Java^{TM} Portlet Specification

Version 2.0

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

January 25, 2008

Stefan Hepper (sthepper@de.ibm.com)

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

Copyright 2008 IBM Corp. All rights reserved.

IBM Corporation (the “Spec Lead”), for the JSR 286 specification (the “Specification”), hereby grants permission to copy and display the Specification, in any medium without fee or royalty, provided that you include the following on ALL copies, or portions thereof, that you make:

1. A link or URL to the Specification at this location:

   http://www.jcp.org/en/jsr/detail?id=286

2. The copyright notice as shown herein.

The Spec Lead commits to grant a perpetual, non-exclusive, worldwide, non sub-licensable, non-transferable, fully paid up license, under royalty-free and other reasonable and non-discriminatory terms and conditions, to certain of their respective patent claims that the Spec Lead deems necessary to implement required portions of the Specification, provided a reciprocal license is granted.

THE SPECIFICATION IS PROVIDED "AS IS," AND THE SPEC LEAD AND ANY OTHER AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS. THE SPEC LEAD AND ANY OTHER AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SPECIFICATION OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of the Spec Lead or any other Authors may NOT be used in any manner, including advertising or publicity pertaining to the Specification or its contents without specific, written prior permission. Title to copyright in the Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.

Contents

PLT.1

Preface

This document is the Java^{TM} Portlet Specification, v2.0. The standard for the Java^{TM} Portlet API is described here.

PLT.1.1 Additional Sources

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

- A reference implementation (RI) has been made available which provides a behavioral benchmark for this specification. Where the specification leaves implementation of a particular feature open to interpretation, implementators may use the reference implementation as a model of how to carry out the intention of the specification
- A Technology Compatibility Kit (TCK) has been provided for assessing whether implementations meet the compatibility requirements of the Java^{TM} 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^{TM} 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 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, v1.4 (J2EE™)
- Java Servlet™, v2.4
- JavaServer Pages™, v2.0 (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 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)
- CC/PP Processing, JSR 188
- W3C: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies

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`

- `l`

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-286-comments@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 V 2.0

The Portlet Specification V2.0 was the result of the work of JSR286 Expert Group,

Subbu Allamaraju, Wesley Budziwojski, Bob Butler, Siddharth Chaudhary(Vignette), Padmanabh Dabke,

David H. DeWolf, Torsten Dettborn, Craig Doremus, Ate Douma, Jorge Ferrer (Liferay, LLC), Michael Freedman (Oracle), Kevin Frender (BEA Systems), Slava Frid, Deepak

Gothe(Sun Microsystems, Inc.), Vishal Goenka (SunGard Higher Education), Stefan Hepper (IBM Corp.), Martin Holzner (Novell, Inc.), Danny Machak (TIBCO Software Inc.), Kito D. Mann, Benjamin Mestrallet (eXo Platform SARL), Stephen Millidge, Jason Novotny, Punit Pandey, Ilya Rybak (SAP AG), Charles Severance, Hani Suleiman, David Sean Taylor (Apache Software Foundation), Julien Viet(JBoss, Inc.), James Ward (Adobe Systems Inc.), Carsten Ziegeler (Day).

We want to give special thanks to (as members of the Expert Group) Subbu Allamaraju, Wesley Budziwojski, Craig Doremus, Ate Douma, Michael Freedman, Kevin Frender for their major contributions to this specification.

Finally we would like to thank Torsten Dettborn (University of Jena) who led the TCK and RI efforts

PLT.1.9 Acknowledgements V 1.0

The Portlet Specification V1.0 was the result of the work of JSR168 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.2

Overview

PLT.2.1 What is a Portal?

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

PLT.2.2 What is a Portlet?

A portlet is 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 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.2.3 What is a Portlet Container?

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

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

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

PLT.2.4 An Example

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

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

PLT.2.5 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]

The only exceptions to this rule are:

- `RenderResponse.setContentType` is no longer required before calling `getWriter` or `getOutputstream`. Calling `getWriter` or `getOutputstream` without previously setting the content type results no longer in an IllegalStateException in V 2.0.
- `getProtocol` for included servlets / JSPs no longer returns `null`, but '`HTTP/1.1`' in V2.0.

