed application.

PLT.20.1 Introduction

A portlet application contains resources that can be accessed by many users. These resources often traverse unprotected, open networks such as the Internet. In such an environment, a substantial number of portlet applications will have security requirements.

The portlet container is responsible for informing portlets of the roles users are in when accesing them. The portlet container does not deal with user authentication. It should leverage the authentication mechanisms provided by the underlying servlet container defined in the *Servlet Specification 2.3, SRV.12.1* Section.

PLT.20.2 Roles

The portlet specification shares the same definition as roles of the *Servlet Specification 2.3, SRV.12.4* Section.

PLT.20.3 Programmatic Security

Programmatic security consists of the following methods of the `Request` interface:

- `getRemoteUser`
- `isUserInRole`
- `getUserPrincipal`

The `getRemoteUser` method returns the user name the client used for authentication. The `isUserInRole` method determines if a remote user is in a specified security role. The `getUserPrincipal` method determines the principal name of the current user and returns a `java.security.Principal` object. These APIs allow portlets to make business logic decisions based on the information obtained.

The values that the portlet API `getRemoteUser` and `getUserPrincipal` methods return the same values returned by the equivalent methods of the servlet response object.[cxlii] Refer to the *Servlet Specification 2.3, SRV.12.3* Section for more details on these methods.

The `isUserInRole` method expects a string parameter with the role-name. A `security-role-ref` element must be declared by the portlet in deployment descriptor with a `role-name` sub-element containing the role-name to be passed to the method. The `security-role-ref` element should contain a `role-link` sub-element whose value is the name of the application security role that the user may be mapped into. This mapping is specified in the `web.xml` deployment descriptor file. The container uses the mapping of `security-role-ref` to `security-role` when determining the return value of the call.[cxliii]

For example, to map the security role reference "FOO" to the security role with role-name "manager" the syntax would be:

```
<portlet-app>
    ...
    <portlet>
        ...
        <security-role-ref>
            <role-name>FOO</role-name>
            <role-link>manager</manager>
        </security-role-ref>
    </portlet>
    ...
    ...
</portlet-app>
```

In this case, if the portlet called by a user belonging to the "manager" security role made the API call `isUserInRole("FOO")`, then the result would be true.

If the `security-role-ref` element does not define a `role-link` element, the container must default to checking the `role-name` element argument against the list of `security-role` elements defined in the web.xml deployment descriptor of the portlet application.[cxliv] The `isUserInRole` method references the list to determine whether the caller is mapped to a security role. The developer must be aware that the use of this default mechanism may limit the flexibility in changing role-names in the application without having to recompile the portlet making the call.

PLT.20.4 Specifying Security Constraints

Security constraints are a declarative way of annotating the intended protection of portlets. A constraint consists of the following elements:

- portlet collection
- user data constraint

A portlets collection is a set of portlet names that describe a set of resources to be protected. All requests targeted to portlets listed in the portlets collection are subject to the constraint.

A user data constraint describes requirements for the transport layer for the portlets collection. The requirement may be for content integrity (preventing data tampering in the communication process) or for confidentiality (preventing reading while in transit). The container must at least use SSL to respond to requests to resources marked integral or confidential.

For example, to define that a portlet requires a confindential transport the syntax would be:

```
<portlet-app>
    ...
    <portlet>
        <portlet-name>accountSummary</portlet-name>
        ...
    </portlet>
    ...
    <security-constraint~~s~~>
         <display-name>Secure Portlets</display-name>
         <portlet-collection>
            <portlet-name>accountSummary</portlet-name>
        </portlet-collection>
        <user-data-constraint/>
           <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint~~s~~>
    ...
</portlet-app>
```

PLT.20.5 Propagation of Security Identity in EJB™ Calls

A security identity, or principal, must always be provided for use in a call to an enterprise bean.

The default mode in calls to EJBs from portlet applications should be for the security identity of a user, in the portlet container, to be propagated to the EJB™ container.

Portlet containers, running as part of a J2EE platform, are required to allow users that are not known to the portlet container to make calls to the the EJB™ container. In these scenarios, the portlet application may specify a `run-as` element in the `web.xml` deployment descriptor. When it is specified, the container must propagate the security identity of the caller to the EJB layer in terms of the security role name defined in the `run-as` element.[cxlv] The security role name must be one of the security role names defined for the `web.xml` deployment descriptor.[cxlvi] Alternatively, portlet application code may be sole processor of the signon into the EJB™ container.

PLT.21

Packaging and Deployment Descriptor

The deployment descriptor conveys the elements and configuration information of a portlet application between Application Developers, Application Assemblers, and Deployers. Portlet applications are self-contained applications that are intended to work without further resources. Portlet applications are managed by the portlet container.

In the case of portlet applications, there are two deployment descriptors: one to specify the web application resources (web.xml) and one to specify the portlet resources (portlet.xml). The web application deployment descriptor is explained in detail in the *Servlet Specification 2.3, SRV.13Deployment Descriptor* Chapter.

PLT.21.1 Portlet and Web Application Deployment Descriptor

For the Portlet Specification version 1.0 there is a clear distinction between web resources, like servlets, JSPs, static markup pages, etc., and portlets. This is due to the fact that, in the *Servlet Specification 2.3,* the web application deployment descriptor is not extensible. All web resources that are not portlets must be specified in the `web.xml` deployment descriptor. All portlets and portlet related settings must be specified in an additional file called `portlet.xml`. The format of this additional file is described in detail below.

The following portlet web application properties need to be set in the `web.xml` deployment descriptor:

- portlet application description using the `<description>` tag
- portlet application name using the `<display>` tag
- portlet application security role mapping using the `<security-role>` tag

PLT.21.2 Packaging

All resources, portlets and the deployment descriptors are package together in one web application archive (WAR file). This format is described in *Servlet Specification 2.3, SRV.9 Web Application* Chapter.

In addition to the resources described in the *Servlet Specification 2.3, SRV.9 Web Application* Chapter a portlet application `WEB-INF` directory consists of:

- The `/WEB-INF/portlet.xml` deployment descriptor.
- Portlet classes in the `/WEB-INF/classes` directory.
- Portlet Java ARchive files `/WEB-INF/lib/*.jar`

PLT.21.2.1 Example Directory Structure

The following is a listing of all the files in a sample portlet application:

```
/images/myButton.gif
/META-INF/MANIFEST.MF
/WEB-INF/web.xml
/WEB-INF/portlet.xml
/WEB-INF/lib/myHelpers.jar
/WEB-INF/classes/com/mycorp/servlets/MyServlet.class
/WEB-INF/classes/com/mycorp/portlets/MyPortlet.class
/WEB-INF/jsp/myHelp.jsp
```

Portlet applications that need additional resources that cannot be packaged in the WAR file, like EJBs, may be packaged together with these resources in an EAR file.

PLT.21.2.2 Version Information

If portlet application providers want to provide version information about the portlet application is recommended to provide a `META-INF/MANIFEST.MF` entry in the WAR file. The `'Implementation-*'` attributes should be used to define the version information.

Example:

```
Implementation-Title: myPortletApplication
Implementation-Version: 1.1.2
Implementation-Vendor: SunMicrosystems. Inc.
```

PLT.21.3 Portlet Deployment Descriptor Elements

The following types of configuration and deployment information are required to be supported in the portlet deployment descriptor for all portlet containers:

- Portlet Application Definition
- Portlet Definition

Security information, which may also appear in the deployment descriptor is not required to be supported unless the portlet container is part of an implementation of the J2EE specification.

PLT.21.4 Rules for processing the Portlet Deployment Descriptor

In this section is a listing of some general rules that portlet containers and developers must note concerning the processing of the deployment descriptor for a portlet application:

- Portlet containers should ignore all leading whitespace characters before the first non-whitespace character, and all trailing whitespace characters after the last non-whitespace character for PCDATA within text nodes of a deployment descriptor.
- Portlet containers and tools that manipulate portlet applications have a wide range of options for checking the validity of a WAR. This includes checking the validity of the web application and portlet deployment descriptor documents held within. It is recommended, but not required, that portlet containers and tools validate both deployment descriptors against the corresponding DTD and XML Schema definitions for structural correctness. Additionally, it is recommended that they provide a level of semantic checking. For example, it should be checked that a role referenced in a security constraint has the same name as one of the security roles defined in the deployment descriptor. In cases of non-conformant portlet applications, tools and containers should inform the developer with descriptive error messages. High end application server vendors are encouraged to supply this kind of validity checking in the form of a tool separate from the container.

In elements whose value is an "enumerated type", the value is case sensitive.

