Java™ Portlet Specification

Version 2.0 Early Draft 1

Send comments about this document to: `jsr-286-comments@jcp.org`

~~May~~ July ~~17~~10 19, 200~~65~~

Stefan Hepper (sthepper@de.ibm.com)

Java(TM) Portlet Specification ("Specification") Version: 2.0 Status: Early Draft, Specification Lead: IBM Corp.

Contents

PLT.1

Preface

This document is the Java™ Portlet Specification, v~~1~~2.~~.~~0. The standard for the Java™ Portlet API is described here.

NOTE: This first early draft only covers a subset of features that the Expert Group wanted to add in this JSR. The features that are part of this early draft addresses the often raised requirement for more coordination support between portlets and the new Web Services for Remote Portlets (WSRP) 2.0 features. Additional features, like a better AJAX support, will be added in the next draft version.

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~~ containers 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~~ 4 (J2EE™)
- Java Servlet™, v2.~~3~~4
- JavaServer Pages™, v~~1.2~~2.0 (JSP™)
- The Java™ Architecture for XML Binding (JAXB) 2.0

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 ~~1776~~ 1766 Tags for the Identification of Languages
- 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
- OASIS Web Services for Remote Portlets (WSRP)

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.iso.org/iso/en/prods-services/iso3166ma/index.html

- ~~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 WSRP Specification can be found in the OASIS web site (http://www.oasis-open.org/).

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 jsr-~~1681~~286-comments@~~sun.com~~jcp.org.

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 Portlet Specification is the result of the work of ~~JSR168~~ JSR2868 Expert Group.~~,~~ ~~Subbu Allamaraju (BEA), Chris Braun (Novell), Don Chapman (SAS), Michael Freedman (Oracle), Laurent Guiraud (SAP), Randal Hanford (Boeing), Andre Kramer (Citrix), Axel Kratel (Borland), Danny Machak (TIBCO), Kris Meukens (EDS), Wes Mitchell (Broadvision), Takao Mohri (Fujitsu), Dean Moses (Vignette), Andrew Rickard (ATG), William Seiger (Sybase), David Sean Taylor (Apache), Stefan Hepper (IBM) and Alejandro Abdelnur (Sun).~~

~~We want to give special thanks to (as members of the Expert Group) Subbu Allamaraju, Henning Blohm, Chris Braun, Don Chapman, Adrian Fletcher, Michael Freedman, Laurent Guiraud, Andre Kramer, Danny Machak, Wes Mitchell, Takao Mohri, Dean Moses, Peter Petersen, Andrew Rickard and David Sean Taylor for their contributions.~~

~~We would like to thank OASIS WSRP Technical Committee, JSR127 Java Server Faces Expert Group and JSR154 Servlet Specification Expert Group for their cooperation.~~

~~We would also like to thank all the people who have sent us feedback during the Community Review and Public Review stages.~~

~~Finally we would like to thank Maneesha Jain (Sun) and Stephan Hesmer (IBM) who led the TCK and RI efforts.~~

~~PLT.1~~PLT.2 Overview

~~PLT.1.1~~PLT.2.1 What is a Portal?

A portal is a web based application that –commonly- provides personalization, ~~single sign on~~authentication, content aggregation from different sources and hosts the presentation layer of ~~Information~~ information ~~Systems~~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.1.2~~PLT.2.2 What is a Portlet?

A portlet is ~~a Java technology based web component,~~ an application that provides a specific piece of content (information or service) to be included as part of a portal page. It is 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~~ information ~~Systems~~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.

This specification will deal with Portlets as Java technology based web components.

~~PLT.1.3~~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.1.4~~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 Compatibility

The Java Portlet Specification V 2.0 does not break binary compatibility with V 1.0. This means that all portlets written against the V 1.0 specification can run unchanged. Portlet V2.0 containers must support deploying JSR 168 portlets and the JSR 168 deployment descriptor. [i]

PLT.2.6 Major changes introduced with V 2.0

~~The Java Portlet Specification V 2.0 does not break binary compatibility with V 1.0. This means that all portlets written against the V 1.0 specification can run unchanged. Portlet V2.0 containers must support deploying JSR 168 portlets and the JSR 168 deployment descriptor.~~ [ii]

The major new features of version 2.0 include:

- Events – enabling portlet to send and receive events and perform state changes or send further events as result of processing an event.
- ~~s~~Shared session attributes – allowing portlets to share session attributes beyond the current web application.

- ~~s~~Shared render parameters – allowing portlets to share render parameters with other portlets.
- Resource serving – provides ability for a portlet to serve a resource.~~enabling resource serving in the context of the portal with access to the portlet API objects~~.
- Portlet filter – allowing on the fly transformations of information in both the request to and the response from a portlet

~~Leveraging JAXB for Event and Shared Session payloads~~

~~The Java Portlet Specification 2.0 leverages the Java Architecture for~~

~~XML Binding (JAXB) 2.0 for defining payload data that needs to be transferred across different types of portlet container via the Web Services for Remote Portlets (WSRP) 2.0 specification. These are the event payload and the shared session attribute value.~~

~~The event payload and the shared session attribute value must be defined by either the following alternatives[iii]:~~

- ~~using the JAXB annotations in the Java object and defining the Java object class name in the deployment descript via the `java-class` element. Defining the Java object class name in the deployment descriptor is optional for publishing events and mandatory for consuming events.~~[iv]
- ~~providing an XML schema in the deployment descriptor via the `xml-schema` element and optionally a JAXB mapping via the `jaxb-mapping` element~~

~~PLT.1.5~~PLT.2.7 Relationship with Java 2 Platform, Standard and Enterprise Edition

The Portlet API v2~~1~~.0 is based on the Java 2 Platform, Standard Edition 5.0 and Enterprise Edition~~,~~ v1.~~43~~. Portlet containers ~~and portlets~~should at least meet the requirements, described in v 1.4 of 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~~ 4* whenever applicable.

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~~can only be invoked through URLs constructed via the portlet API.
- Web clients interact with portlets through a portal system
- Portlets have a more refined request handling, action requests, event request, resource 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. They can in addition allow the portlet container to share application-wide scoped attributes beyond the current web application.
- Send and receive events from other portlets or container defined events.

Portlets do not have access to the following functionality provided by servlets ~~in their render method~~:

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

The portlet has full control over the response when rendering resources via the `serve~~render~~Resource` call.

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 *PLT.16 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, per default:

- Attributes set in the portlet request are available in the included servlet request (see *PLT.16 Dispatching Requests to Servlets and JSPs* Chapter),
- The portlet and the included servlet or JSP share the same output stream (see *PLT.16 Dispatching Requests to Servlets and JSPs* Chapter).
- Attributes set in the portlet session are accessible from the servlet session and vice versa (see *PLT.15 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 at least *Servlet Specification 2.~~3~~4*.

Portlet Concepts

PLT.4.1 Portlets

~~The concept of a p~~Portlets allow you to ~~is to~~ provide a componentized ~~U~~user ~~I~~interface (UI) for services. In a Service Oriented Architecture (SOA) ~~you don't~~one does not write monolithic applications anymore, but separate services that can be orchestrated together into applications~~,~~. This service orchestration requires componentized UIs for the services, monolithic web UIs based on servlets are no longer sufficient.

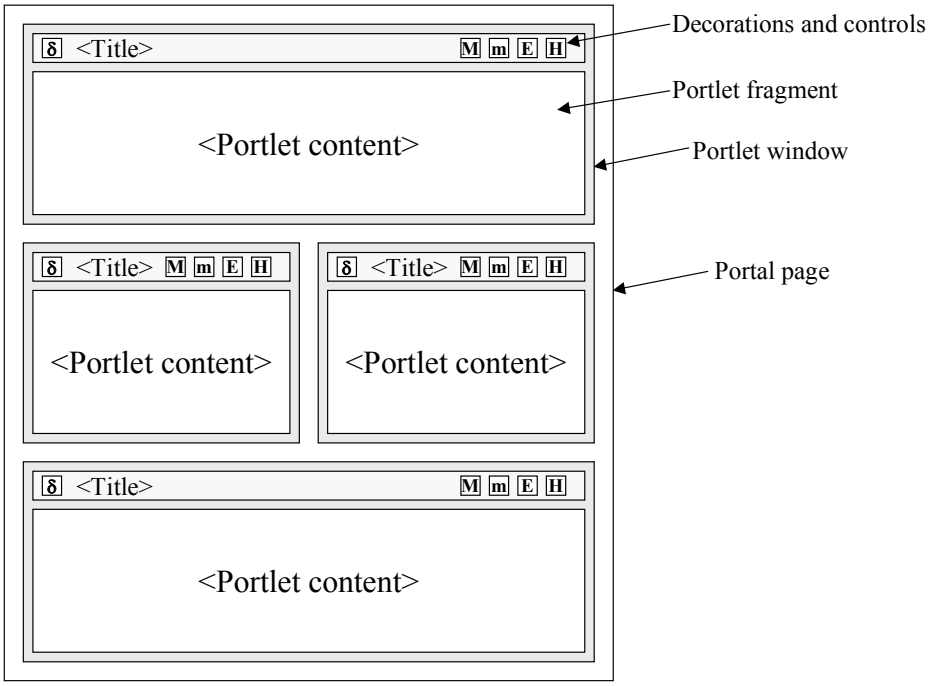
Portlets provide such a component UI model that is intended to aggregate the component UIs into a larger UI with consistent look and feel (see Appendix PLT.C Style Sheet Definitions). The Java Portlet Specification allows coordination on the UI layer with different means, such as events, shared sessions, and shared render parameters, in order to provide a deep and seamless integration between the different services.

The predominant applications using portlets today are portals aggregating the portlet markup into portal pages, but the Java Portlet Specification and portlets itself are not restricted to portals.

~~PLT.4.1~~PLT.4.2 Embedding Portlets as Elements of a Portal Page

A portlet generates markup fragments. A portal ~~normally~~ may add~~s~~ 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 may aggregate~~s~~ portlet windows into a complete document, the portal page.

Figure 4-1 Elements of a Portal Page

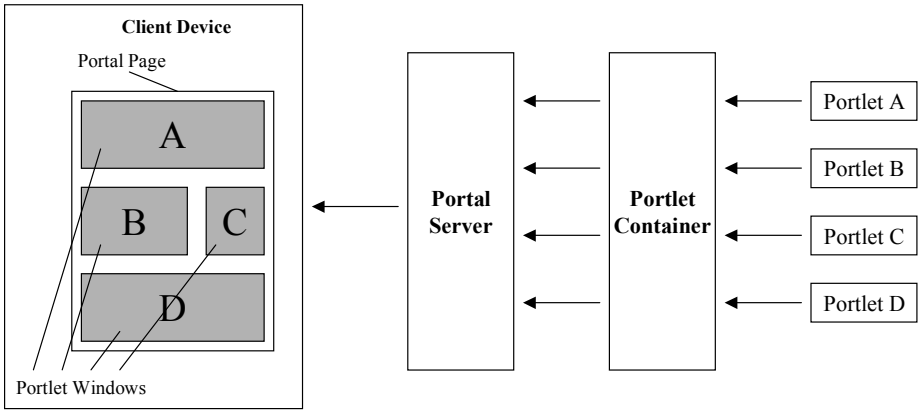


Note that this is only one example on how a portal could make use of the portlet markup fragment. There may exist other portlet implementations with a complete different rendering approach. The important part of the portal page concept in regards to this specification is that the markup fragment of the portlet may ~~not~~ be not the only markup returned in the document to the client. Thus the portlet markup needs to co-exist with whatever other markup the portal produces.

~~PLT.4.2~~PLT.4.2.1 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~~PLT.4.2.2 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.4.3 Portlets and Web Frameworks

The portlet model provides a clear separation of the state changing logic that is embedded in the `processAction` and `processEvent` methods and the rendering of the markup which is performed via the `render` and ~~`render`~~`serveResource` methods. The portlet model thus follows the popular Model-View-Controller pattern which separates the controller logic from the part that generates the view.

The default model that the Java Portlet Specification provides for rendering views is JSPs. However, once one starts creating advanced portlets, existing web frameworks, like Java Server Faces (JSF), Struts, WebWork~~s~~, Spring, or others may be used. When using such a web framework the portlet acts as a bridge between the portlet environment and the web framework.

Version 2.0 of this specification provides additional means making the implementation of such bridges simpler.

~~PLT.1~~PLT.5

The Portlet Interface and Additional Life Cycle Interfaces

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 can optionally implement the additional life cycle interfaces ~~`PortletEvents`~~`EventPortlet` and ~~`PortletResourceServing`~~`ResourceServingPortlet` in order to leverage additional functionality like receiving / sending event~~ing~~s or serving resource ~~serving~~s.

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.1.1~~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[v] instantiate and 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). [vi]

~~PLT.1.2~~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.

The Java Portlet Specification V2.0 provides the additional optional lifecycle interfaces ~~PortletEvents~~EventPortlet and ~~PortletResourceServing~~ResourceServingPortlet that the portlet can implement.

~~PLT.1.2.1~~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.[vii] After loading the portlet classes, the portlet container instantiates them for use.

~~PLT.1.2.2~~PLT.5.2.2 Initialization

After the portlet object is instantiated, the portlet container must initialize the portlet before invoking it to handle requests.[viii] 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. Refer to *PLT.6 Portlet Config* Chapter for information about the `PortletConfig` interface. The configuration object also gives the portlet access to a context object that describes the portlet's runtime environment. Refer to *PLT.10 Portlet Context* Chapter for information about the `PortletContext` interface.

~~PLT.1.2.2.1~~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.[ix] The `destroy` method must not be called because the initialization is considered unsuccessful.[x]

The portlet container may reattempt 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.[xi]

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

~~PLT.1.2.2.2~~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 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.[xiii] 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.[xiv]

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

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

PLT.5.3 Portlet Customization Levels

The portlet model leverages the flyweight pattern and provides the Java instance of the portlet class with all needed data in each request. This keeps the number of Java instances small and thus allows better scalability for large user numbers. In order to distinct between the different ~~customization~~ levels of customization the terms portlet definition, portlet entity and portlet window are introduced in this section.

PLT.5.3.1 Portlet Definition and Portlet Entity

The portlet definition may include a set of preference attributes with their default values. They are used to create preferences objects (see *PLT.14 Portlet Preferences* Chapter).

At runtime, when serving requests, a ~~portlet~~reference object is associated with a ~~preferences object~~portlet. The resulting association is called the portlet entity. This concept is abstract. There is not a concrete object that represents the portlet entity. The portal / portlet container merely associates the proper preference object with the context that is passed to the executing portlet.

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.

Administration, management and configuration of preferences objects are 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.5.3.2 Portlet Window

Consuming applications, like portals, typically have a more concrete concept of portlets th~~e~~an the model of this specification. In a consuming application portlets are customizable, visual components used within portal pages. Such a usage within a portal page is termed a portlet window. Because of the customizable aspects of portlets, each portlet window can have many preference objects associated with it; i.e. there is a 1:N relationship between a portlet window and portlet entities. For example some portal implementations may group the read-only preferences that are managed by the administrator to a portlet entity and the read-write preferences that are managed by the portlet user to a different portlet entity.

However, at runtime the portlet will not be able to distinguish these different preference objects as the portlet container will provide always one aggregated set of preferences to the portlet. Though typically portlet windows maintain distinct sets of portlet entities from other portlet windows (based on the same portlet), this need not be the case. Two (or more) portlet windows can share the same portlet entity set and thus provide distinct views onto the same thing. From a developer's perspective, portlet windows are important because they define distinct runtime views. Hence runtime state (transient state) such as render parameters, portlet mode, window state, and the portlet-scoped session state are maintained based on a portlet window. For example the user may want to reference the

same portlet entity from different pages, but does not want to have the runtime state shared between these two.

Each portlet window gets a unique ID assigned by the portal / portlet container that is valid for the lifetime of this portlet window. The portlet window ID can be accessed by the portlet via the `PortletRequest.getWindowID()` call and is used by the portlet container for keying the portlet-scoped session data. The portlet window ID returned by `PortletRequest.getWindowID()` must not contain a '?' character in order to comply with the requirement for the portlet scope session ID (*see PLT.17.3*)

~~Portlet Definition and Portlet Entity~~

~~The portlet definition may include a set of preference attributes with their default values. They are used to create preferences objects (see *PLT.14 Portlet Preferences* Chapter).~~

~~At runtime, when serving requests, a portlet object is associated with a preferences object. The resulting association is called the portlet entity. This concept is abstract. There is not a concrete object that represents the portlet entity. The portal / portlet container merely associates the proper preference object with the context that is passed to the executing portlet. This pattern is often referred to as flyweight pattern in the literature.~~

~~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.~~

~~Administration, management and configuration of preferences objects 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.5.2.3Portlet Window~~

~~Consuming applications, like portals, typically have a more concrete concept of portlets then the model of this specification. In a consuming application portlets are customizable, visual components used within portal pages. Such a usage within a portal page is termed a portlet window. Because of the customizable aspects of portlets, each portlet window can have many preferences associated with it; i.e. there is a 1:N relationship between a portlet window and portlet entities. Though typically portlet windows maintain distinct sets of portlet entities from other portlet windows (based on~~

the same portlet), this need not be the case. Two (or more) portlet windows can share the same portlet entity set and thus provide distinct views onto the same thing. From a developer's perspective, portlet windows are important because they define distinct runtime views. Hence runtime state (transient state) such as render parameters, portlet mode, window state, and the portlet-scoped session state are maintained based on a portlet window.

The portlet definition may include a set of preference attributes with their default values. They are used to create preferences objects (see *PLT.14 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.1.2.4~~PLT.5.4 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. In addition the portlet may implement one of the optional interfaces `EventPortlet` and `ResourceServingPortlet` that define the additional lifecycle methods `processEvent` and `serveResource`.

When a portal/portlet-container invokes the `processAction` method of a portlet, the portlet request is referred to as an action request. As a result of an action, ~~or triggered by the portal/portlet container,~~the portlet may publish one or more events ~~can be published~~, which result in one or more invocations of the `processEvent` method of a portlet with the portlet request referred to as an event requests. In addition to these portlet initiated events the portal/portlet container may issue portal/portlet container specific events. When a portal/portlet-container invokes the `render` method of a portlet, the portlet request is referred to as a render request. When a portal/portlet-container invokes the `serveResource` method of a portlet, the portlet request is referred to as a resource 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~~ three types, action URLs ~~or~~, render URLs, or resource URLs. Refer to *PLT.7 Portlet URLs* Chapter for details on portlet URLs.

Normally, a client request triggered by an action URL translates into one action request, zero or more event requests and many render requests, one per portlet in the portal page. These render requests may be followed by zero or more resource requests. A client request triggered by a render URL translates into many render requests, one per portlet in the portal page. ~~These render requests may be followed by zero or more resource requests.~~ A client request trigged by a resource URL translates into a render resource request.

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.[xvii] The portal/portlet-container must wait until the action request finishes. Then, the portal/portlet-container may call the `processEvent` methods of the event receiving portlets and after the event processing is finished 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.[xviii] The render requests may be executed sequentially or in parallel without any guaranteed order.

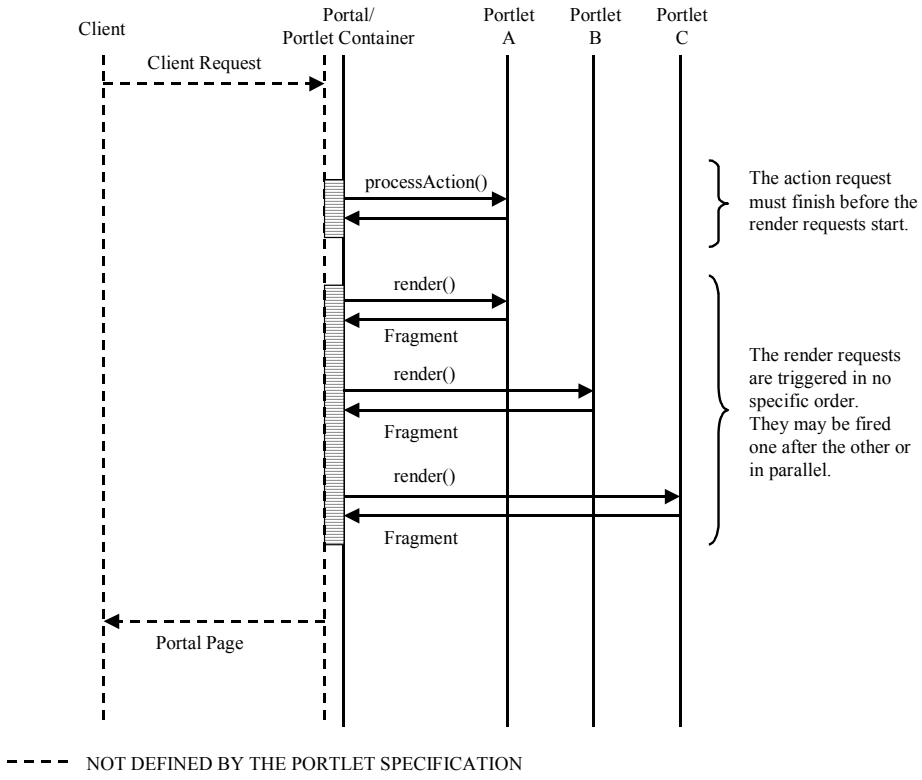
If the client request is triggered by a render URL, the portal/portlet-container must invoke the `render` method for all the portlets in the portal page with the possible exception of portlets for which their content is being cached.[xix] The portal/portlet-container must not invoke the `processAction`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

If the client request is triggered by a resource URL, the portal/portlet-container must invoke the `serveResource` method of the target portlet.[xx] The portal/portlet-container must not invoke the `processAction` of any of the portlets in the portal page for that client request.

5 If a portlet has caching enabled, the ~~portal/~~portlet-container may choose not to invoke the `render` or `serveResource` method. The portal/portlet-container may instead use the portlet's cached content. Refer to *PLT.~~18~~ 21 Caching* Chapter for details on caching.

A portlet object placed into service by a portlet container may end up handling no request during its lifetime.

Figure 5-1 Request Handling Sequence (needs to be updated)



~~PLT.1.2.4.1~~PLT.5.4.1 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[xxi] 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 must be effective for the following ~~render~~ request the portlet receives. There are some exceptional

circumstances, such as changes of access control privileges, that could prevent the portlet mode change from happening. The change of window state should be effective for the following ~~render~~ request the portlet receives. The portlet should not assume that the subsequent request will be in the window state set as the portal/portlet-container could override the window state because of implementation dependencies between portlet modes and window states.

The portlet may also set, in the `ActionResponse` object, render parameters during the processing of an action request. Refer to *PLT.11.1.1 Request Parameters* Section for details on render parameters.