PLT.2.6 Major changes introduced with V 2.0

The major new features of version 2.0 include:

- Events – enabling a portlet to send and receive events and perform state changes or send further events as a result of processing an event.
- Public render parameters – allowing portlets to share parameters with other portlets.
- Resource serving – provides the ability for a portlet to serve a resource.
- Portlet filter – allowing on-the-fly transformations of information in both the request to and the response from a portlet.

PLT.2.6.1 Clarifications that may make V1.0 Portlets Non-compliant

Depending on the implementation of the portlet of a specific runtime behavior of a portlet container the following clarifications may lead to different results when executing a portlet in either a JSR 168 or a JSR 286 container:

- XML escaping of portlet URLs produced via the portlet tag library.
  V 2.0 clarifies that the default is all portlet URLs are XML escaped. The default can be changed with the new attribute `escapeXML`. JSR 168 portlets depending on the fact that portlet URLs created via the tag library are not XML escaped can change the default to non-escaped via the portlet container runtime option `javax.portlet.escapeXml` (see PLT.26.7)
- Defining multiple values for the same parameter name in the Portlet `param` tag.
  V 2.0 clarifies that if the same name of a parameter occurs more than once within 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. Portlets assuming that the last occurrence wins and replaces the previous set values will behave differently in V2.0 containers.
- `getProtocol` for included servlets / JSPs no longer returns `null`.
  V 2.0 defines that getProtocol now returns '`HTTP/1.1`' and thus is better aligned with the servlet model that expects the `getProtocol` to return this value in the `GenericServlet`.
- Parameters set on the portlet URL and the post body are aggregated into the request parameter set. Portlet URL parameters are presented before post body data. JSR 168 did not define if and how post body and portlet URL parameters are being merged. The added clarification mirrors the behavior defined in the servlet specification for servlets.
- `RenderResponse.setContentType` is no longer required before calling `getWriter` or `getOutputstream`. Calling `getWriter` or `getOutputstream` without previously setting the content type will no longer result in an IllegalStateException.

PLT.2.6.2 Changes to the Programming Model

The following additions were made to the V1.0 programming model:

- Use application level resource bundles instead of inline localization in the `portlet.xml`.
  In V1.0 the only ability to localize values on the portlet application level was inside the `portlet.xml` using the xml:lang attribute. With V2.0 portlet application developers can now provide the localized values in a resource bundle and thus have the localized values in separate files instead of cluttering the deployment descriptor.

PLT.2.6.3 List of all Changes in the Specification

This section list all changes that are not editorial in nature:

- PLT.1.4: added JAXB 2.0 reference
- PLT 1.8: added V2.0 Acknowledgments
- PLT 2.5: added compatibility section
- PLT 2.6: added complete section with major changes introduced with V2.0
- PLT 2.7: added JavaSE and JavaEE requirements for V2.0, now based on servlet 2.4, J2EE 1.4 and Java 5.0 and a special Java 1.4 compiled version.
- PLT 3.0: updated to reflect new portlet capabilities
- PLT 3.2: added section on using servlet application lifecycle listeners for portlet applications
- PLT 4.1: added portlet section
- PLT 4.3: added section about portlets and web frameworks
- PLT 5.0, 5.2: added references to new lifecycle interfaces EventPortlet and ResourceServingPortlet
- PLT 5.2.3: moved the End of Service section from end of this chapter to this place
- PLT 5.3: grouped the existing sections Portlet Definition and Portlet Entity and Portlet Window to a new section Portlet Customization Levels
- PLT 5.4: added new event and resource lifecycle methods
- PLT 5.4.2: added Event Request section
- PLT 5.4.3: added reference to HTTP spec that render should be a safe operation
- PLT 5.4.4: added Resource Request section
- PLT 5.4.5.1: added Action Dispatching section
- PLT 5.4.5.2: added Event Dispatching section
- PLT 5.4.5.3: added Resource Serving Dispatching section
- PLT 5.4.5.4: added render mode annotation description
- PLT 6.3: added Default Event Namespace section
- PLT 6.4: added Public Render Parameter Names section
- PLT 6.5: added Processing Event QNames section
- PLT 6.6: added Publishing Event QNames section
- PLT 6.7: added Supported Locales section
- PLT 6.8: added Supported Container Runtime Options section
- PLT 7.1: added resource URLs and clarification that portlet URLs are only valid within the current request and may not be real URLs.