PLT.21.5 Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://java.sun.com/xml/ns/portlet"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:portlet="http://java.sun.com/xml/ns/portlet"
xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.0" xml:lang="EN">
  <annotation>
    <documentation>
    This is the XML Schema for the Portlet 1.0 deployment descriptor.
    </documentation>
  </annotation>
  <annotation>
    <documentation>
    The following conventions apply to all J2EE
    deployment descriptor elements unless indicated otherwise.
    - In elements that specify a pathname to a file within the
      same JAR file, relative filenames (i.e., those not
      starting with "/") are considered relative to the root of
      the JAR file's namespace.  Absolute filenames (i.e., those
      starting with "/") also specify names in the root of the
      JAR file's namespace.  In general, relative names are
      preferred.  The exception is .war files where absolute
      names are preferred for consistency with the Servlet API.
    </documentation>
  </annotation>
  <!-- <include schemaLocation="j2ee_1_4.xs"/> -->
  <!-- ********************************************************** -->
  <import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <element name="portlet-app" type="portlet:portlet-appType">
    <annotation>
      <documentation>
      The portlet-app element is the root of the deployment descriptor
      for a portlet application
      </documentation>
    </annotation>
  </element>
  <complexType name="portlet-appType">
    <sequence>
      <element name="portlet" type="portlet:portletType" maxOccurs="unbounded"/>
      <element name="custom-portlet-mode" type="portlet:custom-portlet-modeType"
minOccurs="0" maxOccurs="unbounded"/>
      <element name="custom-window-state" type="portlet:custom-window-stateType"
minOccurs="0" maxOccurs="unbounded"/>
      <element name="user-attribute" type="portlet:user-attributeType"
minOccurs="0" maxOccurs="unbounded"/>
      <element name="security-constraint" type="portlet:security-constraintType"
minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="version" type="string" use="required"/>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="custom-portlet-modeType">
    <annotation>
      <documentation>
      A custom portlet mode that one or more portlets in
      this portlet application supports.
      Used in: portlet-app
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="portlet-mode" type="portlet:portlet-modeType"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="custom-window-stateType">
    <annotation>
      <documentation>
      A custom window state that one or more portlets in this
      portlet application supports.
      Used in: portlet-app
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="window-state" type="portlet:window-stateType"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="expiration-cacheType">
    <annotation>
      <documentation>
      Expriation-cache defines expiration-based caching for this
      portlet. The parameter indicates
      the time in seconds after which the portlet output expires.
      -1 indicates that the output never expires.
      Used in: portlet
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="init-paramType">
    <annotation>
      <documentation>
      The init-param element contains a name/value pair as an
      initialization param of the portlet
      Used in:portlet
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="name" type="portlet:nameType"/>
      <element name="value" type="portlet:valueType"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="keywordsType">
    <annotation>
      <documentation>
      Locale specific keywords associated with this portlet.
      The kewords are separated by commas.
      Used in: portlet-info
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="mime-typeType">
    <annotation>
      <documentation>
      MIME type name, e.g. "text/html".
      The MIME type may also contain the wildcard
      character '*', like "text/*" or "*/*".
      Used in: supports
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <simpleType name="modifiableType">
    <annotation>
      <documentation>
      modifiable indicates that a setting can not
      be changed in any of the standard portlet modes
      (VIEW, EDIT or HELP).
      Valid values are:
      - 0 for non-modifiable
      - 1 for modifiable
      Used in: preferences
      </documentation>
    </annotation>
    <restriction base="portlet:string">
      <enumeration value="0"/>
      <enumeration value="1"/>
    </restriction>
  </simpleType>
  <complexType name="nameType">
    <annotation>
      <documentation>
      The name element contains the name of a parameter.
      Used in: init-param, ...
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="portletType">
    <annotation>
      <documentation>
      The portlet element contains the declarative data of a portlet.
      Used in: portlet-app
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="init-param" type="portlet:init-paramType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="portlet-name" type="portlet:portlet-nameType"/>
      <element name="display-name" type="portlet:display-nameType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="portlet-class" type="portlet:portlet-classType"/>
      <element name="expiration-cache" type="portlet:expiration-cacheType"
minOccurs="0"/>
      <element name="supports" type="portlet:supportsType"
maxOccurs="unbounded"/>
      <element name="supported-locale" type="portlet:supported-localeType"
minOccurs="0" maxOccurs="unbounded"/>
      <element name="portlet-info" type="portlet:portlet-infoType"/>
      <element name="portlet-preferences" type="portlet:portlet-preferencesType"
minOccurs="0"/>
      <element name="security-role-ref" type="portlet:security-role-refType"
minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="portlet-classType">
    <annotation>
      <documentation>
       The portlet-class element contains the fully
       qualified class name of the portlet.
      Used in: portlet
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="portlet-collectionType">
    <annotation>
      <documentation>
      The portlet-collectionType is used to identify a subset
      of portlets within a portlet application to which a
      security constraint applies.
      Used in: security-constraint
      </documentation>
    </annotation>
    <sequence>
      <element name="portlet-name" type="portlet:portlet-nameType"
maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="portlet-infoType">
    <choice>
      <sequence>
        <element name="title" type="portlet:titleType"/>
        <element name="short-title" type="portlet:short-titleType"
minOccurs="0"/>
        <element name="keywords" type="portlet:keywordsType" minOccurs="0"/>
      </sequence>
      <element name="resource-bundle" type="portlet:resource-bundleType"/>
    </choice>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="portlet-modeType">
    <annotation>
      <documentation>
      Portlet modes. The specification pre-defines the following values
      as valid portlet mode constants:
      EDIT, HELP, VIEW.
      Portlet mode names are not case sensitive.
      Used in: custom-portlet-mode, supports
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="portlet-nameType">
    <annotation>
      <documentation>
      The portlet-name element contains the canonical name of the
      portlet. Each portlet name is unique within the portlet
      application.
      Used in: portlet, portlet-mapping
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="portlet-preferencesType">
    <annotation>
      <documentation>
      Portlet persistent preference store.
      Used in: portlet
      </documentation>
    </annotation>
    <sequence>
      <element name="preference" type="portlet:preferenceType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="preferences-validator" type="portlet:preferences-