The portlet may publish events via the `ActionResponse setEvent` or `setEvents` methods and thus publish state changes to other portlets. See *PLT.14* for more details on sending and receiving events.

PLT.5.4.2 Event Request

Events can be used to coordinate state between different portlets. The `processEvent` method of the `EventPortlet` interface receives two parameters, `EventRequest` and `EventResponse`.

The `EventRequest` object provides access to information such as the event payload, the window state, the portlet mode, the current render parameters, the portal context, the portlet session and the portlet preferences data.

A portlet may change its portlet mode and its window state during an event request. This is done using the `EventResponse` object. The change of portlet mode must be effective for the following request the portlet receives. There are some exceptional circumstances, such as changes of access control privileges, that could prevent the portlet mode change from happening. The change of window state should be effective for the following request the portlet receives. The portlet should not assume that the subsequent request will be in the window state set as the portal/portlet-container could override the window state because of implementation dependencies between portlet modes and window states.

The portlet may also set, in the `EventResponse` object, new render parameters during the processing of an event request. Refer to *PLT.11.1.1 Request Parameters* Section for details on render parameters.

The portlet may publish events via the `EventResponse setEvent` or `setEvents` methods and thus publish state changes to other portlets. See *PLT.14* for more details on sending and receiving events.

~~PLT.1.2.4.2~~PLT.5.4.3 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 *PLT.16 Dispatching Requests to Servlets and JSPs* Chapter for details on this.

PLT.5.4.4 Resource Request

In order to render resources via the portlet the portlet can implement the `ResourceServingPortlet` interface and create resource URLs that will trigger the `serveResource` method on this interface. The `serveResource` method of the `ResourceServingPortlet` interface receives two parameters, `ResourceRequest` and `RenderResponse`.

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

The portlet can produce content using the `Re~~nder~~sourceResponse` writer or output stream, or it may delegate the generation of content to a servlet or a JSP. Refer to *PLT.16 Dispatching Requests to Servlets and JSPs* Chapter for details on this.

PLT.5.4.5 GenericPortlet

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

The processEvent method in the GenericPortlet class tries to dispatch to methods annotated with the tag `@ProcessEvent(Retention=RUNTIME, name=<event name>)` and following signature:

```
void <methodname> (EventRequest, EventResponse) throws
PortletException, java.io.IOException;
```

Typically, portlets will extend the `GenericPortlet` class directly or indirectly and they will provide one method per consuming event that complies with the above definition in order to have the events dispatched to different methods.

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 *PLT.8 Portlet Modes* Chapter). These methods are:

- `doView` for handling VIEW requests[xxii]
- `doEdit` for handling EDIT requests[xxiii]
- `doHelp` for handling HELP requests[xxiv]

If the window state of the portlet (see *PLT.9 Window States* Chapter) is `MINIMIZED`, the `render` method of the `GenericPortlet` does not invoke any of the portlet mode rendering methods.[xxv]

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

~~PLT.1.2.4.3~~PLT.5.4.6 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.1.2.4.4~~PLT.5.4.7 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~~.[xxvi] The portal/portlet-container should continue processing the other portlets visible in the portal 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 appropriate manner.

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.[xxvii] 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 not to 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`.[xxviii]

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.1.2.4.5~~PLT.5.4.8 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.5.2.5End 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.[xxx] 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.[xxx]~~

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

~~After the `destroy` method completes, the portlet container must release the portlet object so that it is eligible for garbage collection.[xxxii] 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 may 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 and keywords (see the *PLT.21.10 Resource Bundles* Section).

These resource elements can be directly included in the portlet definition in the deployment descriptor, or they can be placed in a resource bundle.

An example of a deployment descriptor defining portlet information inline could be:

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

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

```
<portlet>
  ...
  <resource-bundle>com.foo.myApp.QuotePortlet</resource-bundle>
  ...
</portlet>
```

If the portlet definition defines a resource bundle, the portlet-container must look up these values in the ResourceBundle. If the root resource bundle does not contain the resources for these values and the values are defined inline, the portlet container must add the inline values as resources of the root resource bundle.[xxxiii]

If the portlet definition does not define a resource bundle and 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 *PLT.21.10 Resource Bundles* Section.[xxxiv]

The `render` method of the `GenericPortlet` uses the `ResourceBundle` object of the `PortletConfig` to retrieve the title of the portlet from the associated ResourceBundle or the inline information in the portlet definition.

~~PLT.1~~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.1.1~~PLT.7.1 Portlet URLs

The Portlet API defines the `PortletURL` and `ResourceURL` interface. Portlets must create portlet URLs either using `PortletURL` or the `ResourceURL` objects. A portlet creates `PortletURL` objects invoking the `createActionURL`, ~~and the~~ `createRenderURL`, and the `createResourceURL` methods of the ~~`RenderResponse`~~ `PortletResponse` interface. The `createActionURL` method creates action URLs. The `createRenderURL` method creates render URLs. The `createResourceURL` creates ~~render~~ resource URLs.

Because some portal/portlet-containers implementations may encode internal state as part of the URL query string, portlet developers should not code forms using the HTTP GET method.

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.[xxxv] The portal/portlet-container must ensure that all the parameters set when constructing the render URL become render parameters of the subsequent render requests for the portlet.[xxxvi]

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 be accessed via HTTP method GET as they should not change any state on the server. As a consequence, render URLs become bookmarkable.

Render URLs should not be used within forms as the portal/portlet-container may ignore form parameters.

A resource URL allows the portlet rendering resources with access to information of the portlet request. When rendering resources the portlet has full control over the outputstream and can render binary markup. Resource URLs should be accessed via HTTP method GET as they should not change any state on the server.

Note that portlet URLs are only valid within the current request and need to be either written to the outputstream or passed in Portlet API methods that are capable of re-writing the portlet URL token into a real URL, like as parameter on a redirect URL in the `sendRedirect` method of the `ActionResponse`.

PLT.7.1.1 BaseURL interface

The `BaseURL` interface provides the basic methods that are common for all URLs pointing back to the portlet, like `ResourceURLs`, `ActionURLs`, and `RenderURLs`. `BaseURLs` are always created ~~as instances of~~ either as a~~n~~ ~~R~~resource URL, ~~A~~action URL, or ~~R~~render URL.

Portlets can add application specific parameters to the ~~`Portlet`~~`BaseURL` objects using the `setParameter` and `setParameters` methods. A call to any of the `setParameter` methods must replace any parameter with the same name previously set.[xxxvii] All the parameters a portlet adds to a ~~`PortletURL`~~`BaseURL` object must be made available to the portlet as request parameters.[xxxviii] Portlet developers should note that the parameters of the current render request are not carried over when creating a ~~`Portlet`~~`BaseURL`, except when creating a `ResourceURL` that contains the current render parameters.

The portlet-container must `"x-www-form-urlencoded"` encode parameter names and values added to a ~~`PortletURL`~~`BaseURL` object.[xxxix]

If ~~Portlet~~ portlet developers ~~should not encode~~namespace parameter names or values before adding them to a ~~`PortletURL`~~`BaseURL` object they are also responsible for removing the namespace. The portlet container will not remove any namespacing the portlet has done on these parameters..

If a portal/portlet-container encodes additional information as parameters, it must ~~encode~~ namespace them properly to avoid collisions with the parameters set and used by the portlet.[xl]

~~If the portlet mode is not set for a URL, it must stay the same as the mode of the current request.~~[xli]

~~If the window state is not set for a URL, it should stay the same as the window state of the current request.~~

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.setParameter("customer","foo.com");
url.setParameter("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 a special token 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.1.1.1~~PLT.7.1.2 Including a Portlet Mode or a Window State

A portlet URL can include a specific portlet mode (see *PLT.8 Portlet Modes* Chapter) or window state (see *PLT.9 Window States* Chapter). 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.setParameter("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.[xlii]. The change of portlet mode must be effective for the request triggered by the portlet URL.[xliii] There are some exceptional circumstances, such as changes access control privileges, that could prevent the portlet mode change from happening. If the portlet mode is not set for a URL, it must have the portlet mode of the current request as default[xliv].

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.[xlv] The change of window state should be effective for the request triggered by the portlet URL. The portlet should not assume that the request triggered by the portlet URL will be in the window state set as the portal/portlet-container could override the window state because of implementation dependencies between portlet modes and window states. If the window state is not set for a URL, it must have the window state of the current request as default[xlvi].

~~PLT.1.1.2~~PLT.7.1.3 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.[xlvii]

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 *PLT.14 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.

Portlets are not required to support the EDIT portlet mode.

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.

Portlets are not required to support the HELP portlet mode.

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 *PLT.A 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.[xlviii] 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.[xlix] The portlet must not be invoked in a portlet mode that has not been declared as supported for a given markup type.[l]

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.[li]

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.[lii]

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.[liii] 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.[liv]

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 context-wide initialization parameters are the same as 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 the same that are accessible through the `ServletContext` of the portlet application.[lv]

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.[lvi]

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

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`.[lviii] 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)[lix]:

PLT.10.3.1 Correspondence between ServletContext and PortletContext methods

The following methods of the `PortletContext` should provide the same functionality as the methods of the `ServletContext` of similar name: `getAttribute`, `getAttributeNames`, `getInitParameter`, `getInitParameterNames`, `getMimeType`, `getRealPath`, `getResource`, `getResourcePaths`, `getResourceAsStream`, `log`, `removeAttribute` and `setAttribute`.

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 the `processAction`, `processEvent`, `serveResource` and `render` methods of the portlet.

~~PLT.1.1~~PLT.11.1 PortletRequest Interface

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

~~PLT.1.1.1~~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 encoded in the URL (added when creating the PortletURL) and the string parameters sent by the client to the portlet as part of the client request.[lx] The parameters the request object returns must be "`x-www-form-urlencoded`" decoded.[lxi]

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` [lxii]. 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.[lxiii]. The `getParameterMap` method must return an unmodifiable `Map` object[lxiv]. If the request does not have any parameter, the `getParameterMap` must return an empty `Map` object[lxv].The values in the returned `Map` object are from type `String` array.

PLT.11.1.1.1 Action, Resource and Event Request Parameters

The portlet-container must not propagate parameters received in an action, resource or event request to subsequent render requests of the portlet.[lxvi]

If a portlet wants to do that in either the `processAction` or `processEvent`, it can use render URLs or it must use the `setRenderParameter` or `setRenderParameters` methods of the ~~`ActionStateModifyingResponse`~~`StateAwareResponse` object within the `processAction` or `processEvent` call.

PLT.11.1.1.2 Render Request Parameters

If a portlet receives a render request that is the result of a client request targeted to another portlet in the portal page or an event, the parameters ~~must~~ should be the same parameters as of the previous render request from this client.[lxvii]

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

If a portlet receives a render request that is the result of clicking on a render URL targeting this portlet the render parameters received with the render request must be the parameters set on the render URL.[lxix]

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.[lxx]

A portlet must not see any non-shared parameter targeted to other portlets.[lxxi] If ~~P~~portlets ~~should not~~ namespace or encode URL parameters or form parameters they are also responsible for removing the namespace. The portlet container will not remove any namespacing the portlet has done on these parameters.

~~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`.~~[lxxii] ~~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.~~[lxxiii] ~~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.11.1.1.3 Shared Render Parameters

In order to allow co-ordination of render parameters with other portlets the portlet can declare shared render parameters in its deployment descriptor using the `shared-render-parameter` ~~tag~~element ~~o~~in the portlet application ~~level~~section. ~~On~~In the portlet ~~level~~section each portlet can specify the shared render parameters it would like to receive or set via the `supported-shared-render-parameter` ~~tag~~element. The `supported-shared-render-parameter` ~~tag~~element must reference a shared render parameter defined ~~at~~in portlet application ~~level~~section with the first name entry ~~of the~~in that `shared-render-parameter` ~~tag~~element[lxxiv].

Example:

```
<shared-render-parameter>
        <name>foo</name>
        <name>foo2</name>
</shared-render-parameter>
<shared-render-parameter>
        <name>bar</name>
</shared-render-parameter>
<portlet>
        <portlet-name>portletA</portlet-name>
        …
        <supported-shared-render-parameter>foo</supported-shared-render-parameter>
</portlet>
<portlet>
        <portlet-name>portletB</portlet-name>
        …
        <supported-shared-render-parameter>bar</supported-shared-render-parameter>
</portlet>
```

The portlet container must only send those shared render parameter to a portlet which the portlet has defined support for using ~~the first name entry in the~~ `supported-shared-render-parameter` ~~tag~~element ~~as parameter name~~ in the portlet.xml[lxxv]. The portlet

container must only share those render parameters of a portlet which the portlet has declared as supported shared render parameters using `supported-shared-render-parameter` element in the portlet.xml [lxxvi]. The portlet container is free to only provide a subset of the defined shared render parameters to portlets that are not target of a render URL. A shared render parameter that is not supplied for this request should be viewed by the portlet as having the value `null`.

If the portlet was the target of a render URL and this render URL has set a specific shared render parameter the portlet must receive at least this render parameter~~, however, the value may have been changed in the meantime and the portlet thus may get a different value than the one specified in the URL.~~ [lxxvii]

If a portlet sets a render parameter to null it must be treated by the portlet container / portal as deleted. [lxxviii]

Portlet should only set shared render parameters on a URL that it wants to change, as the non-changed shared render parameters are provided by the portlet container to the portlet per default.

~~All statements previously made about render parameters also apply to shared render parameters, as they are render parameters~~[lxxix]~~.~~ The parameter name should uniquely identify the shared render parameter and use the Java package naming standard (INSERT REF HERE) and character restrictions.

It is up to the portal implementation to decide which portlets may share the same shared render parameters. The portal should use ~~additional~~the information provided in the deployment descriptor, like the ~~alias~~names and description , in order to perform such a mapping between shared render parameters of different portlets. It is also an implementation choice of the portal whether different portlet ~~instance~~entities of the same portlet will receive the same shared render parameters. An example where different portlet entities may not want to share the same render parameters is a generic viewer portlet that takes as shared render parameter the news article ID to display. The user may have several of this viewer portlets on her pages that may be connected to different content systems.

To enable localization support of ~~public~~shared parameters for administration and configuration tools, developers should provide a display name in the portlet application ResourceBundle (see the *PLT.~~XXXXX~~24.10 Resource Bundles* Section). The entry for the display name should be constructed as `'javax.portlet.app.shared-render-parameter.<~~param~~name>.display-name'`.

~~PLT.1.1.2~~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. [lxxx] It is the responsibility of the portal/portlet-container to properly

~~encode~~ namespace these extra parameters to avoid name collisions with parameters the portlets define.

Parameter names beginning with the `"javax.portlet."` prefix are reserved for definition by this specification for use by portal/portlet-container implementations.

~~PLT.1.1.3~~PLT.11.1.3 Request Attributes

Request attributes are objects associated with a portlet during a single portlet request. Portlets can not assume that attributes are shared between action, resource, event and render requests. 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.1.1.4~~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: content-length, content-type, accept-language. 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.1.1.5~~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 deployed portlet application. If the portlet application is rooted at the base of the web server URL namespace (also known as "default" context), this path must be an empty string.[lxxxi] Otherwise, it must be the path the portlet application is rooted to, the path must start with a '/' and it must not end with a '/' character.[lxxxii]

~~PLT.1.1.6~~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`.[lxxxiii]

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.1.1.7~~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.[lxxxiv]

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

The `getResponseContentTypes` method must return only the content types supported by the current portlet mode of the portlet.[lxxxv]

~~PLT.1.1.8~~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.1.1.9~~PLT.11.1.9 Portlet Mode

The `getPortletMode` method of the `PortletRequest` interface allows a portlet to find out its current portlet mode. 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, the portlet or the user has been constrained further by the portal.

~~PLT.1.1.10~~PLT.11.1.10 Window State

The `getWindowState` method of the `PortletRequest` interface allows a portlet to find out its current window state.

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.1.2~~PLT.11.2 ~~ActionRequest~~ ClientHttpRequest Interface

The ~~`ActionRequest`~~ `ClientHttpRequest` interface extends the `PortletRequest` interface and it is used as base class for the `ActionRequest` and `ResourceRequest`~~in the `processAction` method of the `Portlet` interface~~. In addition to the functionality provided by the `PortletRequest` interface, the ~~`ActionRequest`~~ `ClientHttpRequest` interface represents the request information ~~-~~of the HTTP request issued from the client to the consuming application / portal, such as ~~gives access to~~ the input stream~~ of the request~~.

~~PLT.1.2.1~~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~~`ClientHttpRequest` 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 `IllegalStateException`. Similarly, if the reader is obtained, a call to the `getPortletInputStream` must throw an `IllegalStateException`.[lxxxvi]

To help manage the input stream, the ~~Action~~ClientHttpRequest interface also provides the following methods:

- getContentType
- getCharacterEncoding
- setCharacterEncoding
- getContentLength

The setCharacterEncoding method only sets the character set for the Reader that the getReader method returns.

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.[lxxxvii]

PLT.11.3 ActionRequest Interface

The ActionRequest interface extends the ClientHttpRequest interface and is used in the processAction method of the Portlet interface. Currently, the ActionRequest interface does not define any additional method.

PLT.11.4 ResourceRequest Interface

The ResourceRequest interface extends the ClientHttpRequest interface and is used in the serveResource method of the ~~PortletResourceServing~~ResourceServingPortlet interface. Currently, the ResourceRequest interface does not define any additional method.

PLT.11.5 EventRequest Interface

The EventRequest interface extends the PortletRequest interface and provides current render parameters via one of the getParameter methods. If the portlet wants to maintain render parameters it needs to set them again in the EventResponse interface with the of the setRenderParameter~~s~~ methods. If no render parameters are set during the processEvent invocation, the following processEvent or render requests must not contain any non-shared ~~request~~render parameters.[lxxxviii] Note that the container may provide shared render parameters for following processEvent or render requests even if the portlet did not set any new render parameters, as other portlets may have set new values to the shared render parameters.

The EventRequest interface provides the event that triggered the processEvent call via the getEvent method which returns an Event object. The Event object provides the event name via getName and the name should match one of the receiving events defined by the portlet with <supported-processing-event> element. If the event is declared with a type in the deployment descriptor Event.getValue must return a serializable Java object that implements the specified Java type, or if the type is defined as JAXB XML type or schema, the corresponding JAXB mapping.

If the portlet does not set a new portlet or window state at the `EventResponse` interface the current portlet mode and window state are preserved.

~~PLT.1.3~~PLT.11.6 RenderRequest Interface

The `RenderRequest` interface extends the `PortletRequest` interface and is used in the `render` method of the `Portlet` interface. Currently, the `RenderRequest` interface does not define any additional method.

~~PLT.1.4~~PLT.11.7 Lifetime of the Request Objects

Each request object is valid only within the scope of a particular `processAction`, `processEvent`, `serveResource` 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.

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 the `processAction` and the `render` methods of the portlet.

~~PLT.1.1~~PLT.12.1 PortletResponse Interface

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

~~PLT.1.1.1~~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.1.1.2~~PLT.12.1.2 Encoding of URLs

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 implementations 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.

Portlet developer should be aware that the returned URL may not be a well formed URL but a special token at the time the portlet is generating its content. Thus portlets should

not add additional parameters on the resulting URL or expect to be able to parse the URL.

PLT.12.1.3 Namespacing

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 `getNamespace` method must provide the portlet with a mechanism that ensures the uniqueness of the returned string in the whole portal page.[lxxxix] For example, the `getNamespace` method would return a unique string that could be prefixed to a JavaScript variable name within the content generated by the portlet, ensuring its uniqueness in the whole page. The `getNamespace` method must return the same value for the lifetime of the portlet window.[xc]

The `getNamespace` method must return a valid identifier as defined in the *3.8 Identifier* Section of the *Java Language Specification Second Edition*.[xci]

PLT.12.2 ~~StateModifyingResponse~~StateAwareResponse Interface

The ~~`StateModifyingResponse`~~`StateAwareResponse` interface extends the `PortletResponse` interface and in addition provides methods to set new render parameters, a new portlet mode, or window state. `ActionResponse` and `EventResponse` both extend this interface.

PLT.12.2.1 Render Parameters

Using the `setRenderParameter` and `setRenderParameters` methods portlets may set render parameters. A call to any of the `setRenderParameter` methods must replace any parameter with the same name previously set. [xcii] Subsequent lifecycle calls, like `processEvent` or `render` that are part of the current client request should contain the newly set render parameters. If no other requests that influence render parameters, like subsequent `processEvent` calls of this client request, occur these parameters will be used in all subsequent render requests until a new client request or event targets the portlet.

Portlet developers do not need to `"x-www-form-urlencoded"` encode render parameters names and values set in the ~~StateModifyingResponse~~StateAwareResponse.

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 would be effective in the following `processEvent` and `render` requests. If a portlet attempts to set a portlet mode that it is not allowed to switch to, a `PortletModeException` must be thrown.[xciii]

The `setWindowState` method allows a portlet to change its current window state. The new window state would be effective in the following `processEvent` and `render`

requests. If a portlet attempts to set a window state that it is not allowed to switch to, a `WindowStateException` must be thrown.[xciv]

Portlets cannot assume that subsequent `processEvent` or `render` calls will be called with the set portlet mode or window state as the portal/portlet-container could override these changes.

PLT.12.2.3 Publishing Events

The portlet can publish events via the `setEvent` method for a single event, or with `setEvents` for multiple events. It is also valid to call `setEvent` multiple times in the current `processAction` or `processEvent` method and thus publish multiple events. Note that neither the order of the event in the events Map of the `setEvents` method nor the order of calling `setEvent` multiple times implies any order on how these events may be delivered to target portlets. The event payload must have a valid JAXB binding and implement `java.io.Serializable`.[xcv]

~~PLT.1.2~~PLT.12.3 ActionResponse Interface

The `ActionResponse` interface extends the ~~`PortletStateModifyingResponse`~~`StateAwareResponse` 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.1.2.1~~PLT.12.3.1 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.[xcvi]

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.[xcvii]

~~PLT.12.2.2Portlet Modes and Window State Changes~~

~~The `setPortletMode` method allows a portlet to change its current portlet mode. The new portlet mode would 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.~~[xcviii]

~~The `setWindowState` method allows a portlet to change its current window state. The new window state would 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.[xciv]~~

~~Portlets cannot assume that subsequent renders will be called in the set portlet mode or window state as the portal/portlet-container could override these changes.~~

~~If the `setPortletMode` or `setWindowState` methods are called after the `sendRedirect` method has been called an `IllegalStateException` must be thrown.[e] If the exception is caught by the portlet, the redirection must be executed.[ci] If the exception is propagated back to the portlet-container, the redirection must not be executed.[cii]~~

~~PLT.12.2.3Render Parameters~~

~~Using the `setRenderParameter` and `setRenderParameters` methods of the `ActionResponse` interface portlets may set render parameters during an action request. A call to any of the `setRenderParameter` methods must replace any parameter with the same name previously set.[ciii] These parameters will be used in all subsequent render requests until 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.[civ]~~

~~Portlet developers do not need to `"x-www-form-urlencoded"` encode render parameters names and values set in the ActionResponse.~~

~~If the `setRenderParameter` or `setRenderParameters` methods are called after the `sendRedirect` method has been called an `IllegalStateException` must be thrown.[cv] If the exception is caught by the portlet, the redirection must be executed. If the exception is propagated back to the portlet-container, the redirection must not be executed.[cvi]~~

PLT.12.4 EventResponse Interface

The `EventResponse` interface extends the ~~`StateModifyingResponse`~~`StateAwareResponse` interface ~~and~~ and does not add any additional methods. One thing to note is that if a portlet receives multiple `processEvent` calls~~events~~ while processing one client request the new portlet mode or window state that the portlet may have set, may ~~not~~ be not validated ~~if they are valid~~ by the portal between these ~~different~~multiple `processEvent` calls. This means that even if the portlet container may not throw an exception when the portlet sets a new portlet mode or window state that the portal may still not approve this portlet mode or window state change and call the portlet render method with a different portlet mode or window state.