- PLT 7.1.1: added new BaseURL section with content of the old Portlet URL section in order to reflect the new BaseURL interface
- PLT 7.1.1.1: added new URL properties section
- PLT 7.1.2: clarified that if a portlet mode or window state is not set on a URL that the current portlet mode and window state is chosen as default
- PLT 7.1.3: clarified setSecure semantics
- PLT 7.2: added Portlet URL listeners section
- PLT 8.4: added portlet managed modes
- PLT 8.5: added renderMode annotation description
- PLT 8.7: added Setting next possible Portlet Modes section
- PLT 9.5: added Defining Window State Support section
- PLT 10.4: added Container Runtime Options section
- PLT 11.1.1: listed and described all methods to access request parameters
- PLT 11.1.1.1: added Form and Query Parameters section
- PLT 11.1.1.2: added clarification that request parameters are not propagated between different lifecycle requests
- PLT 11.1.1.3: added events to the description; clarified that the portlet receives always the render parameters explicitly set on a render URL; clarified that render parameters are cleared with each processAction and processEvent invocation
- PLT 11.1.1.4: added Resource Request Parameter section
- PLT 11.1.2: added Public Render Parameter section
- PLT 11.1.4.1: added User Information Request Attribute section
- PLT 11.1.4.2 : added CC/PP Request Attribute section
- PLT 11.1.4.3: added Render Part Request Attribute for Setting Headers in the Render Phase section
- PLT 11.1.4.4: added Lifecycle Phase Request Attribute section
- PLT 11.1.4.5: added Action-scoped Request Attributes section
- PLT 11.1.5.1: added Cookies section
- PLT 11.1.8: clarify that the getResponseContentType and getResponseContentType methods return the same value within a client request and added that these methods provide the information based on the HTTP Accept headers for serveResource calls.
- PLT 11.1.12: added Access to the Portlet Window ID section
- PLT 11.3: added ActionRequest Interface section
- PLT 11.4: added ResourceRequest Interface section
- PLT 11.5: added EventRequest Interface section
- PLT 12.1.1: clarify that response properties map to header values and introduce the new GenericPortlet.doHeaders method
- PLT 12.1.2: added resource URLs; clarified that returned URLs may not be real URLs
- PLT 12.1.3: added Namespacing section
- PLT 12.1.4: added Setting Cookies section
- PLT 12.2: added StateAwareResponse Interface section