validatorType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="preferenceType">
    <annotation>
      <documentation>
      Persistent preference values that may be used for customization
      and personalization by the portlet.
      Used in: user-preferences, portlet-preferences
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="name" type="portlet:nameType"/>
      <element name="value" type="portlet:valueType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="modifiable" type="portlet:modifiableType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="preferences-validatorType">
    <annotation>
      <documentation>
      The class specified under preferences-validator implements
      the PreferencesValidator interface to validate the
      preferences settings.
      Used in: user-preferences, portlet-preferences
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="resource-bundleType">
    <annotation>
      <documentation>
      Filename of the resource bundle containing the language specific
      portlet informations in different languages.
      Used in: portlet-info
      </documentation>
    </annotation>
    <simpleContent>
```

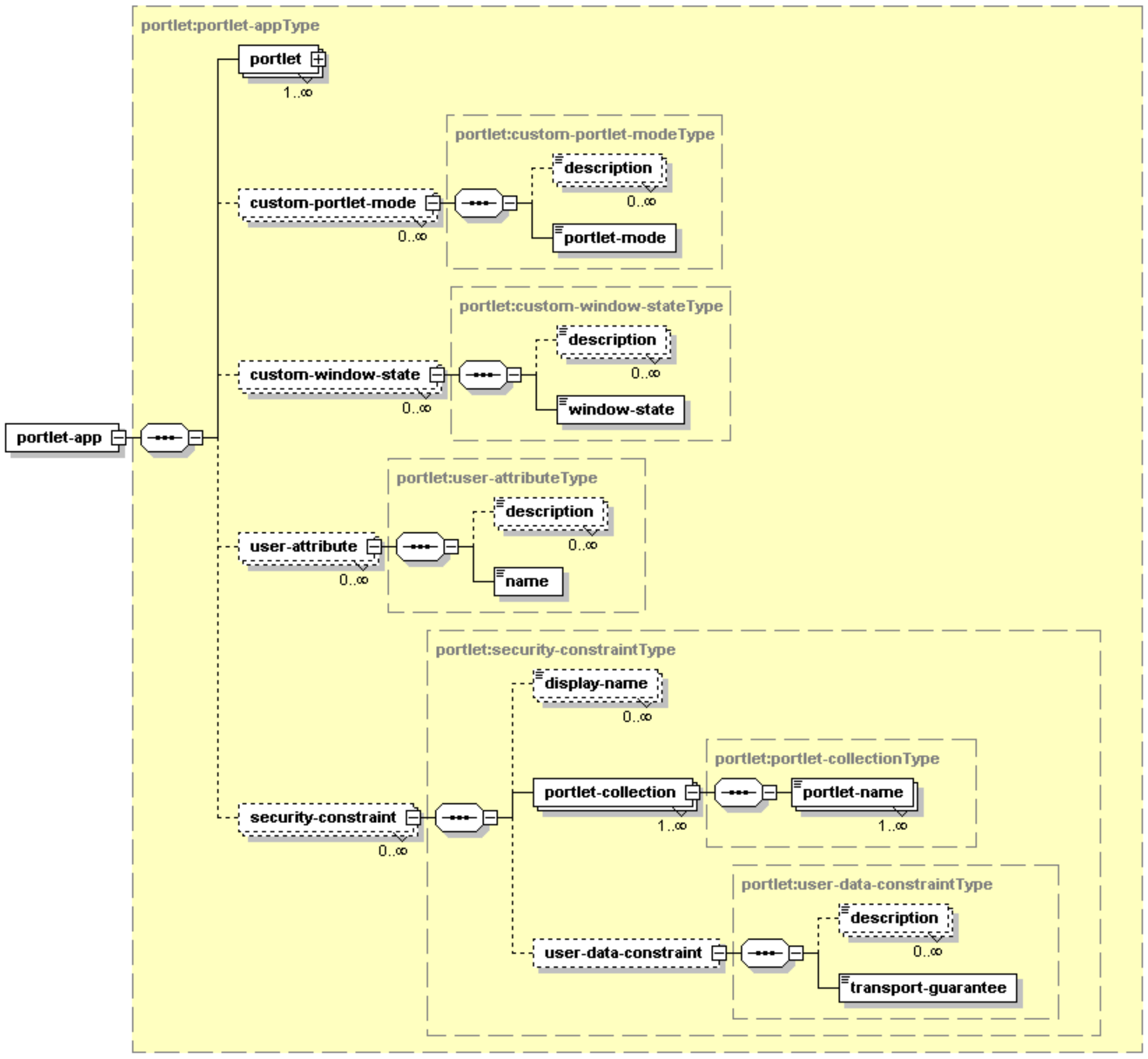
```
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="role-linkType">
    <annotation>
      <documentation>
      The role-link element is a reference to a defined security role.
      The role-link element must contain the name of one of the
      security roles defined in the security-role elements.
      Used in: security-role-ref
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="security-constraintType">
    <annotation>
      <documentation>
      The security-constraintType is used to associate
      intended security constraints with one or more portlets.
      Used in: portlet-app
      </documentation>
    </annotation>
    <sequence>
      <element name="display-name" type="portlet:display-nameType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="portlet-collection" type="portlet:portlet-collectionType"
maxOccurs="unbounded"/>
      <element name="user-data-constraint" type="portlet:user-data-
constraintType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="security-role-refType">
    <annotation>
      <documentation>
      The security-role-ref element contains the declaration of a
      security role reference in the web application’s code. The
      declaration consists of an optional description, the security
      role name used in the code, and an optional link to a security
      role. If the security role is not specified, the Deployer must
      choose an appropriate security role.
      The value of the role name element must be the String used
      as the parameter to the
      EJBContext.isCallerInRole(String roleName) method
      or the HttpServletRequest.isUserInRole(String role) method.
      Used in: portlet
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="role-name" type="portlet:role-nameType"/>
      <element name="role-link" type="portlet:role-linkType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="short-titleType">
    <annotation>
      <documentation>
      Locale specific short version of the static title.
      Used in: portlet-info
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="supportsType">
    <annotation>
      <documentation>
      Supports indicates the portlet modes and window states that the
      portlet supports for a specific content type. All portlets must
      support the view mode and normal window state.
      Used in: portlet
      </documentation>
    </annotation>
    <sequence>
      <element name="mime-type" type="portlet:mime-typeType"/>
      <element name="portlet-mode" type="portlet:portlet-modeType" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="supported-localeType">
    <annotation>
      <documentation>
      Indicated the locales the portlet supports.
      Used in: portlet
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="titleType">
    <annotation>
      <documentation>
      Locale specific static title for this portlet.
      Used in: portlet-info
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <simpleType name="transport-guaranteeType">
    <annotation>
      <documentation>
      The transport-guaranteeType specifies that
      the communication between client and portlet should
      be NONE, INTEGRAL, or CONFIDENTIAL.
      NONE means that the portlet does not
      require any transport guarantees. A value of
      INTEGRAL means that the portlet requires that the
      data sent between the client and portlet be sent in
      such a way that it can't be changed in transit.
      CONFIDENTIAL means that the portlet requires
      that the data be transmitted in a fashion that
      prevents other entities from observing the contents
      of the transmission.
      In most cases, the presence of the INTEGRAL or
      CONFIDENTIAL flag will indicate that the use
      of SSL is required.
      Used in: user-data-constraint
      </documentation>
    </annotation>
    <restriction base="portlet:string">
      <enumeration value="NONE"/>
      <enumeration value="INTEGRAL"/>
      <enumeration value="CONFIDENTIAL"/>
    </restriction>
  </simpleType>
  <complexType name="user-attributeType">
    <annotation>
      <documentation>
      User attribute defines a user specific attribute that the
      portlet application needs. The portlet within this application
      can access this attribute via the request parameter USER_INFO
      map.
      Used in: portlet-app
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="name" type="portlet:nameType"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="user-data-constraintType">
    <annotation>
      <documentation>
      The user-data-constraintType is used to indicate how
      data communicated between the client and portlet should be
      protected.
      Used in: security-constraint
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="transport-guarantee" type="portlet:transport-
guaranteeType"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <complexType name="valueType">
    <annotation>
      <documentation>
      The value element contains the value of a parameter.
      Used in: init-param
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <complexType name="window-stateType">
    <annotation>
      <documentation>
      Portlet window state. The specification pre-defines the
      following values as valid window state constants:
      MINIMIZED, NORMAL, MAXIMIZED.
      Window state names are not case sensitive.
      Used in: custom-window-state
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string"/>
    </simpleContent>
  </complexType>
  <!--- everything below is copied from j2ee_1_4.xs -->
  <complexType name="descriptionType">
    <annotation>
      <documentation>
      The description element is used to provide text describing the
      parent element. The description element should include any
      information that the portlet application war file producer wants
      to provide to the consumer of the portlet application war file
      (i.e., to the Deployer). Typically, the tools used by the
      portlet application war file consumer will display the
      description when processing the parent element that contains the
      description.
      Used in: init-param, portlet, portlet-app, security-role
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="string">
        <attribute ref="xml:lang"/>
      </extension>
    </simpleContent>
  </complexType>
  <complexType name="display-nameType">
    <annotation>
```

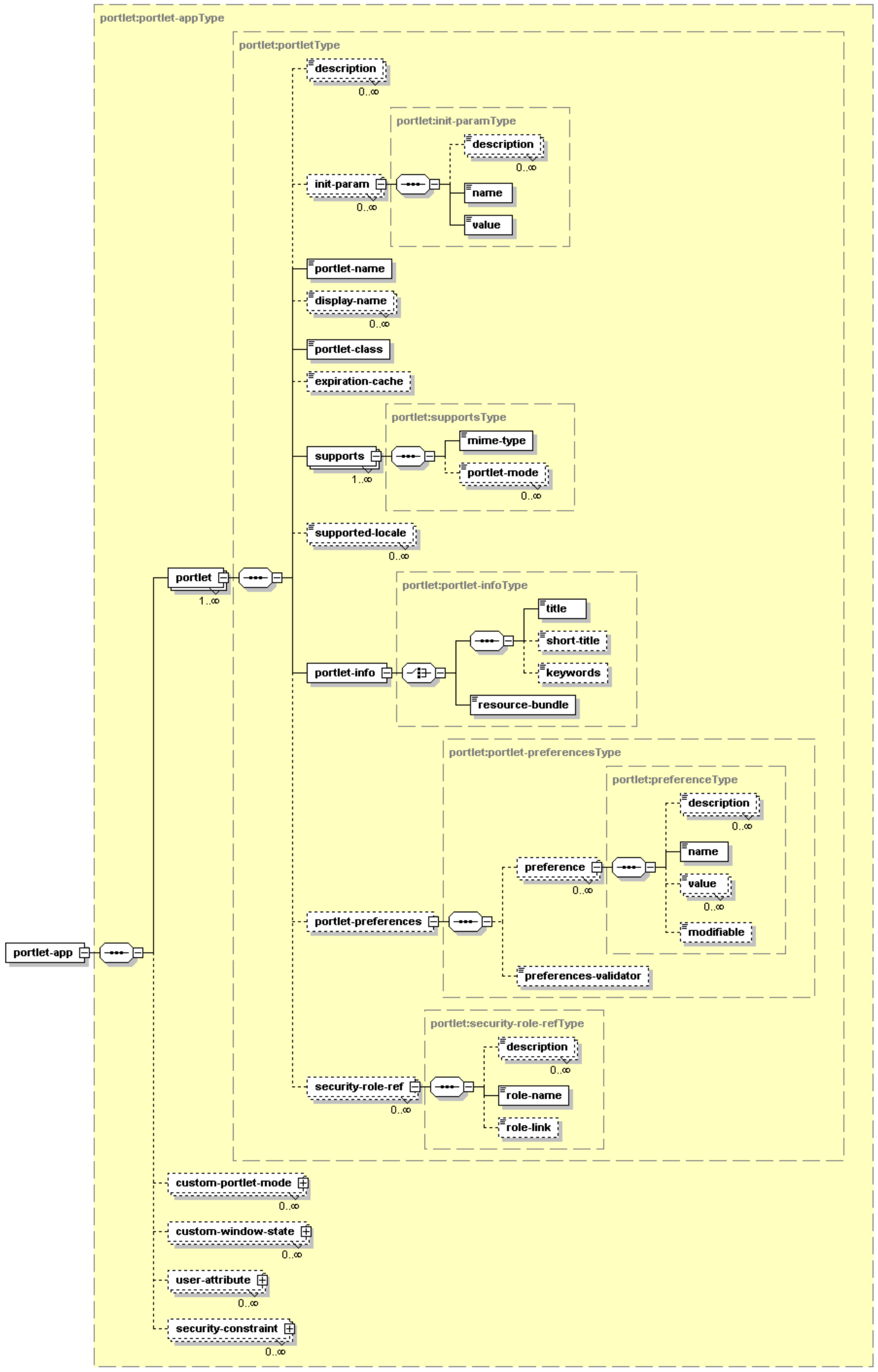
```
      <documentation>
      The display-name type contains a short name that is intended
      to be displayed by tools. It is used by display-name
      elements.  The display name need not be unique.
      Example:
        ...
        <display-name xml:lang="en">Employee Self Service</display-name>