~~PLT.1.3~~PLT.12.5 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.12.6 ResourceResponse Interface

The `ResourceResponse` interface extends the `RenderResponse` interface and it is used in the `serveResource` method of the `ResourceServingPortlet` interface. This interface allows a portlet to generate content that is directly served to the client, including binary content.

~~PLT.1.3.1~~PLT.12.6.1 Content Type

A portlet must set the content type of the response using the `setContentType` method of the `RenderResponse` interface. The `setContentType` method must throw an `IllegalArgumentException` if the content type set does not match (including wildcard matching) any of the content types returned by the `getResponseContentType` method of the `PortleRequest` object[cvii]. The portlet container should ignore any character encoding specified as part of the content type.

If the `getWriter` or `getPortletOutputStream` methods are called before the `setContentType` method, they must throw an `IllegalStateException`.[cviii]

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

If the portlet has set a content type, the `getContentType` method must return it. Otherwise, the `getContentType` method must return `null`.[cix]

~~PLT.1.3.2~~PLT.12.6.2 Output Stream and Writer Objects

A portlet may generate its content by writing to the `OutputStream` or to the `Writer` of the `RenderResponse` object. A portlet must use only one of these objects. The portlet container must throw an `IllegalStateException` if a portlet attempts to use both.[cx]

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.

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.

~~PLT.1.3.3~~PLT.12.6.3 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).[cxi]

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.[cxii] This allows the container to reuse a set of fixed size buffers, providing a larger buffer than requested if appropriate. The method should be called before any content is written using a `OutputStream` or `Writer`. If any content has been written, this method may throw an `IllegalStateException`.

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.[cxiii] 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.[cxiv] The response and its associated buffer must be unchanged.[cxv]

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

~~PLT.12.3.4Namespace 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 `getNamespace` method must provide the portlet with a mechanism that ensures the uniqueness of the returned string in the whole portal page.[cxvii] For example, the `getNamespace` method would return a unique string that could be prefixed to a JavaScript variable name within the content generated by the portlet, ensuring its uniqueness in the whole page. The `getNamespace` method must return the same value if invoked multiple times within a render request.[cxviii]~~

~~The `getNamespace` method must return a valid identifier as defined in the *3.8 Identifier* Section of the *Java Language Specification Second Edition*.[cxix]~~

~~PLT.1.3.5~~PLT.12.6.4 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 committed, if called after it should be ignored.[cxx]

~~PLT.1.4~~PLT.12.7 Lifetime of Response Objects

Each response object is valid only within the scope of a particular `processAction`, `processEvent` or `render` method call. Containers commonly recycle response 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.

Resource ~~Rendering~~Serving

Portlets can create two different kinds of resource links in order to serve resources:

1. Direct links to the resource in the same portlet ~~WAR file~~web application. These links are constructed by the portlet and encoded with the PortletResponse.encodeURL() method.
   Note that this method may not return a valid URL.
   Direct links ~~will not~~are not guaranteed to pass through the portal server ~~and thus cannot assumed will notto be protected by the portal security and~~ . Direct links will not have the portlet context available.
   Direct links should be used for use cases where the access to the portlet context and access through the portal is not needed, as they are more efficient than resource serving requests through the portal.
2. Resource URL links pointing back to the portlet. Via ~~this~~these links the ~~render~~Re`serveResource` method of the ~~portlet~~`ResourceServingPortlet` interface is called and the portlet can serve the resource. Thus resources served via resource URLs ~~are~~may be protected by the portal security and can leverage the portlet context. Static resources should still be served with direct links in order to allow portals to configure and optimize static resource serving in a consistent manner.

The remainder of this chapter defines how resource URL links can be created and how the portlet is called to ~~render~~serve the resource.

PLT.13.1 ~~PortletResourceServing~~ResourceServingPortlet Interface

A portlet that wants to serve~~r~~ resources addressed via a resource URL must implement the ~~PortletResourceServing~~`ResourceServingPortlet` interface with the method ~~render~~`serveResource`. The portal / portlet container must not render any output in addition to the content returned by the ~~portlet~~`serveResource` call. The portal / portlet container should expect that the portlet may return binary content for a ~~render~~Re`serveResource` call.

The ~~renderResource~~`serveResource` call is ~~outside of the basic action processing / rendering part of the render phases~~normally following a `render` call and can be viewed as a logical extension ~~therefore part of~~ the render phase. ~~and t~~Thus ~~the same restrictions as for render calls apply: t~~he portlet ~~shouldmust~~should not change any state in the ~~renderResource~~`serveResource` call via the Portlet API. The ~~renderResource~~`serveResource` call should be provided with the current portlet mode

and window state. The ~~renderResource~~serveResource call should also be provided with the current render parameters.

The serveResource call can also be used to implement Asynchronous Javascript and XML (AJAX) use cases that want to fetch markup from the portlet without a complete page refresh. The supported use case types for serveResource includes retrieving new markup fragments based on the current portlet state and allows the portlet to include portlet URLs in the returned markup fragment. Use cases that modify any state via the Portlet API are not supported with serveResouce method.

NOTE: For portlet state changing AJAX use cases a different mechanism will be introduced in a future draft of V 2.0.

PLT.13.2 Access to Request and Response Headers

Given that the portal / portlet container do not render any additional markup for a render resource response it is important for the portlet to be able to access the incoming request headers and to be able to set new headers for the response.

A portlet can access the headers of the HTTP client request through the getProperty or getProperties call, like all portlet requests (see ~~*Chapter*~~*PLT ~~XXXX~~11.1.4*).

A portlet can set HTTP headers for the response via the setProperty or addProperty call in the PortletResponse. To be successfully transmitted back to the client, headers must be set before the response is committed. Headers set after the response is committed will be ignored by the portlet container.

PLT.13.3 Resource URLs

The portlet can create resource URLs pointing back to itself via the createResourceURL method on the ~~Render~~PortletResponse. When an end user invokes such a resource URL the portlet container must call the ~~renderResource~~serveResource method of the portlet or return a valid cached result for this resource URL[cxxi] If the portlet does not implement the PortletResourceServing interface it is left to the portal / portlet container to either provide some meaningful error handling or ignore the URL.

The portlet container must not call the processAction ~~or handleEvent~~ method[cxxii]. ~~Besides this the resource URL should be seen as a specific render URL and all statements made in the Chapter XXXX Portlet URLs section BaseURL apply[cxxiii].~~

Resource URLs are provided with the current portlet mode ~~and~~, window state, and render parameters that the portlet can access via the ~~Portlet~~ResourceRequest with getPortletMode ~~and~~, getWindowState, or one of the getParameter methods. [cxxiv] ResourceURLs cannot change the current portlet mode, window state or render parameters[cxxv]. Parameters set on a resource URL are not render parameters but parameters for rendering this resource and will last olny for ~~only this~~the current serveResource request.

If a parameter is set that has the same name as a render parameter that this resource URL contains, the render parameter must be the last entry in the parameter value array. [cxxvi]

~~Resource URLs should be accessed via HTTP method GET as they should not change any state on the server.~~

PLT.14

Coordination between portlets

In order to provide coordination between portlets the Java Portlet Specification introduces ~~different~~the following mechanisms:

- ~~•~~ sharing data between artifacts in the same web application via the session in the application scope (see ~~*Chapter XXXX*~~*PLT.17.2*)
- ~~•~~ sharing data across ~~web~~portlet applications in the session scope via the shared session attributes (see ~~Chapter *XXXX*~~*PLT.17.5*)
- ~~• publies~~shared render parameters in order to share render state between portlets (see ~~*Chapter XXXX*~~*PLT.11.1.1.1.3*)
- portlet events that a portlet can receive and send

In this chapter we'll cover the portlet events in more detail.

PLT.14.1 Shared Session State

Shared session state is intended to allow~~ing~~ portlets to share state that is related to the current user session and independent of the current navigation. An example for this would be a shopping cart that stores items the user would like to purchase. Items should not be removed if the user navigates back to previous views and thus should not be stored using shared render parameters. For more details on shared session attributes see ~~*the Session Chapter XXXX*~~*PLT.17.5*.

PLT.14.2 Shared Render Parameters

Shared render parameters are intended for sharing view state across portlets. Using shared render parameters instead of events avoids the additional process ~~action~~event call

and enables the end-user using the browser navigation and bookmarking if the portal stores the render parameters in the URL.

An example where shared render parameters are useful is the following: a weather portlet wants to display the weather of a selected city. It therefore uses the shared render parameters for encoding the zip code. The user now adds additional portlets on the page that also have zip code as one of their shared render parameters, like a map portlet displaying the location of the city ~~or selecting a city~~ and a tourist information portlet displaying tourist information for the selected city. If the portal encodes the zip code into the URL the user ~~can now~~even bookmark these information for specific cities.

For more details on shared render parameters see ~~*the Chapter XXXX*~~*PLT.11.1.1.1.3*.

PLT.14.3 Portlet Events

Portlet events are intended to allow~~ing~~ portlets to react on actions or state changes not directly related to an interaction of the user with the portlet. Events could be either portal or portlet container generated or the result of a user interaction with other portlets. The portlet event model is a loosely coupled, brokered, model that allows creating portlets as stand-alone portlets that can be wired together with other portlets at runtime. Portlet programmers should therefore not make any specific assumptions about the environment of portlets they are running together with. The means of wiring different portlets together is portal implementation specific.

Portlet events are not a replacement for reliable messaging (see other JEE APIs, like Java Message Service, JMS, for providing reliable messaging). Portlet events are not guaranteed to be delivered and thus the portlet should always work in a meaningful manner even if some or all events are not being delivered.

In response to an event a portlet may publish new events that should be delivered to other portlets and thus may trigger state changes on these other portlets.

PLT.14.3.1 ~~Portlet~~EventsPortlet Interface

In order to receive events the portlet must implement the ~~`PortletEvents`~~EventPortlet interface in the `javax.portlet` package. The portlet container will call the `processEvent` method for each event targeted to the portlet with an `EventRequest` and `EventResponse` object. Events are targeted by the portal / portlet container to a specific portlet window in the current user request.

Events are a new lifecycle operation that occurs before the rendering phase. ~~A portlet that is target of a user action can optionally receive container specific events before the action processing~~. The portlet may issue events via the `setEvent` or `setEvents` method during the action processing which will be processed by the portlet container after the action processing has finished.~~After the action processing is finished the portlet may issue events via the `sendEvent` method.~~ As a result of issuing an event the portlet may optionally receive events from other portlets or container events. A portlet that is not

target of a user action may optionally receive container events or events from other portlets.

PLT.14.3.2 Receiving and sending events

The portlet can access the event that triggered the current process event call by using the `EventRequest.getEvent` method. This method returns an object of type Event representing the current event. The event must always have a name and may optionally have a value.[cxxvii]

If the event has a value it is based on the type defined in the deployment descriptor. The type in the deployment descriptor can either be a Java type or a XML type defined in the JAXB specification or a schema. If the type is XML-based the portlet container is responsible for instantiating the correct Java object based on the XML type defined. The de-serialized Java object must implement the `java.io.Serializable` interface. [cxxviii] The default XML to Java mapping that every container must support is the JAXB mapping (see ~~*XXXXX*~~*PLT.26*).[cxxix] Portlet containers are free to support additional mapping mechanisms beyond the JAXB mapping. If the value of the event is not null it must reflect the type defined for this event in the deployment descriptor.[cxxx] If a Java class is specified as type in the deployment descriptor the event payload object must be of this declared type.[cxxxi]

For optimization purposes in local Java runtime environments the portlet container can use Java Serialization or direct Java object passing for the event payload. The portlet must not make any assumptions on the mechanism the portlet container chooses to pass the event payload.

Example for receiving an event:

event defined in the DD:

```
<event-definition>
  <name>com.acme.foo</name>
  <java-type>java.lang.String</java-type>
</event-definition>
....
<portlet>
...
<supported-processing-event>
  <name>com.acme.foo</name>
</supported-processing-event>
...
</portlet>
```

event processing in the portlet:

```
void processEvent(EventRequest req, EventResponse resp)
{
...
Event event = req.getEvent();
if ( event.getName().equals("com.acme.foo") )
  {
    String payload = (String) event.getValue();
    ...
```

```
    }
```

The portlet can publish events via the `EventResponse.setEvent` for a single event, or with `EventResponse.setEvents` for multiple events. It is also valid to call `EventResponse.setEvent` multiple times in the current `processEvent` method. Note that neither the order of the event in the events Map of the `EventResponse.setEvents` method nor the order of calling `EventResponse.setEvent` multiple times implies any order on how these events may be delivered to target portlets. The event payload must have a valid JAXB binding and implement `java.io.Serializable`, otherwise a `java.lang.IllegalArgumentException` must be thrown. [cxxxii]

Example for sending an event:

event defined in the DD:

```
<event-definition>
  <name>com.acme.bar</name>
  <java-type>com.acme.Address</java-type>
</event-definition>
....
<portlet>
...
<supported-publishing-event>
  <name>com.acme.bar</name>
</supported-publishing-event>
...
</portlet>
```

event processing in the portlet:

```
@XmlRootElement
    public class Address implements Serializable
    {
       private String street;
       private String city;
       public void setStreet(String s) {street = s;}
       public String getStreet() { return street;}
       public void setCity(String c) { city = c;}
       public String getCity() { return city;}
    }


void processEvent(EventRequest req, EventResponse resp)
{
...
Address sampleAddress = new Address();
sampleAddress.setStreet("myStreet");
sampleAddress.setCity("myCity");
resp.setEvent("com.acme,bar", sampleAddress);
```

PLT.14.3.3 Event declaration

The portlet should declare all events that it would like to receive and the ones it would like to initiate. The portlet container should only distribute events that the portlet has declared as processing events, but the portlet implementation should however be robust enough to deal with receiving events that it did not declare.

~~The portlet may declare events either statically in the deployment descriptor or dynamically via sending new events not previously defined.~~

PLT.14.3.3.1 Declaration in the deployment descriptor

The portlet can declare static events in the `portlet.xml` deployment descriptor (see ~~*Chapter XXXX*~~*PLT.24 Deployment Descriptor*). On the application level the portlet should define the basic event definition with the ~~`portlet:`~~`event-definition` ~~tag~~element. The event definition must contain at least one event name. [cxxxiii] The portlet container must use the first event name entry in the portlet deployment descriptor as event name when submitting an event to the portlet. The portlet can specify additional names beyond the first preferred name in order to enable portals performing an automatic wiring between events.

The event definition ~~must~~should be referenced on the portlet level where the portlet can define the processing events with the `supported-processing-event` ~~tag~~element and the events being published with the `supported-publishing-event` ~~tag~~element. ~~–~~Event definitions are valid for all ~~instance~~entities created based on the portlet definition.

Portlet container or portal defined events do not need to be declared on the application level with the `event-definition` element, but can be directly referenced on the portlet level with the `supported-processing-event` element.

The event name should uniquely identify the event and use the Java package naming standard (INSERT REF HERE) and character restrictions. The portlet is encouraged to organize the event names in a hierarchical manner using the dot '.' as separator. The portlet must not specify events with the same name but different types. Receiving event parameter names are allowed to end with a "*" character to indicate the portlet is willing to process any event whose name starts with the characters before the "*" character.

A localized display name for the portlet event definition should be provided in the application level resource bundle (see ~~*Chapter XXXX*~~*PLT.24.10*) with an entry of the name `javax.portlet.app.event-definition.<~~`~~`event-`~~`name>.display-name`.

PLT.14.3.3.2 Dynamic, non-declared events ~~declaration~~

The portlet can send~~declare~~ dynamic events, which are not declared in the portlet deployment descriptor, at runtime using the ~~`StateModifyingResponse.`~~`setEvent` or ~~`StateModifyingResponse.`~~`setEvents` methods on either the `ActionResponse` or `EventResponse`. ~~An event is called a dynamic event if the event name is not defined in the portlet deployment descriptor.~~ The portlet should note that by using dynamic events the abilities of the portal for distributing the event to other portlets may be limited or even non-existent.

PLT.14.3.4 Event processing

Events are valid only in the current user request and the portlet container must therefore deliver all events within the current request. Event delivery is not guaranteed and the container may restrict event delivery in a meaningful manner, e.g. in order to prevent

endless loops. Events are not ordered and the container may re-order the received events before distributing them. Event distribution is non-blocking and can happen in parallel for different portlet ~~instanceentitie~~windows.

Event distribution must be serialized for a specific portlet ~~instanceentity~~window per client request so that at any given time a portlet ~~instanceentity~~window is only processing one event in the `processEvent` method for the current client request~~ scope~~. The portlet container should therefore queue the events for one portlet ~~instanceentity~~window for one user. When processing the queue the container should take any previously returned event response data, like render parameters, portlet mode, window state, into account and supply these updated values with the event request.

Portlet ~~E~~event processing may appear after the processing of the action and must be finished before the render phase.~~at the following phases of the overall request processing:~~

- ~~`processEvent`, for container raised events~~
- ~~`processAction`, for the portlet that is target of the current user action, after container raised event processing is finished. Action processing must be finished before `processEvent` is called for this portlet entity in the current request scope.~~
- ~~`processEvent`, for events raised by portlets after `processAction` has finished or container raised events.~~
- ~~`render`, `serveResource` after all `processAction` and `processEvent` calls are finished or terminated~~

Container raised events are issued by the portlet container and not a portlet. The portlet should not publish container events, only process them. If a portlet would like to receive a container raised event it ~~must~~should declare the event in the portlet deployment descriptor with the `<supported-processing-event>` ~~tag~~element.

PLT.14.3.5 Exceptions during event processing

A portlet may throw ~~either~~ a `PortletException`, a `PortletSecurityException` or a~~n~~ `UnavailableException` during the `processEvent`.

A `PortletException` signals that an error has occurred during the processing of the event and that the portlet container should take appropriate measures to clean up the event processing. If a portlet throws an exception in the `processEvent` method, all operations on the `EventResponse` must be ignored. The portal/portlet-container should continue processing the other portlets participating in the current client request. Otherwise it is up to the portlet container implementation if the error is faced to the end user, the portlet is removed from the current request cycle or if the render method of the portlet is called.

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. 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 not to 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 event handling must be handled as a `PortletException`.

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.14.3.6 GenericPortlet support

The `GenericPortlet` implements the ~~`PortletEvents`~~EventPortlet interface and provides a default event handling. For a given event the `GenericPortlet` tries to dispatch to methods annotated with the tag `@ProcessEvent(Retention=RUNTIME, name=<event name>)` and following signature:

```
void <methodname> (EventRequest, EventResponse) throws
PortletException, java.io.IOException;
```

Note that the annotation must contain the `Retention=RUNTIME` metadata in order to allow `GenericPortlet` accessing the information at runtime. If no such method can be found the `GenericPortlet` just sets the received render parameters as new render parameters.

Example:

```
@ProcessEvent(Retention=RUNTIME, name="com.acme.foo")
public void processFoo(EventRequest request, EventResponse response) throws
PortletException, java.io.IOException {
        // process event foo
}
```

PLT.15

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

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

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

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

PLT.16

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 to 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.1.1~~PLT.16.1 PortletPreferences Interface

Portlets have access to their preferences attributes through the `PortletPreferences` interface. Portlets have access to the associated `PortletPreferences` object while they are processing requests. Portlets may only modify preferences attributes during a `processAction` or `processEvent` invocation.

Preference attributes are `String` array objects. Preferences attributes can be set to `null`.[cxxxiv]

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

- `getNames`
- `getValue`
- `setValue`
- `getValues`
- `setValues`
- `getMap`
- `isReadOnly`
- `reset`
- `store`

The `getMap` method returns an immutable `Map` of `String` keys and `String[]` values containing all current preference values. Preferences values must not be modified if the values in the `Map` are altered.[cxxxv] 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. If `setValues` method has been called with multiple values, the subsequent `setValue` method overwrites all existing values replacing them with the new single value.

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.[cxxxvi] It is left to the vendor to specify how and from where the default value is obtained.

If a preference attribute is read only, the `setValue`, `setValues` and `reset` methods must throw a `ReadOnlyException` when the portlet is in any of the standard modes.[cxxxvii]

The `store` method must persist all the changes made to the `PortletPreferences` object in the persistent store.[cxxxviii] 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.[cxxxix] 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` or `processEvent` method. [cxl] If the `store` method is invoked within the scope of a `render` or `serveResource` method invocation, it must throw an `IllegalStateException`.[cxli]

The `PortletPreferences` object must reflect the current values of the persistent store when the portlet container invokes the `processAction`, `processEvent`, `render` and ~~`r`~~`serveResource`~~`ender`~~ methods of the portlet. [cxlii]

~~PLT.1.2~~PLT.16.2 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.1.3~~PLT.16.3 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 read only.

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

```
<portlet>
...
  <!-- Portlet Preferences -->
  <portlet-preferences>
    <preference>
      <name>PreferredStockSymbols</name>
      <value>FOO</value>
      <value>XYZ</value>
      <read-only>true</read-only>
    </preference>
    <preference>
      <name>quotesFeedURL</name>
      <value>http://www.foomarket.com/quotes</value>
    </preference>
  </portlet-preferences>
</portlet>
```

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

Portal administration and configuration tools may use and change, default preference attributes when creating a new portlet preferences objects. In addition, the portal may further constraint the modifiability of preferences values.

~~PLT.1.3.1~~PLT.16.3.1 Localizing Preference Attributes

The Portlet Specification does not define a specific mechanism for localizing preference attributes. It leverages the J2SE 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 (see the *PLT.~~21~~24.10 Resource Bundles* Section).