- PLT 12.3: removed the parts of the ActionResponse that are now in StateAwareResponse
- PLT 12.3.1: added description for the new sendRedirect(String location, String renderUrlParamName) method
- PLT 12.4: added EventResponse Interface section
- PLT 12.5: added MimeResponse Interface section
- PLT 12.5.1: reduced the must requirement to set a content type to can and define the fallback of using the first entry in the getResponseContentTypes list
- PLT 12.5.2: added distinction between render and serveResource calls; clearified that for render the portlet should only use the raw OutputStream for binary content
- PLT 12.5.3: added Access to Response Headers section
- PLT 12.5.4: added Setting Markup Head Elements section
- PLT 12.5.6: added Predefined MimeResponse Properties section
- PLT 12.6 added RenderResponse Interface section and moved all parts that only apply to the render response to this new section
- PLT 12.6.1: added that portlets should set the new javax.portlet.renderHeaders container runtime option when using dynamic titles
- PLT 12.6.2: added Next possible portlet modes section
- PLT 12.7: added ResourceResponse Interface section
- PLT 12.8: added processEvent and serveResource references
- PLT 13: added chapter Resource Serving
- PLT 14: added chapter Serving Fragments through Portlets
- PLT 15: added chapter Coordination between portlets
- PLT 16.1: added Support for Markup Head Elements section
- PLT 17.1: added processEvent and serveResource references; clarified that setValue overrides previous values set with setValues
- PLT 17.4: removed the restriction that you need to have one instance of a validator per VM; added that portlet preferences should not be modified in the validator
- PLT 18.2: clarified that a portlet session object is only valid within the current client request
- PLT 18.3: added clarification that portlet session objects may be accessed in parallel; added reference to the new getWindowID method
- PLT 18.5: added Writing to the Portlet Session section
- PLT 19.1: removed the restriction of only having include in render
- PLT 19.2: added include and forward for all lifecycle methods; add restriction that the passed request / response pair must be either the original ones or wrappers using the new wrapper classes
- PLT 19.3: added that the servlet path lookup is based on the rules defined in SVR.11
- PLT 19.3.2: added attributes for all lifecycle phases
- PLT 19.3.3: added Request and Response Objects for Included Servlets/JSPs from within the Action and Event processing Methods section

- PLT 19.3.4: getRemotePort and getLocalPort now return ‘0’ instead of null; clarified that HttpUtils.getRequestURL is undefined; getProtocol now returns ‘HTTP/1.1’ instead of null
- PLT 19.3.5: added Request and Response Objects for Included Servlets/JSPs from within the ServeResource Method section
- PLT 19.3.6: added Comparison of the different Request Dispatcher Includes section
- PLT 19.3.8: added Path and Query Information in Included / Forwarded Servlets section
- PLT 19.4: added The forward Method section
- PLT 19.5: added Servlet filters and Request Dispatching section
- PLT 19.6: added Changing the Default Behavior for Included / Forwarded Session Scope section
- PLT 20: added Portlet Filter chapter
- PLT 21.1: changed that non-mapped user attributes must not be present in the map to a should
- PLT 21.2: changed the sample to use the new enumeration for P3P UserInfo
- PLT 22.1: changed must requirement of defining expiration based caching support in the deployment descriptor to should; added private and public cache scopes; added reference to new CacheControl interface; added clarification that cache settings should be set before writing to the output stream; included new lifecycle method processEvent and serveResource
- PLT 22.2: added Validation Cache section
- PLT 25: added reference to V1.0 deployment descriptor in the appendix
- PLT 25.1: added locale character set mapping
- PLT 25.2.2: added reference to the Java Product Versioning specification
- PLT 25.5: update to new V2.0 deployment descriptor; additions in V2.0 are:
    - in custom-portlet-mode section: added portal-managed element
    - added resource-bundle element on application level
    - added filter element on application level
    - added filter-mapping element on application level
    - added default-namespace element on application level
    - added event-definition element on application level
    - added public-render-parameter on application level
    - added listener element on application level
    - added container-runtime-option on application level
    - added cache-scope on portlet level
    - in supports section: added window-state element
    - added supported-processing-event element on portlet level
    - added supported-publishing-event element on portlet level
    - added supported-public-render-parameter element on portlet level
    - added container-runtime-option element on portlet level
- PLT 25.6: added Pictures of the structure of a Deployment Descriptor section
- PLT 25.7: added uniqueness requirements for new elements event-definition, public-render-parameters and filter