      The value of the xml:lang attribute is "en" (English) by
      default.
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet:string">
        <attribute ref="xml:lang"/>
      </extension>
    </simpleContent>
  </complexType>
  <simpleType name="role-nameType">
    <annotation>
      <documentation>
      The role-nameType designates the name of a security role.

      The name must conform to the lexical rules for an NMTOKEN.
      </documentation>
    </annotation>
    <restriction base="NMTOKEN"/>
  </simpleType>
  <simpleType name="string">
    <annotation>
      <documentation>
      This is a special string datatype that is defined by J2EE
      as a base type for defining collapsed strings. When
      schemas require trailing/leading space elimination as
      well as collapsing the existing whitespace, this base
      type may be used.
      </documentation>
    </annotation>
    <restriction base="string">
      <whiteSpace value="collapse"/>
    </restriction>
  </simpleType>
</schema>
```

PLT.21.6 Pictures of the structure of a Deployment Descriptor




portlet:portlet-appType
portlet:portletType
description
0..∞
portlet:init-paramType
description
0..∞
init-param
0..∞
name
value
portlet-name
display-name
0..∞
portlet-class
expiration-cache
portlet:supportsType
mime-type
supports
1..∞
portlet-mode
0..∞
supported-locale
0..∞
portlet
1..∞
portlet:portlet-infoType
title
short-title
portlet-info
keywords
resource-bundle
portlet:portlet-preferencesType
portlet:preferenceType
description
0..∞
name
preference
0..∞
value
0..∞
portlet-preferences
modifiable
preferences-validator
portlet-app
portlet:security-role-refType
description
0..∞
security-role-ref
0..∞
role-name
role-link
custom-portlet-mode
0..∞
custom-window-state
0..∞
user-attribute
0..∞
security-constraint
0..∞

PLT.21.7 Uniqueness of Deployment Descriptor Values

The following deployment descriptor values must be unique in the scope of the portlet application definition:

- portlet `<portlet-name>`
- custom-portlet-mode `<portlet-mode>`
- custom-window-state `<window-state>`
- user-attribute `<name>`

The following deployment descriptor values must be unique in the scope of the portlet definition:

- init-param `<name>`
- supports `<mime-type>`
- preference `<name>`
- security-role-ref `<role-name>`

PLT.21.8 Localization

The portlet deployment descriptor allows for localization on two levels:

- Localize values needed at deployment time
- Advertise supported locales at run-time

Both are described in the following sections.

PLT.21.8.1 Localization of Deployment Descriptor Values

Localization of deployment descriptor values allows the deployment tool to provide localized deployment messages to the deployer. The following deployment descriptor elements may exist multiple times with different locale information in the `xml:lang` attribute:

- all `<description> elements`
- `portlet application <display-name>`
- portlet `<display-name>`

PLT.21.8.2 Supported Locales by the Portlet

The portlet should always declare the locales it is going to support at run-time using the `<supported-locale>` element in the deployment descriptor. ~~This will enable the portal to offer sets of portlets to users based on the preferred user locale.~~

PLT.21.9 Deployment Descriptor Example

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet" version="1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/portlet.xsd">
  <portlet>
    <description xml:lang="EN">Portlet displaying the time in different
time zones</description>
    <description xml:lang="DE">Dieses Portlet zeigt die Zeit in
verschiedenen Zeitzonen an. </description>
    <portlet-name>TimeZoneClock</portlet-name>
    <display-name xml:lang="EN">Time Zone Clock Portlet</display-name>
    <display-name xml:lang="EN">ZeitzonenPortlet</display-name>
    <portlet-class>com.myco.samplets.util.zoneclock.ZoneClock</portlet-
class>
    <expiration-cache>-1</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>config</portlet-mode>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
    </supports>
    <supports>
      <mime-type>text/wml</mime-type>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
    </supports>
    <supported-locale>EN</supported-locale>
    <portlet-info>
~~      <locale>EN</locale>~~
      <title>Time Zone Clock</title>
      <short-title>TimeZone</short-title>
      <keywords>Time, Zone, World, Clock</keywords>
    </portlet-info>
    <portlet-preferences>
      <preference>
        <description xml:lang="EN">Server URL for getting the current
time</description>
        <description xml:lang="DE">Server URL f• aktuelle
Zeit</description>
        <name>time-server</name>
        <value>http://timeserver.myco.com</value>
        <~~non-~~modifiable>0</~~non-~~modifiable>
      </preference>
      <preference>
        <description xml:lang="EN">Port number of time
server</description>
        <description xml:lang="DE">Portnummer des
Zeitservers</description>
        <name>port</name>
        <value>404</value>
        <~~non-~~modifiable>0</~~non-~~modifiable>
      </preference>
      <preference>
        <description xml:lang="EN">time format for displaying the current
time</description>
        <description xml:lang="DE">Zeitformat f angezeigte Zeit
</description>
        <name>time-format</name>
        <value>HH</value>
        <value>mm</value>
        <value>ss</value>
      </preference>
    </portlet-preferences>
    <security-role-ref>
      <role-name>trustedUser</role-name>
      <role-link>auth-user</role-link>
    </security-role-ref>
  </portlet>
  <custom-portlet-mode>
    <description xml:lang="EN">Pre-defined custom portlet mode
CONFIG</description>
    <portlet-mode>CONFIG</portlet-mode>
  </custom-portlet-mode>
  <custom-window-state>
    <description xml:lang="EN">Occupies 50% of the portal
page</description>
    <window-state>half-page</window-state>
  </custom-window-state>
  <user-attribute>
    <description xml:lang="EN">Pre-defined attribute for the telephone
number of the user at work.</description>
    <name>workInfo/telephone</name>
  </user-attribute>
  <security-constraint>
    <portlet-collection>
      <portlet-name>TimeZoneClock</portlet-name>
    </portlet-collection>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
</portlet-app>
```

PLT.21.10 Resource Bundles

To provide language specific portlet information, like title and keywords, resource bundles can be used. The file name of the resource bundle can be set in the deployment descriptor using the `resource-bundle` tag.

The Portlet Specification 1.0 defines the following constants for this resource bundle:

javax.portlet.title	The title that should be displayed in the titlebar of this portlet. Only one title per locale is allowed.
javax.portlet.short-title	A short version of the title that may be used for devices with limited display capabilities.Only one short title per locale is allowed.
javax.portlet.keywords	Keywords describing the functionality of the portlet. Portals that allow users to search for portlets based on keywords may use these keywords. Multiple keywords per locale are allowed, but must be separated by commas ‘,’.

PLT.21.11 Resource Bundle Example

This section shows the resource bundles for the world population clock portlet from deployment descriptor example. The first resource bundle is for English and the second for German locales.

```
# English Resource Bundle
#
# filename: clock_en.properties
# Portlet Info resource bundle example
javax.portlet.title=World Population Clock
javax.portlet.short-title=WorldPopClock
javax.portlet.keywords=World,Population,Clock

# German Resource Bundle
#
# filename: clock_de.properties
# Portlet Info resource bundle example
javax.portlet.title=Welt Bevoelkerung Uhr
javax.portlet.short-title=WeltbevUhr
javax.portlet.keywords=Welt,Bevoelkerung,Uhr
```

PLT.22

Portlet Tag Library

The portlet tag library enables JSPs that are included from portlets to have direct access to portlet specific elements such as the `RenderRequest` and `RenderResponse`. It also provides JSPs with access to portlet functionality such as creation of portlet URLs.

JSP pages using the tag library must declare this in a taglib like this (using the suggested prefix value):

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

PLT.22.1 defineObjects Tag

The `defineObjects` tag must define the following variables in the JSP page:

- RenderRequest renderRequest
- RenderResponse renderResponse
- PortletConfig portletConfig

These variables must reference the same portlet API objects stored in the request object of the JSP as defined in the PLT.### Included Request Attributes section.

A JSP using the `defineObjects` tag may use these variables from scriptlets throughout the page.

The `defineObjects` tag must not define any attribute and it must not support any body content.