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

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

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

~~PLT.1.4~~PLT.16.4 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>
    ...
    <preferences-validator>
    com.foo.portlets.XYZValidator
    </preferences-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.[cxlv] If the application is a distributed application, the portlet container must create an instance per portlet definition per VM.[cxlvi]

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.[cxlvii] 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.[cxlviii] If the validation is successful, the store operation must be completed.[cxlix] Portlet preferences ~~cannot~~should not be modified when they are being validated by a `PreferencesValidator` object. ~~If the store method is invoked within the scope of the `PreferenceValidator's` validate method invocation, an `IllegalStateException` must be thrown.~~

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.17

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.1.1~~PLT.17.1 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.[cl] 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.[cli]

~~PLT.1.2~~PLT.17.2 Session Scope

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

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.[cliii]

~~PLT.1.3~~PLT.17.3 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.[cliv] The portlet should take into account that objects that are stored in the application scope can be accessed by other portlets in parallel and thus should synchronize write access to these objects.

Portlets can allow portlet containers to share `APPLICATION_SCOPE` even beyond the current web application by declaring them as shared session parameters in the portlet deployment descriptor with the `<shared-session-attribute>` ~~tag~~element. Attributes that are not primitive Java types and should be shared across web applications must implement the `java.lang.Serializable` interface and be JAXB serializable (see ~~1.5~~PLT.23).

In order to ensure that application scope attributes are propagated in distributed session environment an explicit `setAttribute` or `removeAttribute` call must be done for changed attributes, even if these attributes can be changed implicitly like in the case of Java Collections.

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.[clv] The object must be stored in the `APPLICATION_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 be equal to the ID returned by the `PortletRequest.getWindowID()` method and not contain a '?' character.[clvi] `<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,PortletSession.APPLICATION_SCOPE);
session.setAttribute("bkg.color","RED",PortletSession.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 `PortletSessionUtil` 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 `decodeAttributeName` method of the `PortletSessionUtil` class allows retrieving the attribute name without any portlet-container fabricated prefix. Portlet developers should always use the `PortletSessionUtil` class to deal with attributes in the `PORTLET_SCOPE` when accessing them through the servlet API.

~~PLT.1.4~~PLT.17.4 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 container must ensure that all attributes placed in the `PortletSession` are also available in the `HttpSession` of the portlet application.~~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.[clvii] Conversely, data stored by portlets in the `PortletSession` in the portlet application scope is accessible to servlets and JSPs through the `HttpSession`. [clviii]

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

~~PLT.1.4.1~~PLT.17.4.1 HttpSession Method Mapping

The `getCreationTime`, `getId`, `getLastAccessedTime`, `getMaxInactiveInterval`, `invalidate`, `isNew` and `setMaxInactiveInterval` methods of the `PortletSession` interface must provide the same functionality as the methods of the `HttpSession` interface with identical names.

The `getAttribute`, `setAttribute`, `removeAttribute` and `getAttributeNames` methods of the `PortletSession` interface must provide the same functionality as the methods of the `HttpSession` interface with identical names adhering to the following rules:

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

NOTE: The JSR 286 EG seeks feedback on this feature of shared session attributes. Given that you can achieve a similar behavior with events is such a feature of value?

PLT.17.5 Shared session attributes

The portlet ~~can~~may define shared session attributes that can be shared across portlet web applications. Shared session attributes should be scoped at least to the current end user and thus allow a sharing of data on the portal level for a given user. They can be viewed as portal managed session data in contrary to the portlet session data, which is managed by the portlet container and only visible to artifacts within the same web application.

At a minimum shared session attributes must have the same visibility scope as non-shared ~~APPLICATION_SCOPE~~ session attributes, i.e. be accessible for all artifacts within the current web application.

It is up to the portal implementation to decide which portlets outside the current web application may share the same attributes. The portal should use additional information provided in the deployment descriptor, like the type and alias, in order to perform such a mapping between shared session attributes of different portlets.

The portlet should note that session sharing beyond the current web application is not reliable, e.g. ~~they may~~when connecting to remote systems that may currently not be available. It is not guaranteed that shared session attributes are at any point in time synchronized between different web applications as events like session time outs of a web application session may occur.

The propagation of shared attributes should be done by the portal / portlet container after a specific lifecycle method has finished (e.g. processAction), but within the scope of the current client request.

Shared session attributes are set, obtained and removed using the PORTLET_SCOPE or APPLICATION_SCOPE portlet session methods. If an attribute is set to null the portal should treat this value as being removed. This removal should be propagated to all other participants of the shared session. Session timeouts of one participant of a shared session attribute should not effect the value of this shared session attribute for participants that still have a valid session.

The HttpSession listeners also apply to the shared session attributes~~, like for normal APPLICATION_SCOPE attributes~~. The portlet should take into account that via shared

attributes a session creation or HttSession listener can be triggered from a portlet outside the current web application.

Shared attributes values must be serializable and in addition must have a valid JAXB binding (see also the ~~Overview~~JAXB chapter *~~XXXX~~PLT.26*) in order to allow a sharing with portlets running on different VMs or remotely via WSRP.

As a user may have many portlets in use and the storage for these shared attributes on the portal is definite, the portlet should try to minimize the amount of data it stores in ~~portal~~shared session attributes. If the portal runs out of storage space it may ignore or delay the propagation of the shared session attribute to other portlets outside the current web application. The same restriction applies to shared session attributes that are set via the rendering phase in either the `render` or `serveResource` method. In general it is strongly discouraged to set application scoped attributes in the rendering phase (see below).

Shared session attributes set in included servlets or JSPs should be treated like they were set in the portlet.

Components, like servlets, that access the shared session attributes outside the portlet context, i.e. not via a include from the portlet, should be able to read the attribute, but changes to the attribute are not likely to be propagated outside the current web application.

PLT.17.5.1 Declaration in the deployment descriptor

The portlet must declare shared session attributes in the portlet.xml deployment descriptor (see *~~Chapter XXXX~~PLT.24 Deployment Descriptor*). For sharing attributes ~~O~~on the application level the portlet must define the shared session attribute definition with the `<shared-application-session-attribute>` ~~tag~~element.

For sharing attributes on the portlet level the portlet must define the shared session attribute definition with the `<shared-portlet-session-attribute>` ~~tag~~element.

The shared session attribute definition must contain at least one attribute name. [clxiv] The portlet must use the first shared session attribute name entry in the portlet deployment descriptor as attribute name when accessing the portlet session.

The shared session attribute should uniquely identify the attribute and use the Java package naming standard (INSERT REF HERE) and character restrictions.

A localized display name for the shared application session attribute definition should be provided in the application level resource bundle (see *~~Chapter XXXX~~PLT.24.10*) with an entry of the name

`'javax.portlet.app.shared-application-session-attribute.<attribute-name>.display-name'.`

A localized display name for the shared portlet session attribute definition should be provided in the portlet resource bundle (see *~~Chapter XXXX~~PLT.24.10*) with an entry of the name

'javax.portlet.~~app.~~shared-portlet-session-attribute.<name>.display-name'.

PLT.17.5.2 Example

The following code snippets show an example for using shared application session attributes:

declaring the application session attribute in the deployment descriptor:

```
<shared-application-session-attribute>
        <name>com.acme.bar</name>
</shared-application-session-attribute>
```

using the shared application session attribute in the portlet:

```
@XmlRootElement
    public class Address implements Serializable
    {
       private String street;
       private String city;
       public void setStreet(String s) {street = s;}
       public String getStreet() { return street;}
       public void setCity(String c) { city = c;}
       public String getCity() { return city;}
    }

void processAction(ActionRequest req, ActionResponse resp)
{
...
Address sampleAddress = new Address();
sampleAddress.setStreet("myStreet");
sampleAddress.setCity("myCity");
PortletSession session = req.getPortletSession();
```

```
session.setAttribute("com.acme.bar", sampleAddress,
                     PortletSession.APPLICATION_SCOPE);
....
}
```

PLT.17.6 Writing to the Portlet Session

When writing to the portlet session the distinct lifecycle phases action and render should be taken into account, as writing in the render phase may create issues as explained below.

PLT.17.6.1 Process action and process event phase

Setting attributes in the action or event phase to the portlet session in the PORTLET_SCOPE will likely not create any concurrency issues. Setting attributes in the APPLICATION_SCOPE or the shared APPLICATION_SCOPE ~~may~~are more likely to create concurrency issues as these scopes are shared with other portlets that may run in parallel and also change the same attribute.

~~The~~A set or remove attribute call~~s~~ must be conducted as an atomic transaction~~s~~. The portlet container implementation is responsible for handling concurrent writes to avoid inconsistency in portlet session attributes.

PLT.17.6.2 Rendering phase

The portlet API allows portlets writing to the portlet session even in the rendering phase in either `render` or `serveResource`. The ability to write to the session in the rendering phase is merely introduced in order to allow easier migration of existing, servlet-based, web applications and the implementation of bridges frameworks that bridge from the portlet environment to web application frameworks.

In general the usage of the set methods on the portlet session in render is strongly discouraged as it breaks the concept of rendering being idempotent and re-playable. This is especially true for APPLICATION_SCOPE attributes and shared ~~APPLICATION_SCOPE~~ attributes as different portlets share these attributes. For shared session attributes the portal ~~may~~will likely not propagate the new settings to entities outside the current web application in the current client request. The changes may be propagated in a subsequent request.

~~PLT.1.5~~PLT.17.7 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.1.6~~PLT.17.8 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.1.7~~PLT.17.9 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.1.8~~PLT.17.10 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*.[clxv]

PLT.18

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.

Servlets and JSPs invoked from within portlet should generate markup fragments following the recommendations of the *PLT.B Markup Fragment* Appendix.

~~PLT.1.1~~PLT.18.1 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. [clxvi]

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`.[clxvii]

A `PortletRequestDispatcher` can be used in either the `render` or the `serveResource` method or any methods called by these methods, like `doView`.

~~PLT.1.1.1~~PLT.18.1.1 Query Strings in Request Dispatcher Paths

The `getRequestDispatcher` method of the `PortletContext` that creates `PortletRequestDispatcher` objects using path information allows the optional attachment of query string information to the path. For example, a Developer may obtain a `PortletRequestDispatcher` by using the following code:

```
String path = "/raisons.jsp?orderno=5";
PortletRequestDispatcher rd = context.getRequestDispatcher(path);
rd.include(renderRequest, renderResponse);
```

Parameters specified in the query string used to create the `PortletRequestDispatcher` must be aggregated with the portlet render parameters and take precedence over other portlet render parameters of the same name passed to the included servlet or JSP. The

parameters associated with a `PortletRequestDispatcher` are scoped to apply only for the duration of the include call.[clxviii]

~~PLT.1.2~~PLT.18.2 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 or the `serveResource` method of the `ResourceServingPortlet` interface or instances of the corresponding subclasses of the request and response wrapper classes that were introduced for version 2.0 of the specification. [clxix] In the latter case, the wrapper instances must wrap the request or response objects that the container passed into the `render` or `serveResource` method.[~~clxx~~]

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.[clxxi]

~~PLT.1.3~~PLT.18.3 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 or the `serveResource` method of the `ResourceServingPortlet` 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 its behavior may be non-deterministic.

Servlets and JSPs included from portlets must be handled as HTTP GET requests.[clxxii]

The lookup of the servlet given a path is done according to the servlet path matching rule defined in SRV.11 section of the servlet specification.

~~PLT.1.3.1~~PLT.18.3.1 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[clxxiii]:

```
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.1.3.2~~PLT.18.3.2 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

For includes from the render method the following additional attributes must be set:

Request Attribute	Type
javax.portlet.request	javax.portlet.RenderRequest

For includes from the serveResource method the following additional attribute must be set:

Request Attribute	Type
javax.portlet.request	javax.portlet.ResourceRequest

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

~~PLT.1.3.3~~PLT.18.3.3 Request and Response objects for Included Servlets/JSPs from within the Render method

The target servlet or JSP of portlet request dispatcher has access to a limited set of methods of the request and the response objects when the include is done from within the `render` method.

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

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`.[clxxvi]

The following methods of the `HttpServletRequest` must be equivalent to the methods of the `PortletRequest` of similar name: `getScheme`, `getServerName`, `getServerPort`, `getAttribute`, `getAttributeNames`, `setAttribute`, `removeAttribute`, `getLocale`, `getLocales`, `isSecure`, `getAuthType`, `getContextPath`, `getRemoteUser`, `getUserPrincipal`, `getRequestedSessionId`, `isRequestedSessionIdValid`.[clxxvii]

The following methods of the `HttpServletRequest` must be equivalent to the methods of the `PortletRequest` of similar name with the provision defined in *PLT.~~16~~18.1.1 Query Strings in Request Dispatcher Paths* Section: `getParameter`, `getParameterNames`, `getParameterValues` and `getParameterMap`.[clxxviii]

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

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`, `getDateHeader` and `getIntHeader`.[clxxxi].

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`.[clxxxii]

The `getMethod` method of the `HttpServletRequest` must always return '`GET`'.[clxxxiii]

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

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

The `getLocale` method of the `HttpServletResponse` must be based on the `getLocale` method of the `RenderResponse`.[clxxxvii]

PLT.18.3.4 Request and Response objects for Included Servlets/JSPs from within the ServeResource method

The target servlet or JSP of portlet request dispatcher has access to a limited set of methods of the request and the response objects when the include is done from within the `serveResource` method.

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

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.[clxxxix]

The following methods of the HttpServletRequest must be equivalent to the methods of the PortletRequest of similar name: getScheme, getServerName, getServerPort, getAttribute, getAttributeNames, setAttribute, removeAttribute, getLocale, getLocales, isSecure, getAuthType, getContextPath, getRemoteUser, getUserPrincipal, getRequestedSessionId, isRequestedSessionIdValid.[cxc]

The following methods of the HttpServletRequest must be equivalent to the methods of the ResourceRequest of similar name: getCharacterEncoding, setCharacterEncoding, getContentType and getReader. [cxci] The HttpServletRequest getInputStream must be equivalent to the method getPortletInputStream of the ResourceRequest.

The following methods of the HttpServletRequest must be equivalent to the methods of the PortletRequest of similar name with the provision defined in *PLT.18.1.1 Query Strings in Request Dispatcher Paths* Section: getParameter, getParameterNames, getParameterValues and getParameterMap.[cxcii]

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, getDateHeader and getIntHeader.[cxciii].

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

The getMethod method of the HttpServletRequest must always return 'GET'.[cxcv]

The following methods of the HttpServletResponse must return null: encodeRedirectURL and encodeRedirectUrl.[cxcvi]The following methods of the HttpServletResponse must be equivalent to the methods of the RenderResponse of similar name: getCharacterEncoding, setContentType, setBufferSize, flushBuffer, resetBuffer, reset, getBufferSize, isCommitted, getOutputStream, getWriter, encodeURL and encodeUrl.[cxcvii]

****

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

********* NOTE: the above section needs to be updated once we introduce support for HTTP headers **********

The `getLocale` method of the `HttpServletResponse` must be based on the `getLocale` method of the `RenderResponse`.[cxcix]

~~PLT.1.3.4~~PLT.18.3.5 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.[cc] 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.[cci]

PLT.18.4 Servlet filters and Request Dispatching

Since the Java Servlet Specification V2.4 you can specify servlet filters for request dispatcher include calls. Portlet container must support this capability for included servlets via the `PortletRequestDispatcher`. [ccii] The servlet filters for the servlets included via the `PortletRequestDispatcher` must be defined as described in the Java Servlet Specification. See SRV.6.2.5 in the Java Servlet Specification for more information.

PLT.19

Portlet Filter

Filters are Java components that allow on the fly transformations of information in both the request to and the response from a portlet.

PLT.19.1 What is a portlet filter?

A filter is a reusable piece of code that can transform the content of portlet requests and portlet responses. Filters do not generally create a response or respond to a request as portlets do, rather they modify or adapt the requests, and modify or adapt the response.

Among the types of functionality available to the developer needing to use filters are the following:

- The modification of request data by wrapping the request in customized versions of the request object.
- The modification of response data by providing customized versions of the response object.
- The interception of an invocation of a portlet after its call.

Portlet filters are modeled after the servlet filters in order to make them easy to understand for people already familiar with the servlet model and to have one consistent filter concept in JEE.

PLT.19.2 Main Concepts

The main concepts of this filtering model are described in this section. The application developer creates a filter by implementing the `javax.portlet.Filter` interface and providing a public constructor taking no arguments. The class is packaged in the portlet application WAR along with the static content and portlets that make up the portlet application. A filter is declared using the `<filter>` element in the portlet deployment descriptor. A filter or collection of filters can be configured for invocation by defining `<filter-mapping>` elements in the portlet deployment descriptor. This is done by mapping filters to a particular portlet by the portlet's logical name, or mapping to a group of portlets using the '*' as a wildcard.

PLT.19.2.1 Filter Lifecycle

After deployment of the portlet application, and before a request causes the portlet container to access a portlet, the portlet container must locate the list of portlet filters that

must be applied to the portlet as described below[cciii]. The portlet container must ensure that it has instantiated a filter of the appropriate class for each filter in the list, and called its

`init(FilterConfig config)` method[cciv]. The filter may throw an exception to indicate that it cannot function properly. If the exception is of type `UnavailableException`, the container may examine the `isPermanent` attribute of the exception and may choose to retry the filter at some later time.

Only one instance per `<filter>` declaration in the deployment descriptor is instantiated per Java Virtual Machine of the portlet container. The container provides the filter `config` as declared in the filter's deployment descriptor, the reference to the `PortletContext` for the portlet application, and the set of initialization parameters.

When the container receives an incoming request, it takes the first filter instance in the list and calls its `doFilter` method, passing in the `PortletRequest` and `PortletResponse`, and a reference to the `FilterChain` object it will use.

Depending on the target method of `doFilter` call the `PortletRequest` and `PortletResponse` must be instances of the following interfaces[ccv]:

- `ActionRequest` and `ActionResponse` for `processAction` calls
- `EventRequest` and `EventResponse` for `processEvent` calls
- `RenderRequest` and `RenderResponse` for `render` calls
- `ResourceRequest` and ~~`Render`~~`ResourceResponse` for `serveResource` calls

The `doFilter` method of a filter will typically be implemented following this or some subset of the following pattern:

1. The method examines the request information.
2. The method may wrap the request object passed in to its `doFilter` method with a customized implementation of one of the request wrappers (`ActionRequestWrapper, EventRequestWrapper, RenderRequestWrapper, ResourceRequestWrapper`) in order to modify request data.
3. The method may wrap the response object passed in to its `doFilter` method with a customized implementation of one of the response wrappers (`ActionResponse, EventResponse, RenderResponse`) to modify response data.
4. The filter may invoke the next component in the filter chain. The next component may be another filter, or if the filter making the invocation is the last filter configured in the deployment descriptor for this chain, the next component is the target method of the portlet. The invocation of the next component is effected by calling the `doFilter` method on the `FilterChain` object, and passing in the request and response with which it was called or passing in wrapped versions it may have created. The filter chain's implementation of the `doFilter` method, provided by the portlet container, must locate the next component in the filter chain and invoke its `doFilter` method, passing in the appropriate request and response objects. Alternatively, the filter chain can block the request by not making the call to invoke the next component, leaving the filter responsible for filling out the response object.

5. After invocation of the next filter in the chain, the filter may examine the response data.
6. Alternatively, the filter may have thrown an exception to indicate an error in processing. If the filter throws ~~an~~a `UnavailableException` during its `doFilter` processing, the portlet container must not attempt continued processing down the filter chain. It may choose to retry the whole chain at a later time if the exception is not marked permanent.
7. When the last filter in the chain has been invoked, the next component accessed is the target method on the portlet at the end of the chain.
8. Before a filter instance can be removed from service by the portlet container, the portlet container must first call the `destroy` method on the filter to enable the filter to release any resources and perform other cleanup operations. [ccvi]

PLT.19.2.2 Wrapping Requests and Responses

Central to the notion of filtering is the concept of wrapping a request or response in order that it can override behavior to perform a filtering task. In this model, the developer not only has the ability to override existing methods on the request and response objects, but to provide new API suited to a particular filtering task to a filter or the target portlet down the chain. In order to support this style of filter the container must support the following requirement. When a filter invokes the `doFilter` method on the portlet container's filter chain implementation, the container must ensure that the request and response object that it passes to the next component in the filter chain, or to the target portlet if the filter was the last in the chain, is the same object that was passed into the `doFilter` method by the calling filter or one of the above mentioned wrappers. [ccvii]

PLT.19.2.3 Filter Environment

A set of initialization parameters can be associated with a filter using the `<init-params>` element in the portlet deployment descriptor. The names and values of these parameters are available to the filter at runtime via the `getInitParameter` and `getInitParameterNames` methods on the filter's `FilterConfig` object. Additionally, the `FilterConfig` affords access to the `PortletContext` of the portlet application for the loading of resources, for logging functionality, and for storage of state in the `PortletContext`'s attribute list.

PLT.19.2.4 Configuration of Filters in a Portlet Application

A filter is defined in the deployment descriptor using the `<filter>` element. In this element, the programmer declares the following:

- `filter-name`: used to map the filter to a portlet
- `filter-class`: used by the portlet container to identify the filter type
- `init-params`: initialization parameters for a filter

Optionally, the programmer can specify a textual description, and a display name for tool manipulation. The portlet container must instantiate exactly one instance of the Java class defining the filter per filter declaration in the deployment descriptor[ccviii]. Hence, two

instances of the same filter class will be instantiated by the portlet container if the developer makes two filter declarations for the same filter class.

Here is an example of a filter declaration:

```
<filter>
      <filter-name>Log Filter</filter-name>
      <filter-class>com.acme.LogFilter</filter-class>
</filter>
```

Once a filter has been declared in the portlet deployment descriptor, the `<filter-mapping>` element is used to define portlets in the portlet application to which the filter is to be applied. Filters can be associated with a portlet using the `<portlet-name>` element. For example, the following code example maps the Log Filter `filter` to the `SamplePortlet` portlet:

```
<filter-mapping>
      <filter-name>Log Filter</filter-name>
      <portlet-name>SamplePortlet</portlet-name>
</filter-mapping>
```

Filters can be associated with groups of portlets using the '*' character as a wildcard at the end of a string to indicate that the filter must be applied to any portlet whose name starts with the characters before the "*" character[ccix]. Example:

```
<filter-mapping>
      <filter-name>Log Filter</filter-name>
      <portlet-name>*</portlet-name>
</filter-mapping>
```

Here the Log Filter is applied to all the portlets portlet application, because every portlet name matches the '*' pattern.

The order the container uses in building the chain of filters to be applied for a particular request is as follows: the `<portlet-name>` matching filter mappings in the same order that these elements appear in the deployment descriptor. The portlet container is free to add additional filters at any place in this filter chain, but must not remove filters matching a specific portlet. [ccx].

It is expected that high performance portlet containers will cache filter chains so that they do not need to compute them on a per-request basis.

PLT.19.2.5 Defining the Target Lifecycle Method for a Portlet Filter

Per default a defined portlet filter matching a portlet must be applied to all lifecycle method calls: `processAction`, `processEvent`, `render`, `serveResource`[ccxi]. In case the filter should only be applied to a subset of the lifecycle methods the `<lifecycle>` element in the `<filter-mapping>` element can be used. The following constants are valid values for the `<lifecycle>` element:

- `ACTION` requesting that the portlet container processes this filter for the `processAction` lifecycle method.
- `EVENT` requesting that the portlet container processes this filter for the `processEvent` lifecycle method.
- `RENDER` requesting that the portlet container processes this filter for the `render` lifecycle method.
- `RESOURCE` requesting that the portlet container processes this filter for the `serveResource` lifecycle method.

The portlet container must apply the matching filter for at least the lifecycle phases defined in the `<lifecycle>` elements, but is free to apply the matching filter to additional lifecycle methods[ccxii]. The filter implementation should take this possibility into account.

~~PLT.19~~PLT.20 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.19.1~~PLT.20.1 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.[ccxiii]

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

~~PLT.19.2~~PLT.20.2 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`.[ccxiv]. 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..[ccxv]

An example of a portlet retrieving user attributes would be:

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

~~PLT.19.3~~PLT.20.3 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 no 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 a Java standard for user information is defined, the current mechanism will be deprecated in favor of it.

~~PLT.1~~PLT.21 Caching

Caching content helps improve the Portal response time for users. 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 user client. Cached content must not be shared across different user clients 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.1.1~~PLT.21.1 Expiration Cache

Portlets that want their content to be cached using expiration cache ~~must~~ should define the default duration (in seconds) of the expiration cache in the deployment descriptor. The portlet container should treat portlets with no default duration in the deployment descriptor as always expired as default.

The following is an example of a portlet definition where the portlet defines that its content should be cached 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 a property in the `RenderResponse` object using the~~—~~ `EXPIRATION_CACHE` constant defined in the ~~`PortletResponse`~~ `RenderResponse` interface. If the expiration property is set to 0, ~~caching is disabled for the portlet~~the returned markup fragment should be treated as always expired. 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~~ should 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 an action or event ~~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 is target of an action or event call~~,~~ the portlet container ~~must~~ should discard the cache and invoke the corresponding request handling methods of the portlet (`processAction` or `processEvent`).

~~PLT.21~~PLT.22

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.21.1~~PLT.22.1 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.21.2~~PLT.22.2 Relationship to PortletContext

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

~~PLT.21.3~~PLT.22.3 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.21.4~~PLT.22.4 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.21.5~~PLT.22.5 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.[ccxviii]

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.[ccxix]

~~PLT.21.6~~PLT.22.6 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.21.7~~PLT.22.7 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 *PLT.21 Packaging and Deployment Descriptor* Chapter for more details on the portlet application deployment descriptor.

~~PLT.21.8~~PLT.22.8 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, when the replacement of the portlet application happens.

~~PLT.21.9~~PLT.22.9 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.21.10~~PLT.22.10 Portlet Application Environment

The Portlet Specification leverages the provisions made by the *Servlet Specification ~~2.3~~ SRV.9.11* Section.

~~PLT.1~~PLT.23 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.1.1~~PLT.23.1 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 accessing 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.1.2~~PLT.23.2 Roles

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

~~PLT.1.3~~PLT.23.3 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.[ccxx] 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.[ccxxi]

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~~role-link>
        </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.[ccxxii] 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.1.4~~PLT.23.4 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 confi~~n~~dential transport the syntax would be:

```
<portlet-app>
    ...
    <portlet>
        <portlet-name>accountSummary</portlet-name>
        ...
    </portlet>
    ...
    <security-constraint>
         <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>
    ...
</portlet-app>
```

~~PLT.1.5~~PLT.23.5 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.[ccxxiii] The security role name must be one of the security role names defined for the `web.xml` deployment descriptor.[ccxxiv] Alternatively, portlet application code may be the sole processor of the signon into the EJB™ container.

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~~4, SRV.13Deployment Descriptor* Chapter.

For backwards compatibility of portlet applications written to the 1.0 version of the Java Portlet Specification, portlet containers are also required to support the 1.0 version of the

deployment descriptor. The 1.0 version is defined in the appendix.

~~PLT.1.1~~PLT.24.1 Portlet and Web Application Deployment Descriptor

~~For~~ In 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~~element
- portlet application name using the `<display-name>` ~~tag~~element
- portlet application security role mapping using the `<security-role>` ~~tag~~element

~~PLT.1.2~~PLT.24.2 Packaging

All resources, portlets and the deployment descriptors are packaged 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.1.2.1~~PLT.24.2.1 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.1.2.2~~PLT.24.2.2 Version Information

If portlet application providers want to provide version information about the portlet application it 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. The version information should follow the format defined by the Java Product Versioning Specification (http://java.sun.com/j2se/1.4/pdf/versioning.pdf)

Example:

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

~~PLT.1.3~~PLT.24.3 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.1.4~~PLT.24.4 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.24.5 Portlet Deployment Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp1 U (http://www.xmlspy.com) by Stefan Hepper (IBM Entwicklung GmbH) -->
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:portlet="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd" elementFormDefault="qualified" attributeFormDefault="unqualified" version="2.0" xml:lang="en">
    <annotation>
        <documentation>
    is the XML Schema for the Portlet 2.0 deployment descriptor.
        </documentation>
    </annotation>
    <annotation>
        <documentation>
```

```
        following conventions apply to all J2EE
        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>
        <!-- ********************************************************** -->
        <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>
                portlet-app element is the root of the deployment descriptor
                a portlet application. This element has a required attribute version
                specify to which version of the schema the deployment descriptor
                .
                </documentation>
            </annotation>
            <unique name="portlet-name-uniqueness">
                <annotation>
                    <documentation>
                    portlet element contains the name of a portlet.
                    name must be unique within the portlet application.
                    </documentation>
                </annotation>
```

```
            <selector xpath="portlet:portlet"/>
            <field xpath="portlet:portlet-name"/>
        </unique>
        <unique name="custom-portlet-mode-uniqueness">
            <annotation>
                <documentation>
            custom-portlet-mode element contains the portlet-mode.
            portlet mode must be unique within the portlet application.
                </documentation>
            </annotation>
            <selector xpath="portlet:custom-portlet-mode"/>
            <field xpath="portlet:portlet-mode"/>
        </unique>
        <unique name="custom-window-state-uniqueness">
            <annotation>
                <documentation>
            custom-window-state element contains the window-state.
            window state must be unique within the portlet application.
                </documentation>
            </annotation>
            <selector xpath="portlet:custom-window-state"/>
            <field xpath="portlet:window-state"/>
        </unique>
        <unique name="user-attribute-name-uniqueness">
            <annotation>
                <documentation>
            user-attribute element contains the name the attribute.
            name must be unique within the portlet application.
                </documentation>
            </annotation>
            <selector xpath="portlet:user-attribute"/>
```

```
                        <field xpath="portlet:name"/>
                    </unique>
                </element>
                <complexType name="portlet-appType">
                    <sequence>
                        <element name="portlet" type="portlet:portletType" minOccurs="0" maxOccurs="unbounded">
                            <unique name="init-param-name-uniqueness">
                                <annotation>
                                    <documentation>
                                    init-param element contains the name the attribute.
                                    name must be unique within the portlet.
                                    </documentation>
                                </annotation>
                                <selector xpath="portlet:init-param"/>
                                <field xpath="portlet:name"/>
                            </unique>
                            <unique name="supports-mime-type-uniqueness">
                                <annotation>
                                    <documentation>
                                    supports element contains the supported mime-type.
                                    mime type must be unique within the portlet.
                                    </documentation>
                                </annotation>
                                <selector xpath="portlet:supports"/>
                                <field xpath="mime-type"/>
                            </unique>
                            <unique name="preference-name-uniqueness">
                                <annotation>
                                    <documentation>
                                    preference element contains the name the preference.
```

```
                name must be unique within the portlet.
                    </documentation>
                </annotation>
                <selector xpath="portlet:portlet-preferences/portlet:preference"/>
                <field xpath="portlet:name"/>
            </unique>
            <unique name="security-role-ref-name-uniqueness">
                <annotation>
                    <documentation>
                security-role-ref element contains the role-name.
                role name must be unique within the portlet.
                    </documentation>
                </annotation>
                <selector xpath="portlet:security-role-ref"/>
                <field xpath="portlet:role-name"/>
            </unique>
        </element>
        <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"/>
        <element name="event-definition" type="portlet:event-definitionType" minOccurs="0"
maxOccurs="unbounded"/>
        <element name="shared-application-session-attribute" type="portlet:shared-session-
attributeType" minOccurs="0" maxOccurs="unbounded"/>
        <element name="shared-render-parameter" type="portlet:shared-render-parameterType"
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>
                custom portlet mode that one or more portlets in
                portlet application supports.
                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>
                custom window state that one or more portlets in this
                application supports.
                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>
                cache defines expiration-based caching for this
                . The parameter indicates
                time in seconds after which the portlet output expires.
                    indicates that the output never expires.
                in: portlet
                    </documentation>
                </annotation>
                <simpleContent>
                    <extension base="int"/>
                </simpleContent>
        </complexType>
        <complexType name="init-paramType">
                <annotation>
                    <documentation>
                init-param element contains a name/value pair as an
                param of the portlet
                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>
            specific keywords associated with this portlet.
            kewords are separated by commas.
            in: portlet-info
                </documentation>
        </annotation>
        <simpleContent>
                <extension base="string"/>
        </simpleContent>
    </complexType>
    <complexType name="mime-typeType">
        <annotation>
                <documentation>
            type name, e.g. "text/html".
            MIME type may also contain the wildcard
            '*', like "text/*" or "*/*".
            in: supports
                </documentation>
        </annotation>
        <simpleContent>
                <extension base="string"/>
        </simpleContent>
    </complexType>
    <complexType name="nameType">
        <annotation>
                <documentation>
            name element contains the name of a parameter.
            in: init-param, ...
                </documentation>
        </annotation>
        <simpleContent>
```

```
                    <extension base="string"/>
                </simpleContent>
            </complexType>
            <complexType name="portletType">
                <annotation>
                    <documentation>
                portlet element contains the declarative data of a portlet.
                in: portlet-app
                    </documentation>
                </annotation>
                <sequence>
                    <element name="description" type="portlet:descriptionType" 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="init-param" type="portlet:init-paramType" minOccurs="0" maxOccurs="unbounded"/>
                    <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"/>
                    <choice>
                        <sequence>
                            <element name="resource-bundle" type="portlet:resource-bundleType"/>
                            <element name="portlet-info" type="portlet:portlet-infoType" minOccurs="0"/>
                        </sequence>
                        <element name="portlet-info" type="portlet:portlet-infoType"/>
                    </choice>
                    <element name="portlet-preferences" type="portlet:portlet-preferencesType" minOccurs="0"/>
```

```
                <element name="security-role-ref" type="portlet:security-role-refType" minOccurs="0" maxOccurs="unbounded"/>
                <element name="supported-processing-event" type="portlet:nameType" minOccurs="0" maxOccurs="unbounded"/>
                <element name="supported-publishing-event" type="portlet:nameType" minOccurs="0" maxOccurs="unbounded"/>
                <element name="supported-shared-render-parameter" type="portlet:nameType" minOccurs="0" maxOccurs="unbounded"/>
                <element name="shared-portlet-session-attribute" type="portlet:shared-session-attributeType" minOccurs="0" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="id" type="string" use="optional"/>
        </complexType>
        <simpleType name="portlet-classType">
            <annotation>
                <documentation>
                The portlet-class element contains the fully
                qualified class name of the portlet.
            in: portlet
                </documentation>
            </annotation>
            <restriction base="portlet:fully-qualified-classType"/>
        </simpleType>
        <complexType name="portlet-collectionType">
            <annotation>
                <documentation>
            portlet-collectionType is used to identify a subset
            portlets within a portlet application to which a
            constraint applies.
            in: security-constraint
                </documentation>
            </annotation>
            <sequence>
```

```
                <element name="portlet-name" type="portlet:portlet-nameType"
maxOccurs="unbounded"/>
            </sequence>
        </complexType>
        <complexType name="event-definitionType">
            <annotation>
                <documentation>
            event-definitionType is used to declare events the portlet can either
            or emit.
            first name element is treated as prefered name and must be the one the
            is using in its code for referencing this event.
            in: portlet-app
                </documentation>
            </annotation>
            <sequence>
                <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="name" type="portlet:nameType" maxOccurs="unbounded"/>
                <choice>
                    <sequence>
                        <element name="xml-schema" type="string"/>
                        <element name="jaxb-mapping" type="string" minOccurs="0"/>
                    </sequence>
                    <element name="java-class" type="portlet:fully-qualified-classType"/>
                </choice>
            </sequence>
            <attribute name="id" type="string" use="optional"/>
        </complexType>
        <complexType name="portlet-infoType">
            <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>
            <attribute name="id" type="string" use="optional"/>
        </complexType>
        <complexType name="portlet-modeType">
            <annotation>
                <documentation>
            modes. The specification pre-defines the following values
            valid portlet mode constants:
                "edit", "help", "view".
            mode names are not case sensitive.
            in: custom-portlet-mode, supports
                </documentation>
            </annotation>
            <simpleContent>
                <extension base="string"/>
            </simpleContent>
        </complexType>
        <complexType name="portlet-nameType">
            <annotation>
                <documentation>
            portlet-name element contains the canonical name of the
            . Each portlet name is unique within the portlet
            .
            in: portlet, portlet-mapping
                </documentation>
            </annotation>
            <simpleContent>
                <extension base="string"/>
            </simpleContent>
        </complexType>
```

```
    <complexType name="portlet-preferencesType">
        <annotation>
            <documentation>
        persistent preference store.
        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>
        preference values that may be used for customization
        personalization by the portlet.
        in: portlet-preferences
            </documentation>
        </annotation>
        <sequence>
            <element name="name" type="portlet:nameType"/>
            <element name="value" type="portlet:valueType" minOccurs="0" maxOccurs="unbounded"/>
            <element name="read-only" type="portlet:read-onlyType" minOccurs="0"/>
        </sequence>
        <attribute name="id" type="string" use="optional"/>
    </complexType>
    <simpleType name="preferences-validatorType">
```

```
                <annotation>
                        <documentation>
                class specified under preferences-validator implements
                PreferencesValidator interface to validate the
                settings.
                in: portlet-preferences
                        </documentation>
                </annotation>
                <restriction base="portlet:fully-qualified-classType"/>
        </simpleType>
        <simpleType name="read-onlyType">
                <annotation>
                        <documentation>
                only indicates that a setting cannot
                changed in any of the standard portlet modes
                        ("view","edit" or "help").
                default all preferences are modifiable.
                values are:
                        true for read-only
                        false for modifiable
                in: preferences
                        </documentation>
                </annotation>
                <restriction base="portlet:string">
                        <enumeration value="true"/>
                        <enumeration value="false"/>
                </restriction>
        </simpleType>
        <complexType name="resource-bundleType">
                <annotation>
                        <documentation>
```

```
                of the resource bundle containing the language specific
                informations in different languages.
                in: portlet-info
                        </documentation>
                </annotation>
                <simpleContent>
                        <extension base="string"/>
                </simpleContent>
        </complexType>
        <complexType name="role-linkType">
                <annotation>
                        <documentation>
                role-link element is a reference to a defined security role.
                role-link element must contain the name of one of the
                roles defined in the security-role elements.
                in: security-role-ref
                        </documentation>
                </annotation>
                <simpleContent>
                        <extension base="string"/>
                </simpleContent>
        </complexType>
        <complexType name="security-constraintType">
                <annotation>
                        <documentation>
                security-constraintType is used to associate
                security constraints with one or more portlets.
                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"/>
                        <element name="user-data-constraint" type="portlet:user-data-constraintType"/>
                </sequence>
                <attribute name="id" type="string" use="optional"/>
        </complexType>
        <complexType name="security-role-refType">
                <annotation>
                        <documentation>
                security-role-ref element contains the declaration of a
                role reference in the code of the web application. The
                consists of an optional description, the security
                name used in the code, and an optional link to a security
                . If the security role is not specified, the Deployer must
                an appropriate security role.
                value of the role name element must be the String used
                the parameter to the
                .isCallerInRole(String roleName) method
                the HttpServletRequest.isUserInRole(String role) method.
                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="shared-render-parameterType">
```

```
            <annotation>
                <documentation>
            shared-render-parameters defines a render parameter that is allowed to be shared
            other portlets.
            in: portlet-app
                </documentation>
            </annotation>
            <sequence>
                <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="name" type="portlet:nameType" maxOccurs="unbounded"/>
            </sequence>
            <attribute name="id" type="string" use="optional"/>
        </complexType>
        <complexType name="shared-session-attributeType">
            <annotation>
                <documentation>
            shared-session-attribute defines an attribute that is allowed to be shared
            the current web application. The attribute to be shared can be either
            application scoped session attribute or a portlet scoped session attribute.
            application scoped session attribute is defined on the portlet-app level
            the shared-application-session-attribute. The portlet scoped session
            is defined on the portlet level via the shared-portlet-session-attribute.
            in: portlet-app, portlet
                </documentation>
            </annotation>
            <sequence>
                <element name="description" type="portlet:descriptionType" minOccurs="0"
maxOccurs="unbounded"/>
                <element name="name" type="portlet:nameType" maxOccurs="unbounded"/>
                <choice>
                    <sequence>
```

```
                                        <element name="xml-schema" type="string"/>
                                        <element name="jaxb-mapping" type="string" minOccurs="0"/>
                                    </sequence>
                                    <element name="java-class" type="portlet:fully-qualified-classType"/>
                                </choice>
                            </sequence>
                            <attribute name="id" type="string" use="optional"/>
                        </complexType>
                        <complexType name="short-titleType">
                            <annotation>
                                <documentation>
                            specific short version of the static title.
                            in: portlet-info
                                </documentation>
                            </annotation>
                            <simpleContent>
                                <extension base="string"/>
                            </simpleContent>
                        </complexType>
                        <complexType name="supportsType">
                            <annotation>
                                <documentation>
                            indicates the portlet modes a
                            supports for a specific content type. All portlets must
                            the view mode.
                            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>
            the locales the portlet supports.
            in: portlet
                </documentation>
            </annotation>
            <simpleContent>
                <extension base="string"/>
            </simpleContent>
        </complexType>
        <complexType name="titleType">
            <annotation>
                <documentation>
            specific static title for this portlet.
            in: portlet-info
                </documentation>
            </annotation>
            <simpleContent>
                <extension base="string"/>
            </simpleContent>
        </complexType>
        <simpleType name="transport-guaranteeType">
            <annotation>
                <documentation>
            transport-guaranteeType specifies that
```

```
                communication between client and portlet should
                NONE, INTEGRAL, or CONFIDENTIAL.
                means that the portlet does not
                any transport guarantees. A value of
                means that the portlet requires that the
                sent between the client and portlet be sent in
                a way that it can't be changed in transit.
                means that the portlet requires
                the data be transmitted in a fashion that
                other entities from observing the contents
                the transmission.
                most cases, the presence of the INTEGRAL or
                flag will indicate that the use
                SSL is required.
                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>
                attribute defines a user specific attribute that the
                application needs. The portlet within this application
                access this attribute via the request parameter USER_INFO
                .
                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>
            user-data-constraintType is used to indicate how
            communicated between the client and portlet should be
            .
            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>
            value element contains the value of a parameter.
            in: init-param
                </documentation>
            </annotation>
```

```
                <simpleContent>
                        <extension base="string"/>
                </simpleContent>
        </complexType>
        <complexType name="window-stateType">
                <annotation>
                        <documentation>
                window state. Window state names are not case sensitive.
                in: custom-window-state
                        </documentation>
                </annotation>
                <simpleContent>
                        <extension base="string"/>
                </simpleContent>
        </complexType>
        <!--- everything below is copied from j2ee_1_4.xsd -->
        <complexType name="descriptionType">
                <annotation>
                        <documentation>
                description element is used to provide text describing the
                element. The description element should include any
                that the portlet application war file producer wants
                provide to the consumer of the portlet application war file
                        (i.e., to the Deployer). Typically, the tools used by the
                application war file consumer will display the
                when processing the parent element that contains the
                . It has an optional attribute xml:lang to indicate
                language is used in the description according to
                1766 (http://www.ietf.org/rfc/rfc1766.txt). The default
                of this attribute is English(“en”).
                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>
            display-name type contains a short name that is intended
            be displayed by tools. It is used by display-name
            .  The display name need not be unique.
            :
                    ...
                <display-name xml:lang="en">Employee Self Service</display-name>

            has an optional attribute xml:lang to indicate
            language is used in the description according to
            1766 (http://www.ietf.org/rfc/rfc1766.txt). The default
            of this attribute is English(“en”).
                </documentation>
            </annotation>
            <simpleContent>
                <extension base="portlet:string">
                    <attribute ref="xml:lang"/>
                </extension>
            </simpleContent>
        </complexType>
        <simpleType name="fully-qualified-classType">
```

```
            <annotation>
                <documentation>
            elements that use this type designate the name of a
            class or interface.
                </documentation>
            </annotation>
            <restriction base="portlet:string"/>
        </simpleType>
        <simpleType name="role-nameType">
            <annotation>
                <documentation>
            role-nameType designates the name of a security role.

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

- </simpleType>
- </schema>

~~PLT.23.5Deployment Descriptor~~

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:portlet="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
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>
  <!-- ********************************************************** -->
  <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. This element has a required attribute version
      to specify to which version of the schema the deployment descriptor
      conforms.
      </documentation>
    </annotation>
    <unique name="portlet-name-uniqueness">
      <annotation>
        <documentation>
        The portlet element contains the name of a portlet.
        This name must be unique within the portlet application.
        </documentation>
      </annotation>
      <selector xpath="portlet:portlet"/>
      <field xpath="portlet:portlet-name"/>
    </unique>
    <unique name="custom-portlet-mode-uniqueness">
      <annotation>
        <documentation>
        The custom-portlet-mode element contains the portlet-mode.
        This portlet mode must be unique within the portlet application.
        </documentation>
      </annotation>
      <selector xpath="portlet:custom-portlet-mode"/>
      <field xpath="portlet:portlet-mode"/>
    </unique>
    <unique name="custom-window-state-uniqueness">
      <annotation>
        <documentation>
        The custom-window-state element contains the window-state.
        This window state must be unique within the portlet application.
```

```
      </documentation>
    </annotation>
    <selector xpath="portlet:custom-window-state"/>
    <field xpath="portlet:window-state"/>
  </unique>
  <unique name="user-attribute-name-uniqueness">
    <annotation>
      <documentation>
      The user-attribute element contains the name the attribute.
      This name must be unique within the portlet application.
      </documentation>
    </annotation>
    <selector xpath="portlet:user-attribute"/>
    <field xpath="portlet:name"/>
  </unique>
 </element>
 <complexType name="portlet-appType">
  <sequence>
    <element name="portlet" type="portlet:portletType" minOccurs="0"
maxOccurs="unbounded">
      <unique name="init-param-name-uniqueness">
        <annotation>
          <documentation>
          The init-param element contains the name the attribute.
          This name must be unique within the portlet.
          </documentation>
        </annotation>
        <selector xpath="portlet:init-param"/>
        <field xpath="portlet:name"/>
      </unique>
      <unique name="supports-mime-type-uniqueness">
        <annotation>
          <documentation>
          The supports element contains the supported mime-type.
          This mime type must be unique within the portlet.
          </documentation>
        </annotation>
        <selector xpath="portlet:supports"/>
        <field xpath="mime-type"/>
      </unique>
      <unique name="preference-name-uniqueness">
        <annotation>
          <documentation>
          The preference element contains the name the preference.
          This name must be unique within the portlet.
          </documentation>
        </annotation>
        <selector xpath="portlet:portlet-preferences/portlet:preference"/>
        <field xpath="portlet:name"/>
      </unique>
      <unique name="security-role-ref-name-uniqueness">
        <annotation>
          <documentation>
          The security-role-ref element contains the role-name.
          This role name must be unique within the portlet.
          </documentation>
        </annotation>
        <selector xpath="portlet:security-role-ref"/>
        <field xpath="portlet:role-name"/>
      </unique>
    </element>
    <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="int"/>
    </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>
  <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="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="init-param" type="portlet:init-paramType" minOccurs="0"
maxOccurs="unbounded"/>
      <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"/>
      <choice>
        <sequence>
          <element name="resource-bundle" type="portlet:resource-bundleType"/>
          <element name="portlet-info" type="portlet:portlet-infoType"
minOccurs="0"/>
        </sequence>
        <element name="portlet-info" type="portlet:portlet-infoType"/>
      </choice>
      <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>
  <simpleType name="portlet-classType">
    <annotation>
      <documentation>
       The portlet-class element contains the fully
       qualified class name of the portlet.
      Used in: portlet
      </documentation>
    </annotation>
    <restriction base="portlet:fully-qualified-classType"/>
  </simpleType>
```

```
  <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">
    <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>
    <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: portlet-preferences
      </documentation>
    </annotation>
```

```
    <sequence>
      <element name="name" type="portlet:nameType"/>
      <element name="value" type="portlet:valueType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="read-only" type="portlet:read-onlyType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <simpleType name="preferences-validatorType">
    <annotation>
      <documentation>
      The class specified under preferences-validator implements
      the PreferencesValidator interface to validate the
      preferences settings.
      Used in: portlet-preferences
      </documentation>
    </annotation>
    <restriction base="portlet:fully-qualified-classType"/>
  </simpleType>
  <simpleType name="read-onlyType">
    <annotation>
      <documentation>
      read-only indicates that a setting cannot
      be changed in any of the standard portlet modes
      ("view","edit" or "help").
      Per default all preferences are modifiable.
      Valid values are:
      - true for read-only
      - false for modifiable
      Used in: preferences
      </documentation>
    </annotation>
    <restriction base="portlet:string">
      <enumeration value="true"/>
      <enumeration value="false"/>
    </restriction>
  </simpleType>
  <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"/>
      <element name="user-data-constraint" type="portlet:user-data-
constraintType"/>
    </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 code of the web application. 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 a
      portlet supports for a specific content type. All portlets must
      support the view mode.
      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. 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.xsd --->
  <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. It has an optional attribute xml:lang to indicate
      which language is used in the description according to
      RFC 1766 (http://www.ietf.org/rfc/rfc1766.txt). The default
      value of this attribute is English("en").
      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>

      It has an optional attribute xml:lang to indicate
      which language is used in the description according to
      RFC 1766 (http://www.ietf.org/rfc/rfc1766.txt). The default
      value of this attribute is English("en").
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet:string">
        <attribute ref="xml:lang"/>
      </extension>
    </simpleContent>
  </complexType>
  <simpleType name="fully-qualified-classType">
    <annotation>
      <documentation>
      The elements that use this type designate the name of a
      Java class or interface.
      </documentation>
```

```
    </annotation>
    <restriction base="portlet:string"/>
  </simpleType>
  <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.24.6 Pictures of the structure of a Deployment Descriptor

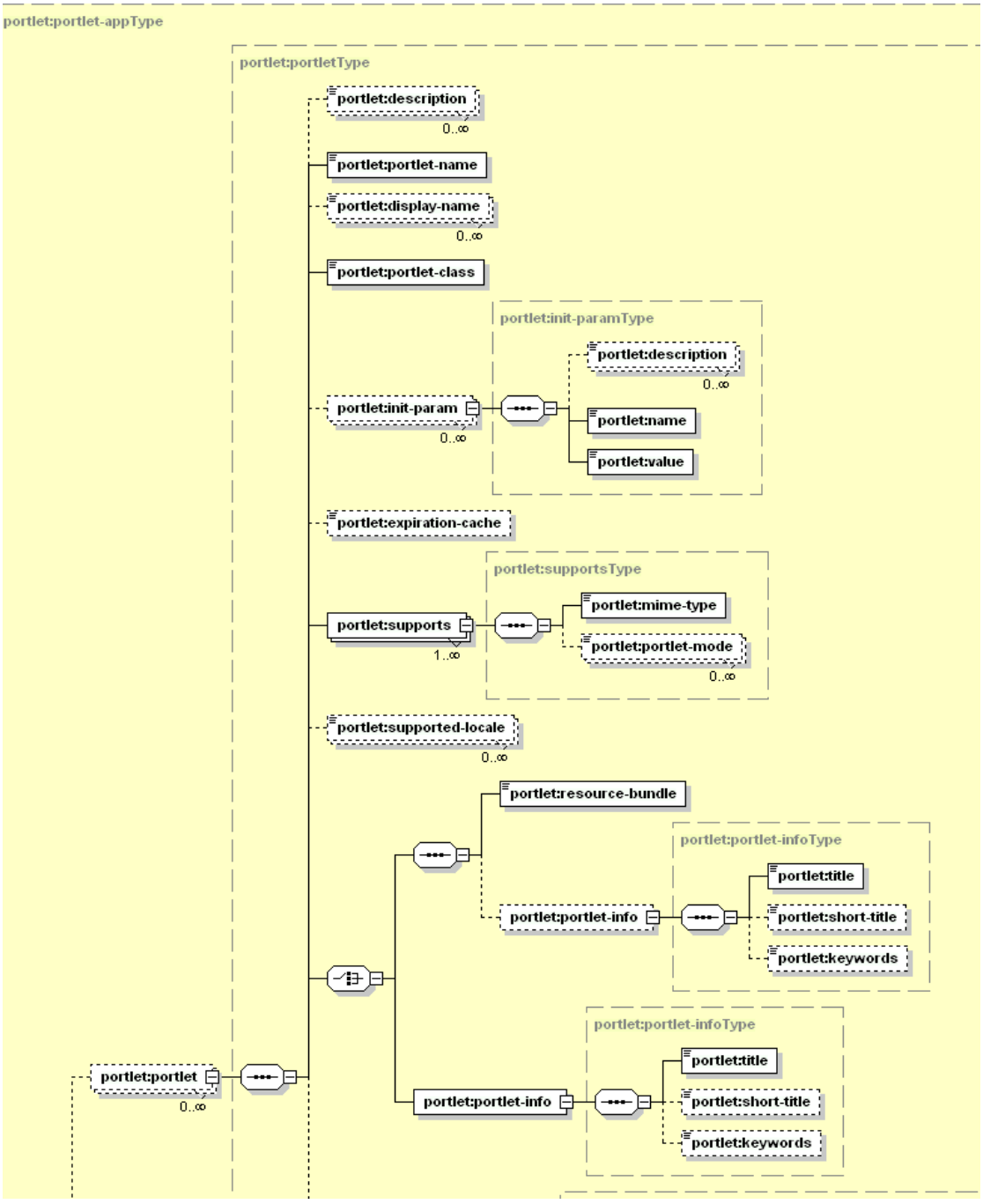


Figure 1: Part one of the portlet element

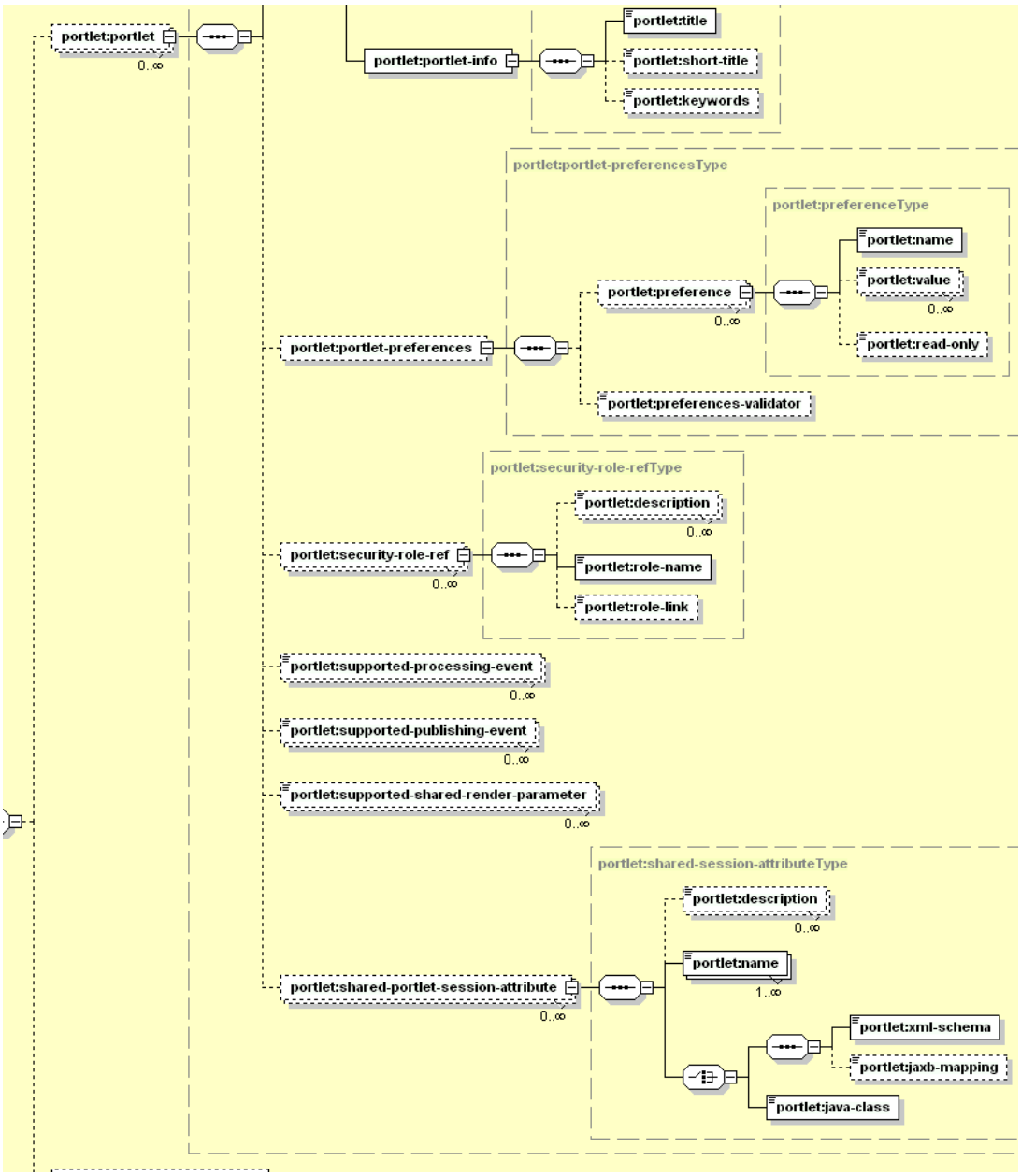


Figure 2: Part 2 of the portlet element

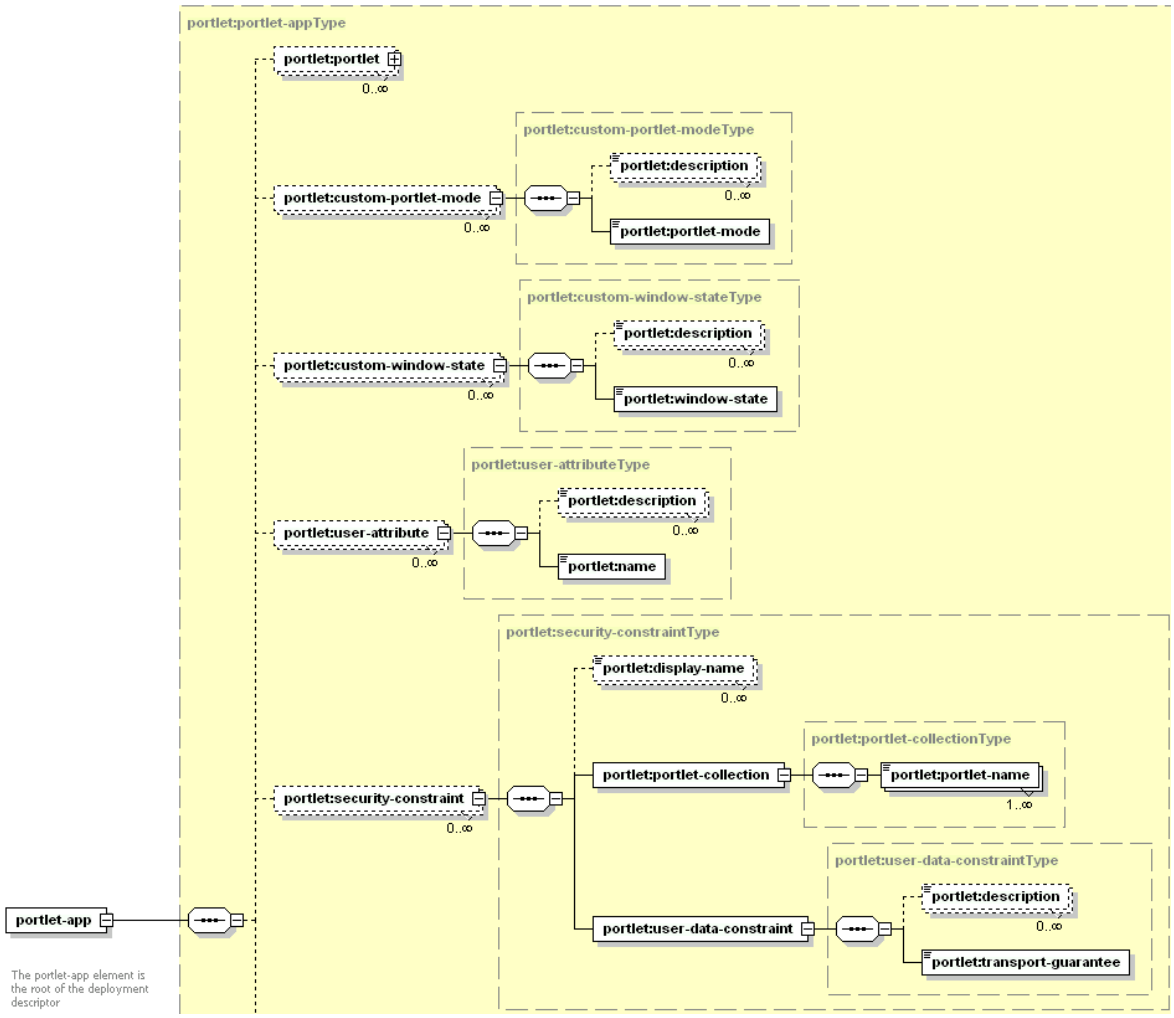


Figure 3: Part 1 of the portlet-app element

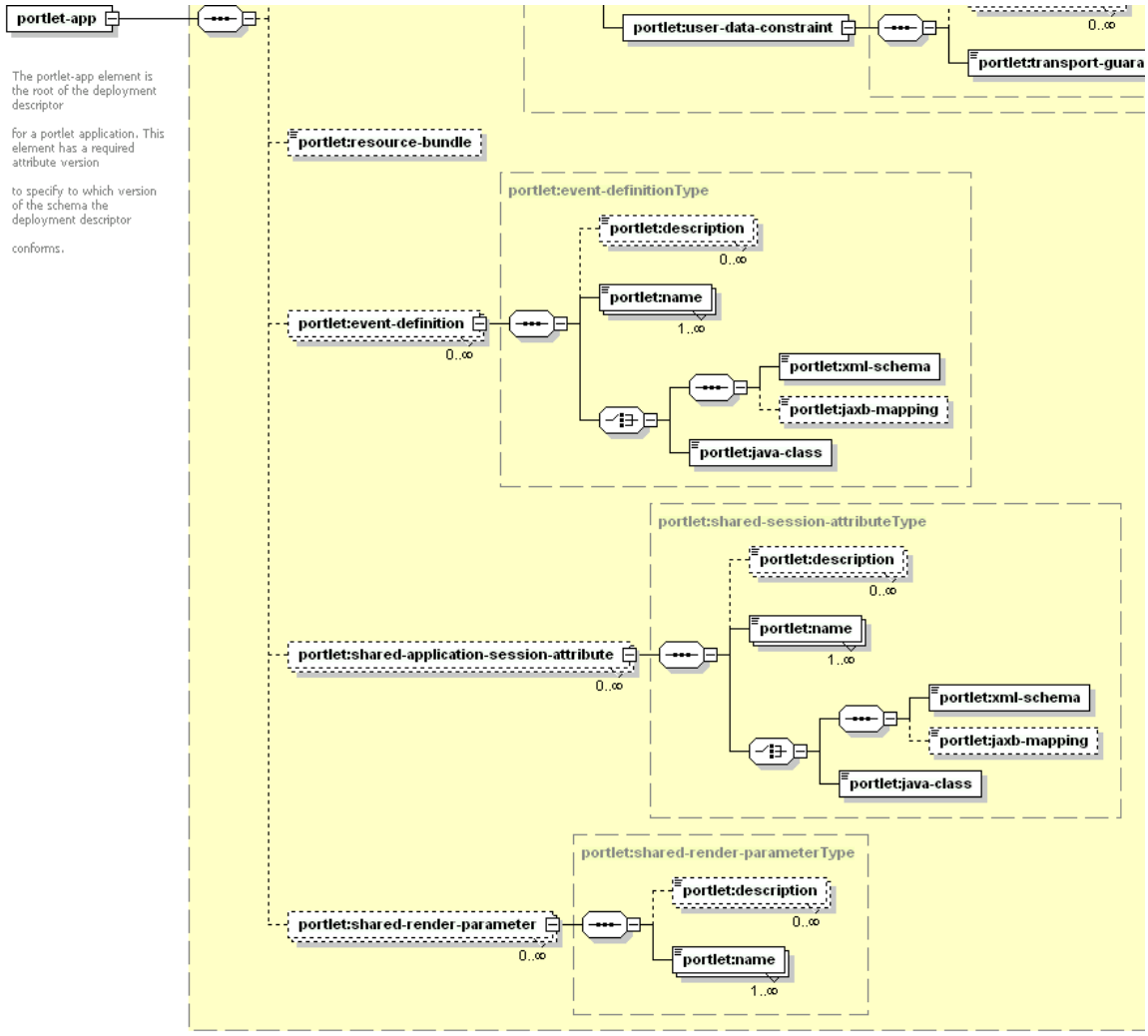
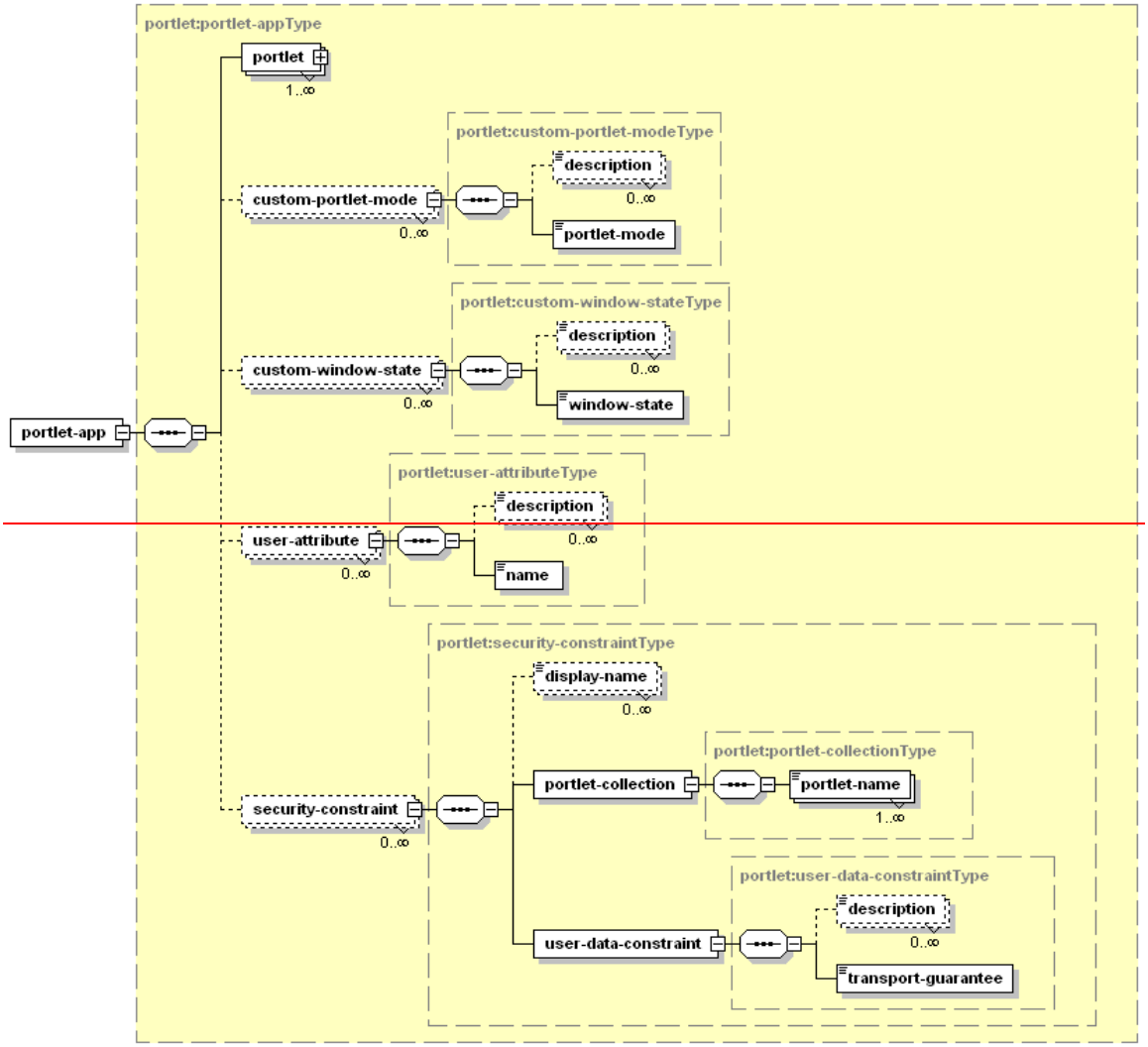
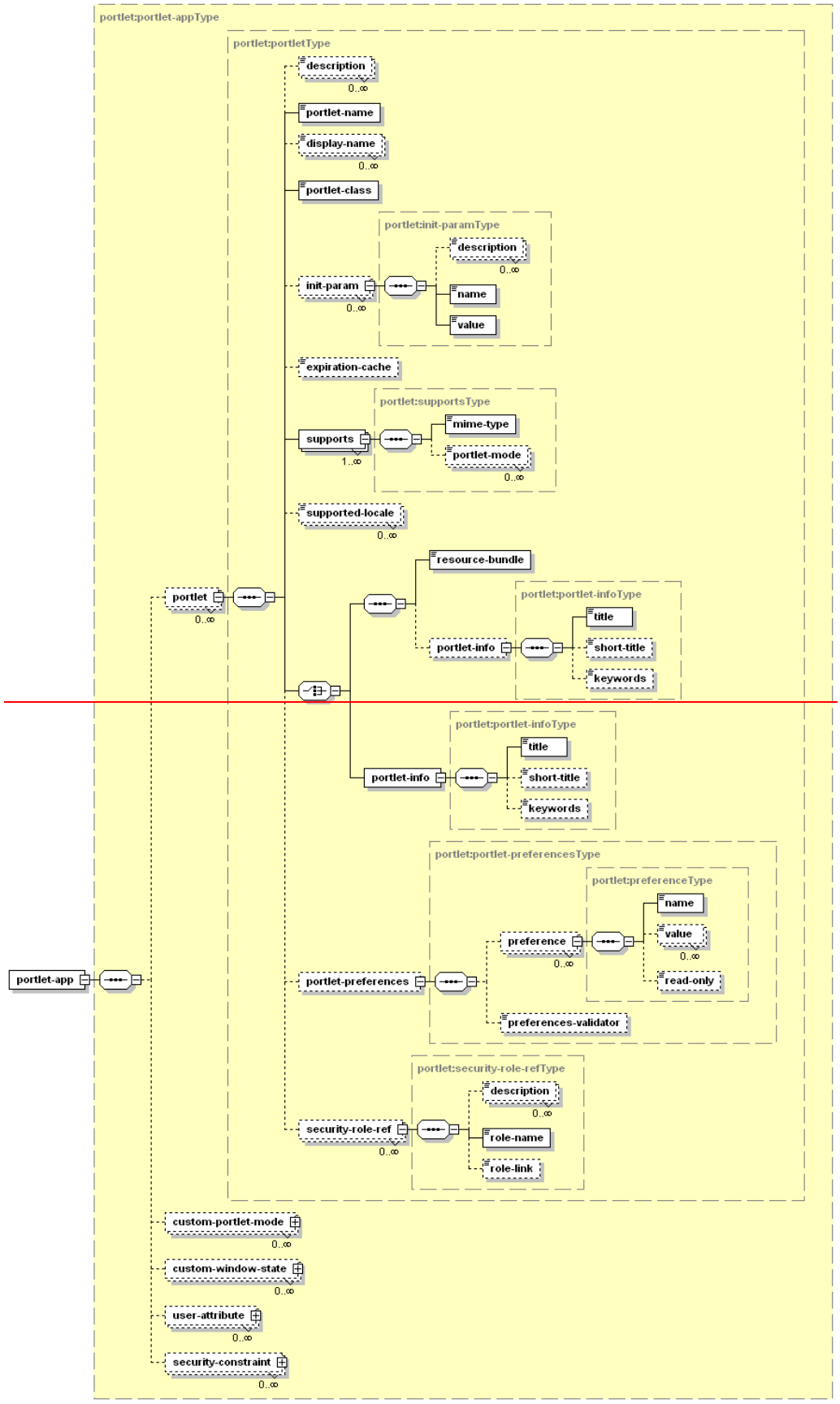


Figure 4: Part 2 of the portlet-app element

~~INSERT PICTURE HERE~~

~~PLT.23.6Pictures of the structure of a Deployment Descriptor~~




portlet:portlet-appType
portlet:portletType
description
0..∞
portlet-name
display-name
0..∞
portlet-class
portlet:init-paramType
description
0..∞
init-param
0..∞
name
value
expiration-cache
portlet:supportsType
supports
1..∞
mime-type
portlet-mode
0..∞
supported-locale
0..∞
resource-bundle
portlet
0..∞
portlet:portlet-infoType
portlet-info
title
short-title
keywords
portlet:portlet-infoType
portlet-info
title
short-title
keywords
portlet:portlet-preferencesType
portlet:preferenceType
preference
0..∞
name
value
0..∞
read-only
portlet-app
portlet-preferences
preferences-validator
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.1.7~~PLT.24.7 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>`
- event-definition `<name>`
- shared-application-session-attribute `<name>`
- shared-render-parameter `<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>`
- shared-portlet-session-attribute `<name>`

~~PLT.1.8~~PLT.24.8 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.1.8.1~~PLT.24.8.1 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 `<display-name>`

The default value for the `xml:lang` attribute is English (`"en"`). Portlet-container implementations using localized values of these elements should treat the English (`"en"`) values as the default fallback value for all other locales.

As an alternative to embedding all localized values in the deployment descriptor the portlet can provide a resource bundle via the `<resource-bundle>` element on the portlet application level (see Resource Bundle section below).

~~PLT.1.8.2~~PLT.24.8.2 Locales Supported 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.

The supported locales declared in the deployment descriptor should follow the lang_COUNTRY_variant format as defined by RFC 1766 (http://www.faqs.org/rfcs/rfc1766.html).

~~PLT.1.9~~PLT.24.9 Deployment Descriptor Example

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
version="1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app 1 0.xsd
                       http://java.sun.com/xml/ns/portlet/portlet-app_1_0.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="de">ZeitzonenPortlet</display-name>
    <portlet-class>com.myco.samplets.util.zoneclock.ZoneClock</portlet-class>
    <expiration-cache>60</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>
      <title>Time Zone Clock</title>
      <short-title>TimeZone</short-title>
      <keywords>Time, Zone, World, Clock</keywords>
    </portlet-info>
    <portlet-preferences>
      <preference>
        <name>time-server</name>
        <value>http://timeserver.myco.com</value>
        <read-only>true</read-only>
      </preference>
      <preference>
        <name>port</name>
        <value>404</value>
        <read-only>true</read-only>
      </preference>
      <preference>
        <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.1.10~~PLT.24.10 Resource Bundles

As an alternative to embed all localized values in the deployment descriptor the portlet can provide a separate resource bundle containing the localized values. For language specific portlet application level] information the fully qualified class name of the resource bundle can be set in the deployment descriptor using the `resource-bundle` element on the portlet application level. ~~The fully qualified class name of the resource bundle can be set in the portlet definition in the deployment descriptor using the `resource-bundle` tag on the portlet application level.~~ The Java Portlet Specification defines the following constants for the application level resource bundle:

javax.portlet.app.custom-portlet-mode.<portlet-mode>.description	Description of custom portlet mode <portlet-mode>.
javax.portlet.app.custom-window-state.<window-state>.description	Description of the custom window state <window-state>.
javax.portlet.app.user-attribute.<name>.description	Description of the user attribute <name>.
javax.portlet.app.event-definition.<name>.description	Description of the event <name>.
javax.portlet.app.event-definition.<name>.display-name	Name under which this event is displayed to users or to tools. The display name need not be unique.
javax.portlet.app.shared-application-session-attribute.<name>.description	Description of the shared application scope session attribute <name>.
javax.portlet.app.shared-~~portlet~~application-session-	Name under which this shared application session attribute is displayed to users or to tools. The display

attribute.<name>.display-name	name need not be unique.
javax.portlet.app.shared-render-parameter.<name>.description	Description of the shared render parameter <name>.
javax.portlet.app.shared-render-parameter.<name>.display-name	Name under which this shared render parameter is displayed to users or to tools. The display name need not be unique.

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

The Java Portlet Specification ~~1.0~~ defines the following constants for ~~this~~ the portlet level resource bundle:

javax.portlet.title	The title that should be displayed in the titlebar of this portlet. Only one title per locale is allowed. Note that this title may be overrided by the portal or programmatically by the portlet.
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 ‘,’.
javax.portlet.description	Description of the portlet.
javax.portlet.display-name	Name under which this portlet is displayed at deployment time or to tools. The display name need not be unique.
javax.portlet.shared-portlet-session-attribute.<name>.description	Description of the shared portlet scope session attribute <name>.
javax.portlet. shared-portlet-session-attribute.<name>.display-name	Name under which this shared portlet session attribute is displayed to users or to tools. The display name need not be unique.

~~PLT.1.11~~PLT.24.11 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=Weltbevölkerungsuhr
javax.portlet.short-title=Weltuhr
javax.portlet.keywords=Welt,Bevölkerung,Uhr
```