- PLT 25.8.1: added reference to new application level resource bundle
- PLT 25.8.2: added reference to RFC 1766; clarified that the supported locale information should be leveraged by the portal application
- PLT 25.10: added application level resource bundle description and pre-defined keys; in the portlet resource bundle table added keys for description and display-name
- PLT 26: added new V2.0 namespace; added reference to JSP 2.0 EL
- PLT 26.1: added new request/response variables; added portletSession variable; added portletSessionScope variable; added portletPreferences variable; added portlet PreferencesValues variable; changed sample to use new cache control API
- PLT 26.2: added attributes copyCurrentRenderParameters, escapeXml, name; added IllegalStateException; updated sample with new attributes
- PLT 26.3: added attributes copyCurrentRenderParameters, escapeXml; added IllegalStateException
- PLT 26.4: added resourceURL Tag section
- PLT 26.5: added the restriction that the namespace tag must match PortletResponse.getNamespace
- PLT 26.6: added new resourceURL tag; added description for empty values; added description for having multiple param tags with the same name
- PLT 26.7: added propertyTag section
- PLT 26.8: added Changing the Default Behavior for escapeXml section
- PLT 27: added Leveraging JAXB for Event payloads chapter
- PLT A: added new RenderMode annotation to samples
- PLT B: lessen restriction on iFrames from forbidden to not recommended
- PLT C.5: added Tables section
- PLT C.6: added rows portlet-form-field-label, portlet-form-field
- PLT C.7: added rows portlet-menu-cascade, portlet-menu-cascade-item, portlet-menu-cascade-item-selected, portlet-menu-cascade-item-hover, portlet-menu-cascade-item-hoverselected, portlet-menu-separator, portlet-menu-cascade-separator, portlet-menu-content, portlet-menu-content-selected, portlet-menu-content-hover, portlet-menu-content-hover-selected, portlet-menu-indicator, portlet-menu-indicator-selected, portlet-menu-indicator-hover,portlet-menu-indicator-hover-selected
- PLT D: split out bdate into year, month, day, hour, minute, second, fractionssecond, timezone; added user.login.id
- PLT E: added Deployment Descriptor Version 1.0 chapter
- PLT.F: updated list with new TCK assertions

PLT.2.6.4 List of all API changes

This section list all non-editorial API changes:

- ActionRequest:
    - extends ClientDataRequest
    - added ACTION_NAME constant

    - added getMethod
- ActionResponse:
    - extends StateAwareResponse
    - added sendRedirect(String location, String renderUrlParamName)
- added new BaseURL interface
- added new CacheControl interface
- added new Event interface
- added new EventInterface interface
- added new Event
- GenericPortlet
    - implements ResourceServingPortlet, EventPortlet
    - added new PortletConfig methods
    - added doHeaders method
    - added getNextPossiblePortletModes method
- added new MimeResponse interface
- PortalContext
    - added constant MARKUP_HEAD_ELEMENT_SUPPORT
- PortletConfig
    - added getPublicRenderParameterNames method
    - added getDefaultNamespace method
    - added getPublishingEventQNames method
    - added getProcessingEventQNames method
    - added getSupportedLocales method
    - added getContainerRuntimeOptions method
- PortletContext
    - added getContainerRuntimeOptions method
- PortletRequest
    - added constants CCPP_PROFILE, ACTION_PHASE, EVENT_PHASE, RENDER_PHASE, RESOURCE_PHASE, LIFECYCLE_PHASE, RENDER_PART, RENDER_HEADERS, RENDER_MARKUP, ACTION_SCOPE_ID
    - added enum P3PUserInfos
    - added getWindowID method
    - added getCookies method
    - added getPrivateParameterMap method
    - added getPublicParameterMap method
- PortletRequestDispatcher
    - added include(PortletRequest request, PortletResponse response) method
    - added forward method
- PortletResponse
    - changed getNamespace: lifetime is now for the portlet window instead just request
    - added addProperty(javax.servlet.http.Cookie cookie) method
    - added addProperty(String key, org.w3c.dom.Element element) method
    - added createElement method
- PortletSession