An example of a JSP using the `defineObjects` tag could be:

```
<portlet:defineObjects/>

<%=renderResponse.setTitle("my portlet title")%>
```

After using the `defineObjects` tag, the JSP invokes the setTitle() method of the renderResponse to set the title of the portlet.

PLT.22.2 actionURL Tag

The portlet `actionURL` tag creates a URL that must point to the current portlet and must trigger an action request with the supplied parameters.

Parameters may be added to the URL by including the `param` tag between the `actionURL` start and end tags.

The following *non-required attributes* are defined for this tag:

- **state** (Type: String, non-required) – indicates the window state that the portlet should have when this link is executed. The following window states are predefined: `minimized`, `normal`, and `maximized`. If no window state is specified, or the window state is unknown/unsupported, the URL points to the current window state of the portlet. The window state attribute is not case sensitive.
- **mode** (Type: String, non-required) – indicates the portlet mode that the portlet should have when this link is executed. The following portlet modes are predefined: `edit`, `help`, and `view`. If no portlet mode is specified, or the portlet mode is unknown/unsupported, the URL points to the current portlet mode of the portlet. The portlet mode attribute is not case sensitive.
- **var** (Type: String, non-required) – name of the exported scoped variable for the action URL.By default, the result of the URL processing is written to the current `JspWriter`.If the result is exported as a JSP scoped variable, defined via the `var` attributes., nothing is written to the current `JspWriter`. Note: After the URL is created it is not possible to extend the URL or add any further parameter using the variable and String concatenation.

An example of a JSP using the `actionURL` tag could be:

```
<portlet:actionURL state="maximized" mode="edit">
   <portlet:param name="action" value="editStocks"/>
</portlet:actionURL>
```

The example creates a URL that brings the portlet into `EDIT` mode and `MAXIMIZED` window state to edit the stocks quote list.

PLT.22.3 renderURL Tag

The portlet `tenderURL` tag must create a URL that pointing to the current portlet and must trigger a render request with the supplied parameters.

Parameters may be added by including the `param` tag between the `renderURL` start and end tags.

The following *non-required attributes* are defined for this tag:

- **state** (Type: String, non-required) – indicates the window state that the portlet should have when this link is executed. The following window states are predefined: `minimized`, `normal`, and `maximized`. If no window state is specified, or the specified window state is unkown/unsupported, the URL points to the current window state of the portlet.
- **mode** (Type: String, non-required) – indicates the portlet mode that the portlet should have when this link is executed. The following portlet modes are predefined: `edit`, `help`, and `view`. If no portlet mode is specified, or the portlet mode is unknown/unsupported, the URL points to the current portlet mode of the portlet.
- **var** (Type: String, non-required) – name of the exported scoped variable for the render URL.By default, the result of the URL processing is written to the current `JspWriter`.If the result is exported as a JSP scoped variable, defined via the `var` attributes., nothing is written to the current `JspWriter`. Note: After the URL is created it is not possible to extend the URL or add any further parameter using the variable and String concatenation.

An example of a JSP using the `renderURL` tag could be:

```
<portlet:renderURL mode=”view” state=”normal”>
   <portlet:param name=”showQuote” value=”myCompany”/>
   <portlet:param name=”showQuote” value=”someOtherCompany”/>
</portlet:renderURL>
```

The example creates a URL to provide a link that shows the stock quote of myCompany and someOtherCompany and changes the portlet mode to `VIEW` and the window state to `NORMAL`.

PLT.22.4 encode Tag

This tag must encodes the given string value to the namespace of the current portlet.

This tag should be used for named elements in the portlet output (for example, form fields or Javascript variables). The encoding ensures that the given name is uniquely associated with this portlet and avoids name conflicts with other elements on the portal page or with other portlets on the page.

The `encode` tag must not support any body content.

The following *required attribute* is defined for this tag:

- **name** (Type: String, required) – the name of the String that should be encoded into the namespace of the portlet.

An example of a JSP using the `encode` tag could be:

```
<input border="0"
       type="text"
       name="<portlet:encode name='email' />">
```

The example creates a text field with the name ‘email’, which is encoded to ensure uniqueness on the portal page.

PLT.22.5 param Tag

This tag defines a parameter that may be added to a `actionURL` or `renderURL`.

The `param` tag must not support any body content.

The following *required attributes* are defined for this tag:

- **name** (Type: String, required) – the name of the parameter to add to the URL. If `name` is null or empty, no action is performed.
- **value** (Type: String, required) – the value of the parameter to add to the URL. If `value` is null, it is processed as an empty value.

An example of a JSP using the `param` tag could be:

```
<portlet:param name="myParam" value="someValue"/>
```

PLT.23

Technology Compatibility Kit Requirements

This chapter defines a set of requirements a portlet container implementation must meet in order to run the portlet Technology Compatibility Kit (TCK).

These requirements are only needed for the purpose of determining whether a portlet container implementation complies with the Portlet Specification or not.

PLT.23.1 TCK Test Components

Based on the Portlet Specification (this document) and the portlet API, a set of testable assertions have been extracted and identified. The portlet TCK treats each testable assertion as a unique test case.

All test cases are run from a Java Test Harness. The Java Test Harness collects the results of all the tests and makes a report on the overall test.

Each portlet TCK test case has two components:

- Test portlet applications: These are portlet applications containing portlets, servlets or JSPs coded to verify an assertion. These test portlet applications are deployed in the portlet container being tested for compliance.
- Test client: It is a standalone java program that sends HTTP requests to portlet container where test portlet applications of the test case have been deployed for compliance testing.

The portlet TCK assumes that the test portlet applications are deployed in the portlet container before the test run is executed.

The test client looks for expected and unexpected sub strings in the HTTP response to decide whether a test has failed or passed. The test client reports the result of the test client to the Java Test Harness.

PLT.23.2 TCK Requirements

In TCK, every test is written as a set of one or more portlets. A test client is written for each test, the test client must interact with a portal page containing the portlets that are part of the test. To accomplish this, TCK needs to obtain the initial URL for the portal page of each test case. All the portlets in the portal page obtained with the initial URL must be in VIEW portlet mode and in NORMAL window state. Subsequent requests to the test are done using URLs generated by PortletURI that are part of the returned portal pages. These subsequent requests must be treated as directed to same portal page composed of the same portlets.

Portal/portlet-containers must disable all caching mechanisms when running the TCK test cases.

Since aggregation of portlets in a portal page and the URLs used to interact with the portlets are vendor specific, TCK provides two alternative mechanisms in the framework to get the URLs to portal pages for the test cases: declarative configuration or programmatic configuration. A vendor must support at least one of these mechanisms to run the conformance tests.

PLT.23.2.1 Declarative configuration of the portal page for a TCK test

TCK publishes an XML file containing the portlets for each test case. Vendors must refer to this file for establishing a portal page for every test. Vendors must provide an XML file with a full URL for the portal page for each test. A call to this URL must generate a portal page with the content of all the portlets defined for the corresponding test case. If redirected to another URL, the new URL must use the same host name and port number as specified in the file. Refer to TCK User guide for details on declarative configuration.

A snippet of the TCK provided XML file for declarative configuration would look like:

```
<test_case>
  <test_name>PortletRequest_GetAttributeTest</test_name>
  <test_portlet>
    <app_name>PortletRequestWebApp</app_name>
    <portlet_name>GetAttributeTestPortlet</portlet_name>
  </test_portlet>
  <test_portlet>
    <app_name>PortletRequestWebApp</app_name>
    <portlet_name>GetAttributeTest_1_Portlet</portlet_name>
  <test_portlet>
</test_case>
```

The corresponding snippet for the vendor’s provided XML file might look like:

```
<test_case_url>
  <test_name>PortletRequest_GetAttributeTest</test_name>
  <test_url>http://foo:8080/portal?pageName=TestCase1</test_url>
</test_case_url>
```

PLT.23.2.1.1 Schema for XML file provided with Portlet TCK

```
<?xml version="1.0" encoding="UTF-8"?>
<!—portletTCKTestCases.xsd-->
<xs:schema targetNamespace="http://java.sun.com/xml/ns/portletTCK"
xmlns:pct="http://java.sun.com/xml/ns/portletTCK"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="pct_test_cases">
    <xs:annotation>
      <xs:documentation>Test Cases defined in Portlet Compatibility
Kit</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pct:test_case" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="test_case">
    <xs:annotation>
      <xs:documentation>Test Case</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pct:test_name"/>
        <xs:element ref="pct:test_portlet" minOccurs="1"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="test_portlet">
    <xs:annotation>
      <xs:documentation>A test Portlet</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pct:portlet_name"/>
        <xs:element ref="pct:app_name"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="test_name" type="xs:string">
    <xs:annotation>
      <xs:documentation>Unique name for a test case</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="app_name" type="xs:string">
    <xs:annotation>
      <xs:documentation>Name of the portlet application a portlet belongs
to.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="portlet_name" type="xs:string">
    <xs:annotation>
      <xs:documentation>Name of the portlet</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

PLT.23.2.1.2 Schema for XML file that provided by vendors