~~PLT.24~~PLT.25

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` or `ResourceRequest` and `RenderResponse`. It also provides JSPs with access to portlet functionality such as creation of portlet URLs.

The portlet-container must provide an implementation of the portlet tag library.[ccxxv] Portlet developers may indicate an alternate implementation using the mechanism defined in the *JSP.7.3.9 Well-Know URIs* Section of the *JSP Specification* ~~*1.2*~~.

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.24.1~~PLT.25.1 defineObjects Tag

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

- RenderRequest renderRequest when included from within the `render` method or `ResourceRequest resourceRequest` when included from within the `serveResource` method
- 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.~~16~~18.3.1 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 contain any body content.[ccxxvii]

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.24.2~~PLT.25.2 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.[ccxxviii]

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:

- **windowState** (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 the specified window state is illegal for the current request, a JspException must be thrown.[ccxxix] Reasons for a window state being illegal may include that the portal does not support this state, the portlet has not declared in its deployment descriptor that it supports this state, or the current user is not allowed to switch to this state. If a window state is not set for a URL, it should stay the same as the window state of the current request.[ccxxx] The window state attribute is not case sensitive.
- **portletMode** (Type: String, non-required) – indicates the portlet mode that the portlet must have when this link is executed, if no error condition ocurred.[ccxxxi] The following portlet modes are predefined: `edit`, `help`, and `view`. If the specified portlet mode is illegal for the current request, a JspException must be thrown. [ccxxxii]Reasons for a portlet mode being illegal may include that the portal does not support this mode, the portlet has not declared in its deployment descriptor that it supports this mode for the current markup, or the current user is not allowed to switch to this mode. If a portlet mode is not set for a URL, it must stay the same as the mode of the current request. [ccxxxiii]The portlet mode attribute is not case sensitive.
- **var** (Type: String, non-required) – name of the exported scoped variable for the action URL. The exported scoped variable must be a `String`. 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`.[ccxxxiv]

  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. If the given variable name already exists in the scope of the page or it is used within an iteration loop, the new value overwrites the old one.[ccxxxv]
- **secure** (Type: String, non-required) – indicates if the resulting URL should be a secure connection (secure="true") or an insecure one (secure="false"). If the specified security setting is not supported by the run-time environment, a JspException must be thrown.[ccxxxvi] If the security is not set for a URL, it must stay the same as the security setting of the current request.

A `JspException` with the `PortletException` that caused this error as root cause is thrown in the following cases:

- If an illegal window state is specified in the `windowState` attribute.

- If an illegal portlet mode is specified in the `portletMode` attribute.
- If an illegal security setting is specified in the `secure` attribute.

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

```
<portlet:actionURL windowState="maximized" portletMode="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.24.3~~PLT.25.3 renderURL Tag

The portlet `renderURL` tag creates a URL that must point to the current portlet and must trigger a render request with the supplied parameters.[ccxxxvii]

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:

- **windowState** (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 the specified window state is illegal for the current request, a JspException must be thrown.[ccxxxviii] Reasons for a window state being illegal may include that the portal does not support this state, the portlet has not declared in its deployment descriptor that it supports this state, or the current user is not allowed to switch to this state. If a window state is not set for a URL, it should stay the same as the window state of the current request.[ccxxxix] The window state attribute is not case sensitive.
- **portletMode** (Type: String, non-required) – indicates the portlet mode that the portlet must have when this link is executed, if not error condition ocurred.[ccxl] The following portlet modes are predefined: `edit`, `help`, and `view`. If the specified portlet mode is illegal for the current request, a JspException must be thrown.[ccxli] Reasons for a portlet mode being illegal may include that the portal does not support this mode, the portlet has not declared in its deployment descriptor that it supports this mode for the current markup, or the current user is not allowed to switch to this mode. If a portlet mode is not set for a URL, it must stay the same as the mode of the current request.[ccxlii] The portlet mode attribute is not case sensitive.
- **var** (Type: String, non-required) – name of the exported scoped variable for the render URL. The exported scoped variable must be a `String`. 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`.[ccxliii]

  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. If the given variable name already exists in the scope of the page or it is used within an iteration loop, the new value overwrites the old one.[ccxliv]

- **secure** (Type: String, non-required) – indicates if the resulting URL should be a secure connection (secure="true") or an insecure one (secure="false"). If the specified security setting is not supported by the run-time environment, a JspException must be thrown. If the security is not set for a URL, it must stay the same as the security setting of the current request.[ccxlv]

A `JspException` with the `PortletException` that caused this error as root cause is thrown in the following cases:

- If an illegal window state is specified in the `windowState` attribute.
- If an illegal portlet mode is specified in the `portletMode` attribute.
- If an illegal security setting is specified in the `secure` attribute.

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

```
<portlet:renderURL portletMode="view" windowState="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.25.4 resourceURL Tag

The portlet `renderURL` tag creates a URL that must point to the current portlet and must trigger a `serveResource` request with the supplied parameters.[ccxlvi]

The `resourceURL` must contain the current portlet mode, window state and render parameters.[ccxlvii]

Parameters may be added by including the `param` tag between the `resourceURL` start and end tags. If such a parameter has the same name as a render parameter in this URL, the render parameter value must be the last value in the attribute value array.[ccxlviii]

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

- **var** (Type: String, non-required) – name of the exported scoped variable for the render URL. The exported scoped variable must be a `String`. 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`.[ccxlix]

  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. If the given variable name already exists in the scope of the page or it is used within an iteration loop, the new value overwrites the old one.[ccl]

- **secure** (Type: String, non-required) – indicates if the resulting URL should be a secure connection (secure="true") or an insecure one (secure="false"). If the specified security setting is not supported by the run-time environment, a

JspException must be thrown. If the security is not set for a URL, it must stay the same as the security setting of the current request.[ccli]

A `JspException` with the `PortletException` that caused this error as root cause is thrown in the following case:

- If an illegal security setting is specified in the `secure` attribute.

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

```
<portlet:resoureURL>
   <portlet:param name="icon1" value="mypict.gif"/>
</portlet:renderURL>
```

The example creates a URL to provide a link that renders the icon named `mypict.gif`.

~~PLT.24.4~~PLT.25.5 namespace Tag

This tag produces a unique value for the current portlet and must match the value of `PortletResponse.getNamespace` method. [cclii]

This tag should be used for named elements in the portlet output (such as Javascript functions and variables). The namespacing 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 `namespace` tag must not allow any body content.

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

```
<A HREF="javascript:<portlet:namespace/>doFoo()">Foo</A>
```

The example prefixes a JavaScript function with the name 'doFoo', ensuring uniqueness on the portal page.

~~PLT.24.5~~PLT.25.6 param Tag

This tag defines a parameter that may be added to an `actionURL`, ~~or~~ `renderURL` or `resourceURL`.[ccliii]

The `param` tag must not contain any body content.[ccliv]

If the same name of a parameter occurs more than once within a an `actionURL`, `renderURL` or `resourceURL` the values must be delivered as parameter value array with the values in the order of the declaration within the URL tag. [cclv]

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.25~~

Leveraging JAXB for Event and Shared Session payloads

PLT.26

The Java Portlet Specification 2.0 leverages the Java Architecture for

XML Binding (JAXB) 2.0 for defining payload data that may be transported across the network via remote protocols such as Web Services for Remote Portlets (WSRP) 2.0 specification. These are the event payload and the shared session attribute value.

The event payload and the shared session attribute value must be defined ~~by either the following alternatives~~[cclvi]~~:~~

using the JAXB annotations in the Java object and defining the Java object class name in the deployment descript via the `java-class` element. Defining the Java object class name in the deployment descriptor is optional for publishing events and mandatory for consuming events.[cclvii]

~~providing an XML schema in the deployment descriptor via the `xml-schema` element and optionally a JAXB mapping via the `jaxb-mapping` element~~

~~PLT.25~~PLT.27

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.25.1~~PLT.27.1 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.25.2~~PLT.27.2 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.25.2.1~~PLT.27.2.1 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.25.2.1.1~~PLT.27.2.1.1 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/portlet/portletTCK_1_0.xsd"
xmlns:pct="http://java.sun.com/xml/ns/portlet/portletTCK_1_0.xsd"
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.25.2.1.2~~PLT.27.2.1.2 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/portlet/portletTCKVendor_1_0.xsd"
xmlns:pct="http://java.sun.com/xml/ns/portlet/portletTCKVendor_1_0.xsd"
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.25.2.2~~PLT.27.2.2 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.25.2.3~~PLT.27.2.3 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.25.2.4~~PLT.27.2.4 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~~portlet-mode>about</~~name~~portlet-mode>
  </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~~portlet-mode> preview </~~name~~portlet-mode>
  </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~~portlet-mode>print</~~name~~portlet-mode>
  </custom-portlet-mode>
  ...
  </portlet-app>
```

~~PLT.APLT.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`, ~~`iframe`,~~ `frame`, `frameset`, `head`, `html` and `title`. ~~The `iframe` tag can be used, however it must be used with caution. The usage of the `iframe` tag should not break the portal paradigm.~~Using the `iframe` tag is not forbidden, but portlets using `iframes` should not expect portal/portlet context for the content of `iframes`

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

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

5 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.

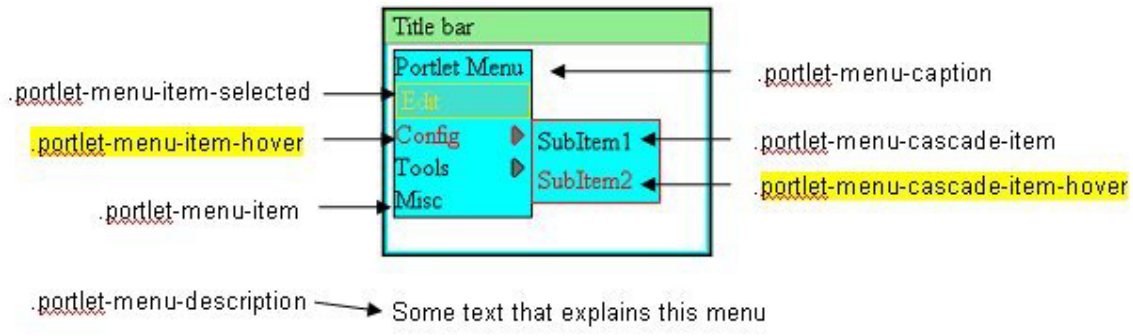
Style	Description
portlet-form-label	Text used for the descriptive label of the whole form (not the labels for fields.
portlet-form-input-field	Text of the user-input in an input field.
portlet-form-button	Text on a button
portlet-icon-label	Text that appears beside a context dependent action icon.
portlet-dlg-icon-label	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)
portlet-form-field-label	Text that appears beside a form field (e.g. input fields, checkboxes, etc.)
portlet-form-field	Text for a field which is 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-cascade	General sub-menu settings such as background color, margins, etc
portlet-menu-cascade-item	A normal, unselected sub-menu item
portlet-menu-cascade-item-selected	Selected sub-menu item
portlet-menu-cascade-item-hover	Normal, unselected sub-menu item when the mouse hovers over it
portlet-menu-cascade-item-hover-selected	Selected sub-menu item when the mouse hovers over it
portlet-menu-separator	Separator between menu items
portlet-menu-cascade-separator	Separator between sub-menu items
portlet-menu-content	Content for a normal, unselected menu or sub-menu item
portlet-menu-content-selected	Content for an selected menu or sub-menu item
portlet-menu-content-hover	Content for an unselected menu or sub-menu item when the mouse hovers over it
portlet-menu-content-hover-selected	Content for a selected menu or sub-menu item when the mouse hovers over it
portlet-menu-indicator	Indicator that a menu item has an associated sub-menu

portlet-menu-indicator-selected	Indicator when the associated menu item is selected
portlet-menu-indicator-hover	Indicator when the associated menu item has the mouse hover over it
portlet-menu-indicator-hover-selected	Indicator when the associated menu item is selected and has the mouse hover over it
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). 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
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
user.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
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
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
user.business-info.telecom.pager.comment
user.business-info.online.email
user.business-info.online.uri

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>
```

~~FutureReleases.doc~~

PLT.D.2

Deployment Descriptor Version 1.0

This appendix defines the deployment descriptor for version 1.0. All portlet containers are required to support portlet applications using the 1.0 deployment descriptor.