    - added getAttributeMap() method
    - added getAttributeMap(int scope) method
- PortletURL
    - extends BaseURL
    - added add/setProperty methods
    - added getParameterMap
    - added writer methods
    - added getPortletMode method
    - added getWindowState
    - added removePublicRenderParameter method
- added PortletURLGenerationListener interface
- added ProcessAction annotation
- added ProcessEvent annotation
- added RenderMode annotation
- RenderRequest
    - added constant ETAG
    - added getETag method
- RenderResponse
    - extends MimeResponse
    - added constants CACHE_SCOPE, PUBLIC_SCOPE, PRIVATE_SCOPE, ETAG, USE_CACHED_CONTENT, NAMESPACED_RESPONSE, MARKUP_HEAD_ELEMENT,
    - added createResourceURL method
    - added getCacheControl method
    - added setNextPossiblePortletModes method
- added ResourceRequest interface
- added ResourceResponse interface
- added ResourceServingPortlet interface
- added ResourceURL interface
- added StateAwareResponse interface
- added javax.portlet.filter package
    - added ActionFilter interface
    - added ActionRequestWrapper class
    - added ActionResponseWrapper class
    - added EventFilter interface
    - added EventRequestWrapper class
    - added EventResponseWrapper class
    - added FilterConfig interface
    - added FilterChain interface
    - added PortletFilter interface
    - added PortletRequestWrapper class
    - added PortletResponseWrapper class
    - added RenderFilter interface
    - added RenderRequestWrapper class
    - added RenderResponseWrapper class
    - added ResourceFilter interface

    - added ResourceRequestWrapper class
    - added ResourceResponseWrapper class

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

The Portlet API v2.0 is based on the Java Platform, Standard Edition 5.0 and Enterprise Edition v1.4. Portlet containers should at least meet the requirements, described in v 1.4 of the J2EE Specification, for executing in a J2EE environment.

As Portlet API v2.0 is intended to enable a common, composable programming model for web development with broad applicability, it is being defined to run across a variety of runtime environments including JavaME (CDC/Foundation) and JavaSE 1.4.2. Additionally, the Portlet API can be exploited in OSGi based Execution Environments that run on top of JavaME and Java SE. These are defined in the JCP via JSR 232 and JSR 291 respectively. The Java Portlet API jar files comply with the OSGi specification and thus can be deployed as OSGi bundles on servers supporting OSGi.

The following Java SE 5.0 features will not be available in the Portlet API V2.0 compiled for Java SE 1.4:

- enum P3PUserInfos
- annotations ProcessAction, ProcessEvent, RenderMode
- generics for collections

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 2.4* whenever applicable.

PLT.3

Relationship with the Servlet Specification

The *Servlet Specification* 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 in the render method, not complete documents. The Portal aggregates portlet markup fragments into a complete portal page
- Portlets can only be invoked through URLs constructed via the portlet API.
- Web clients interact with portlets through a portal system
- Portlets have more refined request handling, i.e. action requests, event request, render request and resource requests
- Portlets have predefined portlet modes and window states that indicate the function the portlet is performing and the amount of real estate 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 a means of accessing and storing persistent configuration and customization data
- Portlets have access to user profile information
- Portlets have URL rewriting functions for creating hyperlinks within their content, which allow portal server agnostic creation of links and actions in page fragments
- Portlets can store transient data in the portlet session in two different scopes: the application-wide scope and the portlet private scope.
- Portlets can send and receive events from other portlets or can receive container defined events.

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

- Setting the character set encoding of the render response
- The URL of the client request to the portal

The portlet has full control over the response when rendering resources via the `serveResource` call.

Because of these differences, the Expert Group has decided that portlets need 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 a portlet application. Portlets, servlets and JSPs within the same portlet application share the 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, by default:

- Attributes set in the portlet request are available in the included servlet request (see *PLT.19 Dispatching Requests to Servlets and JSPs* Chapter),
- The portlet and the included servlet or JSP share the same output stream (see *PLT.19 Dispatching Requests to Servlets and JSPs* Chapter).
- Attributes set in the portlet session are accessible from the servlet session and vice versa (see *PLT.18 Portlet Session* Chapter).