```
<?xml version="1.0" encoding="UTF-8"?>
<!—portletTCKTestURLs.xsd - Schema that must be followed by the vendors to write
the file that has mapping from a portlet TCK -->
<!-- test case to a url. -->
<xs:schema targetNamespace="http://java.sun.com/xml/ns/portletTCK"
xmlns:pct="http://java.sun.com/xml/ns/portletTCK"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="test_case_urls">
    <xs:annotation>
      <xs:documentation>Mapping of Test Cases defined in Portlet Compatibility
Kit to vendor specific URLs</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pct:test_case_url" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="test_case_url">
    <xs:annotation>
      <xs:documentation>Test Case to URL map entry </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pct:test_name"/>
        <xs:element ref="pct:test_url"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="test_name" type="xs:string">
    <xs:annotation>
      <xs:documentation>Unique name for a test case from the
portletTCKTestCases.xml published by TCK</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="test_url" type="xs:string">
    <xs:annotation>
      <xs:documentation>Complete URL that would result in a page containing
contents of portlets defined for this test case.</xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:schema>
```

PLT.23.2.2 Programmatic configuration of the portal page for a test

For programmatic configuration, a vendor must provide a full URL as a configuration parameter to the TCK. The TCK will call this URL with a set of parameters indicating the set of portlets that must appear in a portal page for the given test. Upon receiving this request, the vendor provided URL could dynamically create a portal page with the required portlets. Calls to this vendor provided URL are always HTTP GET requests. The parameter names on the URL are multiple occurrences of "*portletName*". Values of this paramater must be a string consisting of the test case application name and portlet name delimited by a “/”. The response of this call must be a portal page with the required portlets or a redirection to another URL where the portal page will be served. If redirected, the new URL must use the same host and port number as original URL.

A vendor provided URL would look like:

```
VendorPortalURL=http://foo:8080/portal/tckservlet
```

For a test case involving one portlet, TCK would call this URL with the following parameters:

```
http://foo:8080/portal/tckservlet?portletName=PortletRequestWebApp
/GetAttributeTestPortlet
```

PLT.23.2.3 Test Portlets Content

The test cases portlets encode information for the test client within their content. As different vendor implementations may generate different output surrounding the content produced by the portlets, the portlets delimit the information for the test clients using a special element tag, `portlet-tck`.

PLT.23.2.4 Test Cases that Require User Identity

Some of the Portlet TCK require an authenticated user.The TCK configuration file indicates the name and password of the authenticated user and the authentication mechanism TCK will use.

Portlet TCK provides two mechanisms to send the user credentials: HTTP Basic authentication and a Java interface provided by the TCK. If TCK framework is configured to use HTTP Basic authentication, an `Authorization` HTTP header -using the configured user and password values- is constructed and sent with each test case request. If TCK framework is configured to use the Java interface mechanism, the value obtained from the specified interface implementation will be sent as a Cookie HTTP header with request of the test case.

Additionally, a portal vendor may indicate that certain test cases, not required by TCK, to be executed in the context of an authenticated user. This is useful for vendor implementations that require an authenticated user for certain functionality to work. A vendor can specify the names of these test cases in a configuration file. TCK will consult this file to decide if user authentication is needed for each test case. Refer to TCK User Guide to get details on the specific configuration properties.

.

PLT.A

Custom Portlet Modes

Portals may provide support for custom portlet modes. Similarly, portlets may use custom portlet modes. This appendix describes a list of custom portlet modes and their intended functionality. Portals and portlets should use these custom portlet mode names if they provide support for the described functionality.

Portlets should use the `getSupportedPortletModes` method of the `PortalContext` interface to retrieve the portlet modes the portal supports.

PLT.A.1 About Portlet Mode

The `about` portlet mode should be used by the portlet to display information on the portlets purpose, origin, version etc.

Portlet developers should implement the `about` portlet mode functionality by overriding the `doDispatch` method of the `GenericPortlet` class and checking for `PortletMode("about")`.

In the deployment descriptor the support for the `about` portlet mode must be declared using

```
<portlet-app>
  ...
  <portlet>
    ...
    <supports>
      ...
      <portlet-mode>about</portlet-mode>
    </supports>
    ...
  </portlet>
  ...
  <custom-portlet-mode>
    <name>about</name>
  </custom-portlet-mode>
  ...
</portlet-app>
```

PLT.A.2 Config Portlet Mode

The `config` portlet mode should be used by the portlet to display one or more configuration views that let administrators configure portlet preferences that are marked non-modifiable in the deployment descriptor. This requires that the user must have administrator rights. Therefore, only the portal can create links for changing the portlet mode into `config`.

Portlet developers should implement the `config` portlet mode functionality by overriding the `doDispatch` method of the `GenericPortlet` class and checking for `PortletMode("config")`.

The `CONFIG` mode of portlets operates typically on shared state that is common to many portlets of the same portlet definition. When a portlet modifies this shared state via the PortletPreferences, for all affected portlet entities, in the `doView` method the `PortletPreferences` must give access to the modified state.

In the deployment descriptor the support for the `config` portlet mode must be declared using

```
<portlet-app>
  ...
  <portlet>
    ...
    <supports>
      ...
      <portlet-mode>config</portlet-mode>
    </supports>
    ...
  </portlet>
  ...
  <custom-portlet-mode>
    <name>config</name>
  </custom-portlet-mode>
  ...
  </portlet-app>
```

PLT.A.3 Edit_defaults Portlet Mode

The `edit_defaults` portlet mode signifies that the portlet should render a screen to set the default values for the modifiable preferences that are typically changed in the EDIT screen. Calling this mode requires that the user must have administrator rights. Therefore, only the portal can create links for changing the portlet mode into `edit_defaults`.

Portlet developers should implement the `edit_defaults` portlet mode functionality by overriding the `doDispatch` method of the `GenericPortlet` class and checking for `PortletMode("edit_defaults ")`.

In the deployment descriptor the support for the `edit_defaults` portlet mode must be declared using

```
<portlet-app>
  ...
  <portlet>
    ...
    <supports>
      ...
      <portlet-mode> edit_defaults </portlet-mode>
    </supports>
    ...
  </portlet>
  ...
  <custom-portlet-mode>
    <name> edit_defaults </name>
  </custom-portlet-mode>
  ...
  </portlet-app>
```

PLT.A.4 Preview Portlet Mode

The `preview` portlet mode should be used by the portlet to render output without the need of having back-end connections or user specific data available. It may be used at page design time and in portlet development tools.

Portlet developers should implement the `preview` portlet mode functionality by overriding the `doDispatch` method of the `GenericPortlet` class and checking for `PortletMode("preview ")`.

In the deployment descriptor the support for the `preview` portlet mode must be declared using

```
<portlet-app>
  ...
  <portlet>
    ...
    <supports>
      ...
      <portlet-mode> preview </portlet-mode>
    </supports>
    ...
  </portlet>
  ...
  <custom-portlet-mode>
    <name> preview </name>
  </custom-portlet-mode>
  ...
  </portlet-app>
```

PLT.A.5 Print Portlet Mode

The `print`portlet mode signifies that the portlet should render a view that can be printed.

Portlet developers should implement the `print`portlet mode functionality by overriding the `doDispatch` method of the `GenericPortlet` class and checking for `PortletMode("print")`.

In the deployment descriptor the support for the `print`portlet mode must be declared using

```
<portlet-app>
  ...
  <portlet>
    ...
    <supports>
      ...
      <portlet-mode>print</portlet-mode>
    </supports>
    ...
  </portlet>
  ...
  <custom-portlet-mode>
    <name>print</name>
  </custom-portlet-mode>
  ...
  </portlet-app>
```

PLT.B

Markup Fragments

Portlets generate markup fragments that are aggregated in a portal page document. Because of this, there are some rules and limitations in the markup elements generated by portlets. Portlets should conform to these rules and limitations when generating content.

The disallowed tags indicated below are those tags that impact content generated by other portlets or may even break the entire portal page. Inclusion of such a tag invalidates the whole markup fragment.

Portlets generating HTML fragments must not use the following tags: `base`, `body`, `frame`, `frameset`, `head`, `html` and `title`.

Portlets generating XHTML and XHTML-Basic fragments must not use the following tags: `base`, `body`, `head`, `html` and `title`.

HTML, XHTML and XHTML-Basic specifications disallow the use of certain elements outside of the <head> element in the document. However, some browser implementations support some of these tags in other sections of the document. For example: current versions of Internet Explorer and Netscape Navigator both support the style tag anywhere within the document. Portlet developers should decide carefully the use of following markup elements that fit this description: `link`, `meta` and `style`.

PLT.C

CSS Style Definitions

To achieve a common look and feel throughout the portal page, all portlets in the portal page should use a common CSS style sheet when generating content.

This appendix defines styles for a variety of logical units in the markup. It follows the style being considered by the OASIS Web Services for Remote Portlets Technical Committee.

PLT.C.1 Links (Anchor)

A custom CSS class is not defined for the <a> tag. The entity should use the default classes when embedding anchor tags.

PLT.C.2 Fonts

The font style definitions affect the font attributes only (font face, size, color, style, etc).

Style	Description	Example
portlet-font	Font attributes for the "normal" fragment font. Used for the display of non-accentuated information.	Normal Text
portlet-font-dim	Font attributes similar to the .portlet.font but the color is lighter.	Dim Text

If an portlet developer wants a certain font type to be larger or smaller, they should indicate this using a relative size. For example:

```
<div class="portlet-font" style="font-size:larger">Important
information</div>

<div class="portlet-font-dim" style="font-size:80%">Small and
dim</div>
```

PLT.C.3 Messages

Message style definitions affect the rendering of a paragraph (alignment, borders, background color, etc) as well as text attributes.

Style	**Description**	**Example**
portlet-msg-status	Status of the current operation.	*Progress: 80%*
portlet-msg-info	Help messages, general additional information, etc.	Info about
portlet-msg-error	Error messages.	Portlet not available
portlet-msg-alert	Warning messages.	*Timeout occurred, try again later*
portlet-msg-success	Verification of the successful completion of a task.	**Operation completed successfully**

PLT.C.4 Sections

Section style definitions affect the rendering of markup sections such as table, div and span (alignment, borders, background color, etc) as well as their text attributes.

Style	**Description**
portlet-section-header	Table or section header
portlet-section-body	Normal text in a table cell
portlet-section-alternate	Text in every other row in the cell
portlet-section-selected	Text in a selected cell range
portlet-section-subheader	Text of a subheading
portlet-section-footer	Table or section footnote
portlet-section-text	Text that belongs to the table but does not fall in one of the other categories (e.g. explanatory or help text that is associated with the section).

PLT.C.5 Forms

Form styles define the look-and-feel of the elements in an HTML form.

Style	Description
portlet-form-label	Text used for the descriptive label of the whole form (not the labels for fields.
portlet-form-input-field	Text of the user-input in an input field.
portlet-form-button	Text on a button
portlet-icon-label	Text that appears beside a context dependent action icon.
portlet-dlg-icon-label	Text that appears beside a “standard” icon (e.g. Ok, or Cancel)
portlet-form-field-label	Text for a separator of fields (e.g. checkboxes, etc.)
portlet-form-field	Text for a field (not input field, e.g. checkboxes, etc)

PLT.C.6 Menus

Menu styles define the look-and-feel of the text and background of a menu structure. This structure may be embedded in the aggregated page or may appear as a context sensitive popup menu.

Style	Description
portlet-menu	General menu settings such as background color, margins, etc
portlet-menu-item	Normal, unselected menu item.
portlet-menu-item-selected	Selected menu item.
portlet-menu-item-hover	Normal, unselected menu item when the mouse hovers over it.
portlet-menu-item-hover-selected	Selected menu item when the mouse hovers over it.
portlet-menu-cascade-item	Normal, unselected menu item that has sub-menus.
portlet-menu-cascade-item-selected	Selected sub-menu item that has sub-menus.
portlet-menu-description	Descriptive text for the menu (e.g. in a help context below the menu)
portlet-menu-caption	Menu caption

User Information Attribute Names

This appendix defines a set of attribute names for user information and their intended meaning. To allow portals an automated mapping of commonly used user information attributes portlet programmers should use these attribute names. These attribute names are derived from the Platform for Privacy Preferences 1.0 (P3P 1.0) Specification by the W3C (http://www.w3c.org/TR/P3P). ~~Similar~~ The same attribute names are also being considered by the OASIS Web Services for Remote Portlets Technical Committee.

Attribute Name
user.bdate
user.gender
user.employer
user.department
user.jobtitle
user.name.prefix
user.name.given
user.name.family
user.name.middle
user.name.suffix
user.name.nickName
user.home-info.postal.name
user.home-info.postal.street
user.home-info.postal.city
user.home-info.postal.stateprov
user.home-info.postal.postalcode
user.home-info.postal.country
user.home-info.postal.organization
user.home-info.telecom.telephone.intcode
user.home-info.telecom.telephone.loccode
user.home-info.telecom.telephone.number
user.home-info.telecom.telephone.ext
user.home-info.telecom.telephone.comment
user.home-info.telecom.fax.intcode
user.home-info.telecom.fax.loccode
user.home-info.telecom.fax.number
user.home-info.telecom.fax.ext
user.home-info.telecom.fax.comment
user.home-info.telecom.mobile.intcode
user.home-info.telecom.mobile.loccode
user.home-info.telecom.mobile.number
user.home-info.telecom.mobile.ext
user.home-info.telecom.mobile.comment
user.home-info.telecom.pager.intcode

user.home-info.telecom.pager.loccode
user.home-info.telecom.pager.number
user.home-info.telecom.pager.ext
user.home-info.telecom.pager.comment
user.home-info.online.email
user.home-info.online.uri~~l~~
user.business-info.postal.name
user.business-info.postal.street
user.business-info.postal.city
user.business-info.postal.stateprov
user.business-info.postal.postalcode
user.business-info.postal.country
user.business-info.postal.organization
user.business-info.telecom.telephone.intcode
user.business-info.telecom.telephone.loccode
user.business-info.telecom.telephone.number
user.business-info.telecom.telephone.ext
u~~U~~ser.business-info.telecom.telephone.comment
user.business-info.telecom.fax.intcode
user.business-info.telecom.fax.loccode
user.business-info.telecom.fax.number
user.business-info.telecom.fax.ext
~~U~~user.business-info.telecom.fax.comment
user.business-info.telecom.mobile.intcode
user.business-info.telecom.mobile.loccode
user.business-info.telecom.mobile.number
user.business-info.telecom.mobile.ext
~~U~~user.business-info.telecom.mobile.comment
user.business-info.telecom.pager.intcode
user.business-info.telecom.pager.loccode
user.business-info.telecom.pager.number
user.business-info.telecom.pager.ext
~~U~~user.business-info.telecom.pager.comment
~~U~~user.business-info.online.email
~~U~~user.business-info.online.uri~~l~~

NOTE: The `user.bdate` must consist of a string that represents the time in milliseconds since January 1, 1970, 00:00:00 GMT.

PLT.D.1 Example

Below is an example of how these attributes may be used in the deployment descriptor:

```
<portlet-app>
  ...
  <user-attribute>
     <name> user.name.prefix</name>
  </user-attribute>
  <user-attribute>
     <name> user.name.given</name>
  </user-attribute>
  <user-attribute>
     <name> user.name.family</name>
  </user-attribute>
  <user-attribute>
     <name> user.home-info.postal.city</name>
  </user-attribute>
  ...
<.portlet-app>
```

PLT.E

TCK Assertions

The following is the list of assertions that have been identified in the Portlet Specification for the purposes of the compliance test.

Assertions marked as Testable=false are not verifiable.

[i] SPEC:1	Testable=false	Section=PLT.5.1
[ii] SPEC:2	Testable=false	Section=PLT.5.1
[iii] SPEC:3	Testable=false	Section=PLT.5.2.1
[iv] SPEC:4	Testable=true	Section=PLT.5.2.2
[v] SPEC:5	Testable=true	Section=PLT.5.2.2.1
[vi] SPEC:6	Testable=true	Section=PLT.5.2.2.1
[vii] SPEC:7	Testable=true	Section=PLT.5.2.2.1
[viii] SPEC:8	Testable=true	Section=PLT.5.2.2.1
[ix] SPEC:9	Testable=true	Section=PLT 5.2.4
[x] SPEC:10	Testable=true	Section=PLT 5.2.4
[xi] SPEC:11	Testable=true	Section=PLT 5.2.4.1
[xii] SPEC:12	Testable= true	Section=PLT.5.2.4.1
[xiii] SPEC:13	Testable= true	Section=PLT.5.2.4.1
[xiv] SPEC:14	Testable= true	Section=PLT.5.2.4.2.1

[xv] SPEC:15	Testable= true	Section=PLT.5.2.4.2.1
[xvi] SPEC:16	Testable= true	Section=PLT.5.2.4.2.1
[xvii] SPEC:17	Testable=true	Section=PLT 5.2.4.2.1
[xviii] SPEC:18	Testable= true	Section=PLT.5.2.4.4
[xix] SPEC:19	Testable=false	Section=PLT.5.2.4.4
[xx] SPEC:20	Testable= true	Section=PLT.5.2.4.4.
[xxi] SPEC:21	Testable=false	Section=PLT/5.2.5
[xxii] SPEC:22	Testable= false	Section=PLT.5.2.5
[xxiii] SPEC:23	Testable=false	Section=PLT.5.2.5
[xxiv] SPEC:24	Testable= false	Section=PLT.5.2.5
[xxv] SPEC:25	Testable= true	Section=PLT.6.2
[xxvi] SPEC:26	Testable= true	Section=PLT.6.2
[xxvii] SPEC:27	Testable= true	Section=PLT.6.2
[xxviii] SPEC:28	Testable= true	Section=PLT.6.2
[xxix] SPEC:29	Testable= true	Section=PLT.6.2
[xxx] SPEC:30	Testable= true	Section=PLT.6.2
[xxxi] SPEC:31	Testable= true	Section=PLT.7.1.1
[xxxii] SPEC:32	Testable= true	Section=PLT.7.1.1
[xxxiii] SPEC:33	Testable= true	Section=PLT.6.2
[xxxiv] SPEC:34	Testable=true	Section=PLT.8.5
[xxxv] SPEC:35	Testable=true	Section=PLT.8.6
[xxxvi] SPEC:36	Testable=true	Section=PLT.8.6
[xxxvii] SPEC:37	Testable=false	Section=PLT.8.6
[xxxviii] SPEC:38	Testable=true	Section=PLT.9.4

[xxxix] SPEC:39	Testable=false	Section=PLT.10.1
[xl] SPEC:40	Testable=false	Section=PLT.10.1
[xli] SPEC:41	Testable=true	Section=PLT.10.3
[xlii] SPEC:42	Testable=true	Section=PLT.10.3
[xliii] SPEC:43	Testable=true	Section=PLT.10.3
[xliv] SPEC:44	Testable=true	Section=PLT.10.3
[xlv] SPEC:45	Testable=true	Section=PLT.10.3(servlet spec)
[xlvi] SPEC:46	Testable=true	Section=PLT.11.1.1
[xlvii] SPEC:47	Testable= true	Section=PLT.11.1.1
[xlviii] SPEC:48	Testable=true	Section=PLT.11.1.1
[xlix] SPEC:49	Testable=true	Section=PLT.11.1.1
[l] SPEC:50 Testable=false SPEC:28 Testable= true		Section=PLT.11.1.1
[li] SPEC:51	Testable=true	Section=PLT.11.1.1
[lii] SPEC:52	Testable=true	Section=PLT.11.1.1
[liii] SPEC:53	Testable=true	Section=PLT.11.1.1
[liv] SPEC:54	Testable=false	Section=PLT.11.1.2
[lv] SPEC:55	Testable=true	Section=PLT.11.1.5
[lvi] SPEC:56	Testable=true	Section=PLT.11.1.5
[lvii] SPEC:57	Testable=true	Section=PLT.11.1.6
[lviii] SPEC:58	Testable=true	Section=PLT.11.1.7
[lix] SPEC:59	Testable= true	Section=PLT.11.1.9
[lx] SPEC:60	Testable= true	Section=PLT.11.1.9
[lxi] SPEC:61	Testable= true	Section=PLT.11.1.10
[lxii] SPEC:62	Testable= true	Section=PLT.11.1.11

[lxiii] SPEC:63	Testable=true	Section=PLT.11.2.1
[lxiv] SPEC:64	Testable=true	Section=PLT.11.2.1
[lxv] SPEC:84	Testable=true	Section=PLT.12.3.4
[lxvi] SPEC:65	Testable=true	Section=PLT.12.2.1
[lxvii] SPEC:66	Testable=true	Section=PLT.12.2.1
[lxviii] SPEC:67	Testable=true	Section=PLT.12.2.2
[lxix] SPEC:68	Testable=true	Section=PLT.12.2.2
[lxx] SPEC:69	Testable= true	Section=PLT.12.2.2
[lxxi] SPEC:70	Testable= true	Section=PLT.12.2.2
[lxxii] SPEC:71	Testable=true	Section=PLT.12.2.2
[lxxiii] SPEC:72	Testable=true	Section=PLT.12.2.3
[lxxiv] SPEC:73	Testable=true	Section=PLT.12.3.1
[lxxv] SPEC:74	Testable=true	Section=PLT.12.3.1
[lxxvi] SPEC:75	Testable= true	Section=PLT.12.3.1
[lxxvii] SPEC:76	Testable= true	Section=PLT.12.3.2
[lxxviii] SPEC:77	Testable=true	Section=PLT.12.3.3
[lxxix] SPEC:78	Testable=true	Section=PLT.12.3.3
[lxxx] SPEC:79	Testable=true	Section=PLT.12.3.3
[lxxxi] SPEC:80	Testable=true	Section=PLT.12.3.3
[lxxxii] SPEC:81	Testable=true	Section=PLT.12.3.3
[lxxxiii] SPEC:82	Testable=true	Section=PLT.12.3.3
[lxxxiv] SPEC:83	Testable=true	Section=PLT.12.3.3
[lxxxv] SPEC:84	Testable=true	Section=PLT.12.3.4
[lxxxvi] SPEC:85	Testable=false	Section=PLT.12.3.5

lxxxvii SPEC:86	Testable=true	Section=PLT.14.1
lxxxviii SPEC:87	Testable=true	Section=PLT.14.1
lxxxix SPEC:88	Testable=true	Section=PLT.14.1
xc SPEC:89	Testable=true	Section=PLT.14.1
xci SPEC:90	Testable=true	Section=PLT.14.1
xcii SPEC:91	Testable= true	Section=PLT.14.1
xciii SPEC:92	Testable=true	Section=PLT.14.1
xciv SPEC:93	Testable=true	Section=PLT.14.3
xcv SPEC:94	Testable=true	Section=PLT.14.3
xcvi SPEC:95	Testable=false	Section=PLT.14.4
xcvii SPEC:96	Testable=false	Section=PLT.14.4
xcviii SPEC:97	Testable=true	Section=PLT.14.4
xcix SPEC:98	Testable=true	Section=PLT.14.4
c SPEC:99	Testable=true	Section=PLT.14.4
ci SPEC:100	Testable=true	Section=PLT.15.1
cii SPEC:101	Testable=true	Section=PLT.15.1
ciii SPEC:102	Testable=true	Section=PLT.15.2
civ SPEC:103	Testable=true	Section=PLT.15.2
cv SPEC:104	Testable=true	Section=PLT.15.3
cvi SPEC:105	Testable=true	Section=PLT.15.3
cvii SPEC:106	Testable=true	Section=PLT.15.3
cviii SPEC:107	Testable=true	Section=PLT.15.4
cix SPEC:108	Testable=true	Section=PLT.15.4
cx SPEC:109	Testable=true	Section=PLT.15.4

cxi SPEC:110	Testable=true	Section=PLT.15.4
cxii SPEC:111	Testable=true	Section=PLT.15.4.1
cxiii SPEC:112	Testable=true	Section=PLT.15.4.1
cxiv SPEC:113	Testable=true	Section=PLT.15.4.1
cxv SPEC:114	Testable=true	Section=PLT.15.8(servlet spec)
cxvi SPEC:115	Testable=true	Section=PLT.16.1
cxvii SPEC:116	Testable=true	Section=PLT.16.1
cxviii SPEC:117	Testable=true	Section=PLT.16.1.1(servlet spec)
cxix SPEC:118	Testable=true	Section=PLT.16.2
cxx SPEC:119	Testable=true	Section=PLT.16.2
cxxi SPEC:120	Testable=true	Section=PLT.16.3.1
cxxii SPEC:121	Testable=true	Section=PLT.16.3.2
cxxiii SPEC:122	Testable=true	Section=PLT.16.3.3
cxxiv SPEC:123	Testable=true	Section=PLT.16.3.3
cxxv SPEC:124	Testable=true	Section= PLT.16.3.3
cxxvi SPEC:125	Testable=false	Section=PLT.16.3.3
cxxvii SPEC:126	Testable=true	Section= PLT.16.3.3
cxxviii SPEC:128	Testable=true	Section= PLT.16.3.3
cxxix SPEC:129	Testable=true	Section= PLT.16.3.3
cxxx SPEC:130	Testable=true	Section= PLT.16.3.3
cxxxi SPEC:131	Testable=false(impl)	Section= PLT.16.3.3
cxxxii SPEC:132	Testable=true	Section= PLT.16.3.3
cxxxiii SPEC:133	Testable=true	Section=PLT.16.3.4
cxxxiv SPEC:134	Testable=true	Section=PLT.16.3.4

[cxxxv] SPEC:135	Testable=false(impl)	Section=PLT.17.1
[cxxxvi] SPEC:136	Testable= false(impl)	Section=PLT.17.2
[cxxxvii] SPEC:137	Testable= false(impl)	Section=PLT.17.2
[cxxxviii]SPEC:138	Testable= false	Section= PLT.19.2
[cxxxix] SPEC:139	Testable= false	Section= PLT.19.2
[cxl] SPEC:140	Testable=false	Section= PLT.19.5
[cxli] SPEC:141	Testable=true	Section=PLT.19.5(servlet spec)
[cxlii] SPEC:142	Testable=true	Section= PLT.20.2
[cxliii] SPEC:143	Testable=true	Section= PLT.20.2
[cxliv] SPEC:144	Testable=true	Section= PLT.20.2
[cxlv] SPEC:145	Testable=true	Section= PLT.20.4
[cxlvi] SPEC:146	Testable=true	Section= PLT.20.4