PLT.D.2.1 Deployment Descriptor of Version 1.0

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:portlet="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
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>
  <!-- ********************************************************** -->
  <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. This element has a required attribute version
      to specify to which version of the schema the deployment descriptor
      conforms.
      </documentation>
    </annotation>
    <unique name="portlet-name-uniqueness">
      <annotation>
        <documentation>
        The portlet element contains the name of a portlet.
        This name must be unique within the portlet application.
         </documentation>
      </annotation>
      <selector xpath="portlet:portlet"/>
      <field xpath="portlet:portlet-name"/>
    </unique>
    <unique name="custom-portlet-mode-uniqueness">
      <annotation>
        <documentation>
        The custom-portlet-mode element contains the portlet-mode.
        This portlet mode must be unique within the portlet application.
        </documentation>
      </annotation>
      <selector xpath="portlet:custom-portlet-mode"/>
      <field xpath="portlet:portlet-mode"/>
```

```
    </unique>
    <unique name="custom-window-state-uniqueness">
      <annotation>
        <documentation>
        The custom-window-state element contains the window-state.
        This window state must be unique within the portlet application.
        </documentation>
      </annotation>
      <selector xpath="portlet:custom-window-state"/>
      <field xpath="portlet:window-state"/>
    </unique>
    <unique name="user-attribute-name-uniqueness">
      <annotation>
        <documentation>
        The user-attribute element contains the name the attribute.
        This name must be unique within the portlet application.
        </documentation>
      </annotation>
      <selector xpath="portlet:user-attribute"/>
      <field xpath="portlet:name"/>
    </unique>
  </element>
  <complexType name="portlet-appType">
    <sequence>
      <element name="portlet" type="portlet:portletType" minOccurs="0"
maxOccurs="unbounded">
        <unique name="init-param-name-uniqueness">
          <annotation>
            <documentation>
            The init-param element contains the name the attribute.
            This name must be unique within the portlet.
            </documentation>
          </annotation>
          <selector xpath="portlet:init-param"/>
          <field xpath="portlet:name"/>
        </unique>
        <unique name="supports-mime-type-uniqueness">
          <annotation>
            <documentation>
            The supports element contains the supported mime-type.
            This mime type must be unique within the portlet.
            </documentation>
          </annotation>
          <selector xpath="portlet:supports"/>
          <field xpath="mime-type"/>
        </unique>
        <unique name="preference-name-uniqueness">
          <annotation>
            <documentation>
            The preference element contains the name the preference.
            This name must be unique within the portlet.
            </documentation>
          </annotation>
          <selector xpath="portlet:portlet-preferences/portlet:preference"/>
          <field xpath="portlet:name"/>
        </unique>
        <unique name="security-role-ref-name-uniqueness">
          <annotation>
            <documentation>
            The security-role-ref element contains the role-name.
            This role name must be unique within the portlet.
            </documentation>
          </annotation>
          <selector xpath="portlet:security-role-ref"/>
          <field xpath="portlet:role-name"/>
        </unique>
      </element>
      <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="int"/>
    </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>
  <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="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="init-param" type="portlet:init-paramType" minOccurs="0"
maxOccurs="unbounded"/>
      <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"/>
      <choice>
        <sequence>
          <element name="resource-bundle" type="portlet:resource-bundleType"/>
          <element name="portlet-info" type="portlet:portlet-infoType"
minOccurs="0"/>
        </sequence>
        <element name="portlet-info" type="portlet:portlet-infoType"/>
      </choice>
      <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>
  <simpleType name="portlet-classType">
    <annotation>
      <documentation>
```

```
      The portlet-class element contains the fully
      qualified class name of the portlet.
     Used in: portlet
     </documentation>
    </annotation>
    <restriction base="portlet:fully-qualified-classType"/>
  </simpleType>
  <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">
    <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>
    <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: portlet-preferences
      </documentation>
    </annotation>
    <sequence>
      <element name="name" type="portlet:nameType"/>
      <element name="value" type="portlet:valueType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="read-only" type="portlet:read-onlyType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="string" use="optional"/>
  </complexType>
  <simpleType name="preferences-validatorType">
    <annotation>
      <documentation>
      The class specified under preferences-validator implements
      the PreferencesValidator interface to validate the
      preferences settings.
      Used in: portlet-preferences
      </documentation>
    </annotation>
    <restriction base="portlet:fully-qualified-classType"/>
  </simpleType>
  <simpleType name="read-onlyType">
    <annotation>
      <documentation>
      read-only indicates that a setting cannot
      be changed in any of the standard portlet modes
      ("view","edit" or "help").
      Per default all preferences are modifiable.
      Valid values are:
      - true for read-only
      - false for modifiable
      Used in: preferences
      </documentation>
    </annotation>
    <restriction base="portlet:string">
      <enumeration value="true"/>
      <enumeration value="false"/>
    </restriction>
  </simpleType>
  <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"/>
      <element name="user-data-constraint" type="portlet:user-data-
constraintType"/>
    </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 code of the web application. 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 a
      portlet supports for a specific content type. All portlets must
      support the view mode.
      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. 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.xsd -->
  <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. It has an optional attribute xml:lang to indicate
      which language is used in the description according to
      RFC 1766 (http://www.ietf.org/rfc/rfc1766.txt). The default
      value of this attribute is English("en").
      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>

      It has an optional attribute xml:lang to indicate
      which language is used in the description according to
      RFC 1766 (http://www.ietf.org/rfc/rfc1766.txt). The default
      value of this attribute is English("en").
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet:string">
        <attribute ref="xml:lang"/>
      </extension>
    </simpleContent>
```

```
  </complexType>
  <simpleType name="fully-qualified-classType">
    <annotation>
      <documentation>
      The elements that use this type designate the name of a
      Java class or interface.
      </documentation>
    </annotation>
    <restriction base="portlet:string"/>
  </simpleType>
  <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.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:68	Testable=true	Section=PLT.12.2.2
[ii] SPEC:68	Testable=true	Section=PLT.12.2.2
[iii] SPEC:68	Testable=true	Section=PLT.12.2.2
[iv] SPEC:68	Testable=true	Section=PLT.12.2.2
[v] SPEC:1	Testable=false	Section=PLT.5.1
[vi] SPEC:2	Testable=false	Section=PLT.5.1
[vii] SPEC:3	Testable=false	Section=PLT.5.2.1
[viii] SPEC:4	Testable=true	Section=PLT.5.2.2
[ix] SPEC:5	Testable=true	Section=PLT.5.2.2.1
[x] SPEC:6	Testable=true	Section=PLT.5.2.2.1
[xi] SPEC:7	Testable=true	Section=PLT.5.2.2.1
[xii] SPEC:8	Testable=true	Section=PLT.5.2.2.1
[xiii] SPEC:20	Testable=false	Section=PLT/5.2.5
[xiv] SPEC:21	Testable= false	Section=PLT.5.2.5
[xv] SPEC:22	Testable=false	Section=PLT.5.2.5
[xvi] SPEC:23	Testable= false	Section=PLT.5.2.5
[xvii] SPEC:9	Testable=true	Section=PLT 5.2.4
[xviii] SPEC:10	Testable=true	Section=PLT 5.2.4

[xix] SPEC:11	Testable=true	Section=PLT 5.2.4.1
[xx] SPEC:11	Testable=true	Section=PLT 5.2.4.1
[xxi] SPEC:12	Testable= true	Section=PLT.5.2.4.1
[xxii] SPEC:13	Testable= true	Section=PLT.5.2.4.2.1
[xxiii] SPEC:14	Testable= true	Section=PLT.5.2.4.2.1
[xxiv] SPEC:15	Testable= true	Section=PLT.5.2.4.2.1
[xxv] SPEC:16	Testable=true	Section=PLT 5.2.4.2.1
[xxvi] SPEC:17	Testable= true	Section=PLT.5.2.4.4
[xxvii] SPEC:18	Testable=false	Section=PLT.5.2.4.4
[xxviii] SPEC:19	Testable= true	Section=PLT.5.2.4.4.
[xxix] SPEC:20	Testable=false	Section=PLT/5.2.5
[xxx] SPEC:21	Testable= false	Section=PLT.5.2.5
[xxxi] SPEC:22	Testable=false	Section=PLT.5.2.5
[xxxii] SPEC:23	Testable= false	Section=PLT.5.2.5
[xxxiii] SPEC:24	Testable= true	Section=PLT.6.2
[xxxiv] SPEC:25	Testable= true	Section=PLT.6.2
[xxxv] SPEC:26	Testable= true	Section=PLT.7.1
[xxxvi] SPEC:27	Testable= true	Section=PLT.7.1
[xxxvii] SPEC:28	Testable= true	Section=PLT.7.1
[xxxviii] SPEC:29	Testable= true	Section=PLT.7.1
[xxxix] SPEC:30	Testable= true	Section=PLT.7.1
[xl] SPEC:31	Testable= true	Section=PLT.7.1
[xli] SPEC:31	Testable= true	Section=PLT.7.1
[xlii] SPEC:32	Testable= true	Section=PLT.7.1.1
[xliii] SPEC:33	Testable= true	Section=PLT.7.1.1
[xliv] SPEC:33	Testable= true	Section=PLT.7.1.1
[xlv] SPEC:34	Testable= true	Section=PLT.7.1.1
[xlvi] SPEC:33	Testable= true	Section=PLT.7.1.1
[xlvii] SPEC:35	Testable= true	Section=PLT.6.2

[xlviii] SPEC:36	Testable=true	Section=PLT.8.5
[xlix] SPEC:37	Testable=true	Section=PLT.8.6
[l] SPEC:38	Testable=true	Section=PLT.8.6
[li] SPEC:39	Testable=false	Section=PLT.8.6
[lii] SPEC:40	Testable=true	Section=PLT.9.4
[liii] SPEC:41	Testable=false	Section=PLT.10.1
[liv] SPEC:42	Testable=false	Section=PLT.10.1
[lv] SPEC:43	Testable=true	Section=PLT.10.3
[lvi] SPEC:44	Testable=true	Section=PLT.10.3
[lvii] SPEC:45	Testable=true	Section=PLT.10.3
[lviii] SPEC:46	Testable=true	Section=PLT.10.3
[lix] SPEC:47	Testable=true	Section=PLT.10.3(servlet spec)
[lx] SPEC:48	Testable=true	Section=PLT.11.1.1
[lxi] SPEC:49	Testable= true	Section=PLT.11.1.1
[lxii] SPEC:55	Testable=true	Section=PLT.11.1.1
[lxiii] SPEC:56	Testable=true	Section=PLT.11.1.1
[lxiv] SPEC:56	Testable=true	Section=PLT.11.1.1
[lxv] SPEC:56	Testable=true	Section=PLT.11.1.1
[lxvi] SPEC:50	Testable= true	Section=PLT.11.1.1
[lxvii] SPEC:51	Testable=true	Section=PLT.11.1.1
[lxviii] SPEC:52	Testable=true	Section=PLT.11.1.1
[lxix] SPEC:52	Testable=true	Section=PLT.11.1.1
[lxx] SPEC:53	Testable= true	Section=PLT.11.1.1
[lxxi] SPEC:54	Testable=true	Section=PLT.11.1.1
[lxxii] SPEC:55	Testable=true	Section=PLT.11.1.1
[lxxiii] SPEC:56	Testable=true	Section=PLT.11.1.1
[lxxiv] SPEC:53	Testable= true	Section=PLT.11.1.1
[lxxv] SPEC:53	Testable= true	Section=PLT.11.1.1
[lxxvi] SPEC:53	Testable= true	Section=PLT.11.1.1

lxxvii SPEC:53	Testable= true	Section=PLT.11.1.1
lxxviii SPEC:53	Testable= true	Section=PLT.11.1.1
lxxix SPEC:53	Testable= true	Section=PLT.11.1.1
lxxx SPEC:57	Testable=false	Section=PLT.11.1.2
lxxxi SPEC:58	Testable= true	Section=PLT.11.1.5
lxxxii SPEC:59	Testable=true	Section=PLT.11.1.5
lxxxiii SPEC:60	Testable=true	Section=PLT.11.1.6
lxxxiv SPEC:61	Testable=true	Section=PLT.11.1.7
lxxxv SPEC:62	Testable=true	Section=PLT.11.1.7
lxxxvi SPEC:63	Testable=true	Section=PLT.11.2.1
lxxxvii SPEC:64	Testable=true	Section=PLT.11.2.1
lxxxviii SPEC:73	Testable=true	Section=PLT.12.2.3
lxxxix SPEC:86	Testable= true	Section=PLT.12.3.4
xc SPEC:87	Testable= true	Section=PLT.12.3.4
xci SPEC:88	Testable=true	Section=PLT.12.3.4
xcii SPEC:72	Testable= true	Section=PLT.12.2.3
xciii SPEC:67	Testable=true	Section=PLT.12.2.2
xciv SPEC:68	Testable=true	Section=PLT.12.2.2
xcv SPEC:68	Testable=true	Section=PLT.12.2.2
xcvi SPEC:65	Testable=true	Section=PLT.12.2.1
xcvii SPEC:66	Testable=true	Section=PLT.12.2.1
xcviii SPEC:67	Testable=true	Section=PLT.12.2.2
xcix SPEC:68	Testable=true	Section=PLT.12.2.2
c SPEC:69	Testable= true	Section=PLT.12.2.2
ci SPEC:70	Testable= true	Section=PLT.12.2.2
cii SPEC:71	Testable=true	Section=PLT.12.2.2
ciii SPEC:72	Testable= true	Section=PLT.12.2.3
civ SPEC:73	Testable=true	Section=PLT.12.2.3
cv SPEC:74	Testable= true	Section=PLT.12.2.3

[cvi] SPEC:75	Testable= true	Section=PLT.12.2.3
[cvii] SPEC:76	Testable=true	Section=PLT.12.3.1
[cviii] SPEC:77	Testable= true	Section=PLT.12.3.1
[cix] SPEC:78	Testable= true	Section=PLT.12.3.1
[cx] SPEC:79	Testable= true	Section=PLT.12.3.2
[cxi] SPEC:80	Testable=true	Section=PLT.12.3.3
[cxii] SPEC:81	Testable=true	Section=PLT.12.3.3
[cxiii] SPEC:82	Testable=true	Section=PLT.12.3.3
[cxiv] SPEC:83	Testable=true	Section=PLT.12.3.3
[cxv] SPEC:84	Testable=true	Section=PLT.12.3.3
[cxvi] SPEC:85	Testable=true	Section=PLT.12.3.3
[cxvii] SPEC:86	Testable= true	Section=PLT.12.3.4
[cxviii] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxix] SPEC:88	Testable=true	Section=PLT.12.3.4
[cxx] SPEC:89	Testable=false	Section=PLT.12.3.5
[cxxi] SPEC:68	Testable=true	Section=PLT.12.2.2
[cxxii] SPEC:68	Testable=true	Section=PLT.12.2.2
[cxxiii] SPEC:68	Testable=true	Section=PLT.12.2.2
[cxxiv] SPEC:68	Testable=true	Section=PLT.12.2.2
[cxxv] SPEC:68	Testable=true	Section=PLT.12.2.2
[cxxvi] SPEC:68	Testable=true	Section=PLT.12.2.2
[cxxvii] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxviii] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxix] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxx] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxxi] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxxii] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxxiii] SPEC:87	Testable= true	Section=PLT.12.3.4
[cxxxiv] SPEC:90	Testable= true	Section=PLT.14.1

cxxxv SPEC:91	Testable= true	Section=PLT.14.1
cxxxvi SPEC:92	Testable=true	Section=PLT.14.1
cxxxvii SPEC:93	Testable=true	Section=PLT.14.1
cxxxviii SPEC:94	Testable=true	Section=PLT.14.1
cxxxix SPEC:95	Testable=true	Section=PLT.14.1
cxl SPEC:96	Testable=true	Section=PLT.14.1
cxli SPEC:97	Testable= true	Section=PLT.14.1(change)
cxlii SPEC:98	Testable=true	Section=PLT.14.1
cxliii SPEC:99	Testable=true	Section=PLT.14.3
cxliv SPEC:100	Testable=true	Section=PLT.14.3
cxlv SPEC:101	Testable=false	Section=PLT.14.4
cxlvi SPEC:102	Testable=false	Section=PLT.14.4
cxlvii SPEC:103	Testable=true	Section=PLT.14.4
cxlviii SPEC:104	Testable=true	Section=PLT.14.4
cxlix SPEC:105	Testable=true	Section=PLT.14.4
cl SPEC:106	Testable=true	Section=PLT.15.1
cli SPEC:107	Testable=true	Section=PLT.15.1
clii SPEC:108	Testable=true	Section=PLT.15.2
cliii SPEC:109	Testable=true	Section=PLT.15.2
cliv SPEC:110	Testable=true	Section=PLT.15.3
clv SPEC:111	Testable=true	Section=PLT.15.3
clvi SPEC:112	Testable=true	Section=PLT.15.3
clvii SPEC:113	Testable=true	Section=PLT.15.4
clviii SPEC:114	Testable=true	Section=PLT.15.4
clix SPEC:115	Testable=true	Section=PLT.15.4
clx SPEC:116	Testable=true	Section=PLT.15.4
clxi SPEC:117	Testable=true	Section=PLT.15.4.1
clxii SPEC:118	Testable=true	Section=PLT.15.4.1
clxiii SPEC:119	Testable=true	Section=PLT.15.4.1

clxiv SPEC:119	Testable=true	Section=PLT.15.4.1
clxv SPEC:120	Testable=true	Section=PLT.15.8(servlet spec)
clxvi SPEC:121	Testable=true	Section=PLT.16.1
clxvii SPEC:122	Testable=true	Section=PLT.16.1
clxviii SPEC:123	Testable= true	Section=PLT.16.1.1
clxix SPEC:124	Testable=true	Section=PLT.16.2
clxx SPEC:124	Testable=true	Section=PLT.16.2
clxxi SPEC:125	Testable=true	Section=PLT.16.2
clxxii SPEC:126	Testable=true	Section=PLT.16.3
clxxiii SPEC:127	Testable=true	Section=PLT.16.3.1
clxxiv SPEC:128	Testable=true	Section=PLT.16.3.2
clxxv SPEC:129	Testable=true	Section=PLT.16.3.3
clxxvi SPEC:130	Testable=true	Section=PLT.16.3.3
clxxvii SPEC:131	Testable=true	Section= PLT.16.3.3
clxxviii SPEC:132	Testable=true	Section=PLT.16.3.3
clxxix SPEC:133	Testable=true	Section=PLT.16.3.3
clxxx SPEC:134	Testable=true	Section=PLT.16.3.3
clxxxi SPEC:135	Testable=true	Section= PLT.16.3.3
clxxxii SPEC:136	Testable=true	Section= PLT.16.3.3
clxxxiii SPEC:137	Testable=true	Section= PLT.16.3.3
clxxxiv SPEC:138	Testable=true	Section= PLT.16.3.3
clxxxv SPEC:139	Testable=true	Section= PLT.16.3.3
clxxxvi SPEC:140	Testable=false(impl)	Section= PLT.16.3.3
clxxxvii SPEC:141	Testable=true	Section= PLT.16.3.3
clxxxviii SPEC:129	Testable=true	Section=PLT.16.3.3
clxxxix SPEC:130	Testable=true	Section=PLT.16.3.3
cxc SPEC:131	Testable=true	Section= PLT.16.3.3
cxci SPEC:131	Testable=true	Section= PLT.16.3.3
cxcii SPEC:132	Testable=true	Section=PLT.16.3.3

[cxciii] SPEC:135	Testable=true	Section= PLT.16.3.3
[cxciv] SPEC:136	Testable=true	Section= PLT.16.3.3
[cxcv] SPEC:137	Testable=true	Section= PLT.16.3.3
[cxcvi] SPEC:138	Testable=true	Section= PLT.16.3.3
[cxcvii] SPEC:139	Testable=true	Section= PLT.16.3.3
[cxcviii] SPEC:140	Testable=false(impl)	Section= PLT.16.3.3
[cxcix] SPEC:141	Testable=true	Section= PLT.16.3.3
[cc] SPEC:142	Testable=true	Section=PLT.16.3.4
[cci] SPEC:143	Testable=true	Section=PLT.16.3.4
[ccii] SPEC:143	Testable=true	Section=PLT.16.3.4
[cciii] SPEC:48	Testable=true	Section=PLT.11.1.1
[cciv] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccv] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccvi] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccvii] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccviii] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccix] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccx] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccxi] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccxii] SPEC:48	Testable=true	Section=PLT.11.1.1
[ccxiii] SPEC:144	Testable=false(impl)	Section=PLT.17.1
[ccxiv] SPEC:145	Testable=false(impl)	Section=PLT.17.2
[ccxv] SPEC:146	Testable= false(impl)	Section=PLT.17.2
[ccxvi]SPEC:147	Testable= false	Section= PLT.19.2
[ccxvii] SPEC:148	Testable= false	Section= PLT.19.2
[ccxviii] SPEC:149	Testable=false	Section= PLT.19.5
[ccxix] SPEC:150	Testable=true	Section=PLT.19.5(servlet spec)
[ccxx] SPEC:151	Testable=true	Section= PLT.20.2
[ccxxi] SPEC:152	Testable=true	Section= PLT.20.2

ccxxii SPEC:153 Testable=true Section= PLT.20.2

ccxxiii SPEC:154 Testable=true Section= PLT.20.4

ccxxiv SPEC:155 Testable=true Section= PLT.20.4

ccxxv SPEC:156 Testable= true Section=PLT.22

ccxxvi SPEC:157 Testable=true Section= PLT.22.1

ccxxvii SPEC:158 Testable=false Section= PLT.22.1

ccxxviii SPEC:159 Testable=true Section= PLT.22.2

ccxxix SPEC:160 Testable=true Section= PLT.22.2

ccxxx SPEC:161 Testable=true Section= PLT.22.2

ccxxxi SPEC:162 Testable=true Section= PLT.22.2

ccxxxii SPEC:163 Testable=true Section= PLT.22.2

ccxxxiii SPEC:164 Testable=true Section= PLT.22.2

ccxxxiv SPEC:165 Testable=true Section= PLT.22.2

ccxxxv SPEC:166 Testable= true Section=PLT.22.2

ccxxxvi SPEC:167 Testable=false Section= PLT.22.2

ccxxxvii SPEC:168 Testable=true Section= PLT.22.2

ccxxxviii SPEC:169 Testable=true Section= PLT.22.2

ccxxxix SPEC:170 Testable=true Section= PLT.22.3

ccxl SPEC:171 Testable=true Section= PLT.22.3

ccxli SPEC:172 Testable=true Section= PLT.22.3

ccxlii SPEC:173 Testable=true Section= PLT.22.3

ccxliii SPEC:174 Testable=true Section= PLT.22.3

ccxliv SPEC:175 Testable= true Section=PLT.22.3

ccxlv SPEC:176 Testable=false Section= PLT.22.3

ccxlvi SPEC:168 Testable=true Section= PLT.22.2

ccxlvii SPEC:174 Testable=true Section= PLT.22.3

ccxlviii SPEC:174 Testable=true Section= PLT.22.3

ccxlix SPEC:174 Testable=true Section= PLT.22.3

ccl SPEC:175 Testable= true Section=PLT.22.3

[ccli] SPEC:176	Testable=false	Section= PLT.22.3
[cclii] SPEC:177	Testable=true	Section= PLT.22.4
[ccliii] SPEC:178	Testable=true	Section= PLT.22.5
[ccliv] SPEC:179	Testable=false	Section= PLT.22.5

[cclv] SPEC:178	Testable=true	Section= PLT.22.5
[cclvi] SPEC:68	Testable=true	Section=PLT.12.2.2
[cclvii] SPEC:68	Testable=true	Section=PLT.12.2.2