PLT.3.2 Using Servlet Application Lifecycle Events

In chapter SRV.10 the Java Servlet Specification describes a variety of application lifecycle events that the servlet can register for. The following portlet objects defined by this specification mirror its servlet counterparts: `PortletContext` and `PortletSession`. The lifecycle of the `PortletContext` is tied to the `SevletContext` of this web application and the attributes set in the `PortletContext` are mirrored in the `ServletContext`. The lifecycle of the `PortletSession` is tied to the `HttpSession` of this web application and the attributes set in the `PortletSession` are mirrored in the `HttpSession`. Due to this fact the servlet lifecycle listeners for `ServletContext` and `HttpSession` can also be used for notifications on the `PortletContext` and `PortletSession` operations.

Given that the portlet request is independent of the servlet request the servlet request lifecycle listeners do not have a simple mapping to portlet requests. In order to allow portlets to leverage the servlet request listeners for portlets the portlet container needs to create a servlet request mirroring the portlet request. In order to allow the servlet request listeners to distinguish between the case of a plain servlet request and a servlet request targeted towards a portlet the portlet container needs to set the attribute `javax.portlet.lifecycle_phase` in order to mark this request as a request targeted to a portlet.

The following is the list of servlet listeners that also apply to portlets:

- `javax.servlet.ServletContextListener` – for notifications about the servlet context and the corresponding portlet context
- `javax.servlet.ServletContextAttributeListener` – for notifications on attributes in the servlet context or the corresponding portlet context.
- `javax.servlet.http.HttpSessionActivationListener` – for notifications on the activation or passivation of the `HTTPSession` or the corresponding `PortletSession`.
- `javax.servlet.http.HttpSessionAttributeListener` – for notifications on attibutes of the `HTTPSession` or the corresponding `PortletSession`.
- `javax.servlet.http.HttpSessionBindingListener` - for notifications on binding of object to the `HTTPSession` or the corresponding `PortletSession`.
- `javax.servlet.ServletRequestListener` – for notifications about changes to the `HTTPServletRequest` or the mirrored portlet request of the current web application.`javax.servlet.ServletRequestAttributeEvent` - for notifications about changes to the attributes of the `HTTPServletRequest` or the mirrored portlet request of the current web application.

PLT.3.3 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 2.4*.

PLT.4

Portlet Concepts

PLT.4.1 Portlets

Portlets provide a componentized user interface (UI) for services. In a Service Oriented Architecture (SOA) one does not write monolithic applications, 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.

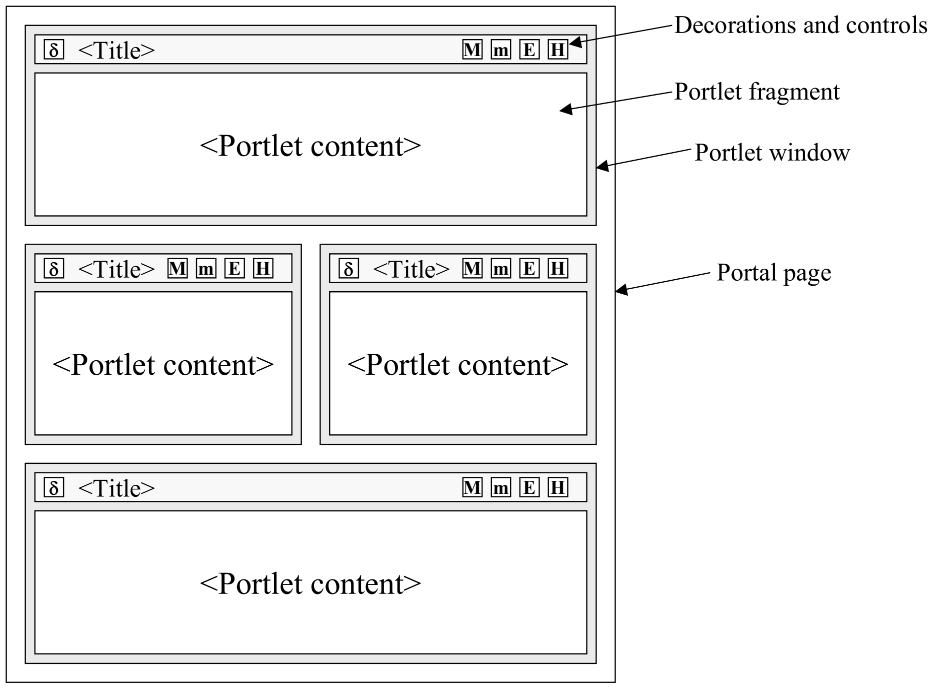
Portlets provide such a component UI model that is intended to aggregate the component UIs into a larger UI with consistent look and feel (see Appendix PLT.C Style Sheet Definitions). The Java Portlet Specification allows coordination on the UI layer with different means, such as events, application sessions, and public 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.4.2 Embedding Portlets as Elements of a Portal Page

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

Figure 4-1 Elements of a Portal Page

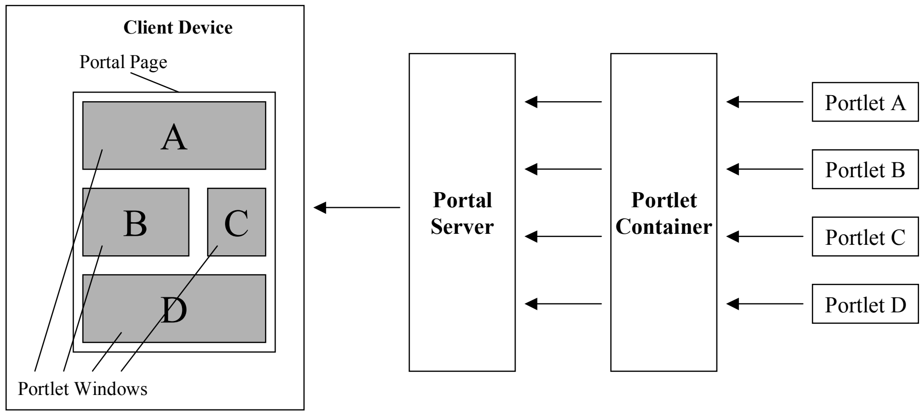


Note that this is only one example on how a portal could make use of the portlet markup fragment. There may exist other portal implementations with a different rendering approach. The important part of the portal page concept in regards to this specification is that the markup fragment of the portlet may 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.4.2.1 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.4.2.2 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 markup which is performed via the `render` and `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 technology 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, Spring MVC framework, Wicket, 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 of making the implementation of such bridges simpler.

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 `EventPortlet` and `ResourceServingPortlet` in order to leverage additional functionality for receiving / sending events or serving resources, respectively.

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

PLT.5.1 Number of Portlet Instances

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

For a portlet, not hosted in a distributed environment (the default), the portlet container must[ii] 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). [iii]

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 `EventPortlet` and `ResourceServingPortlet` that the portlet can implement.

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.[iv] After loading the portlet classes, the portlet container instantiates them for use.

PLT.5.2.2 Initialization

After the portlet object is instantiated, the portlet container must initialize the portlet before invoking it to handle requests.[v] 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.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.[vi] The `destroy` method must not be called because the initialization is considered unsuccessful.[vii]

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.[viii]

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

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 container-defined time period 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.[x] 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.[xi]

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

After the `destroy` method completes, the portlet container must release the portlet object so that it is eligible for garbage collection.[xiii] 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 distinguish between the different 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.17 Portlet Preferences* Chapter).

At runtime, when serving requests, one or more preference objects are associated with a portlet. The resulting association of a specific preference object with a portlet 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 than 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 N:M relationship between portlet windows 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 constant and 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.18.3*)

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 any 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, the portlet may publish one or more events, which result in one or more invocations of the `processEvent` method of this or another portlet with the portlet request referred to as an event request. 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 three types, action URLs, 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 for this client. A client request triggered by a render URL translates into many render requests, one per portlet in the portal page. In addition a render URL may result in `processEvent` calls for container-defined events. A client request trigged by a resource URL translates into a serve 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.[xiv] The portal/portlet-container must wait until the action request finishes. Then, the portal/portlet-container should 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.[xv] The render requests may be executed sequentially or in parallel without any guaranteed order.

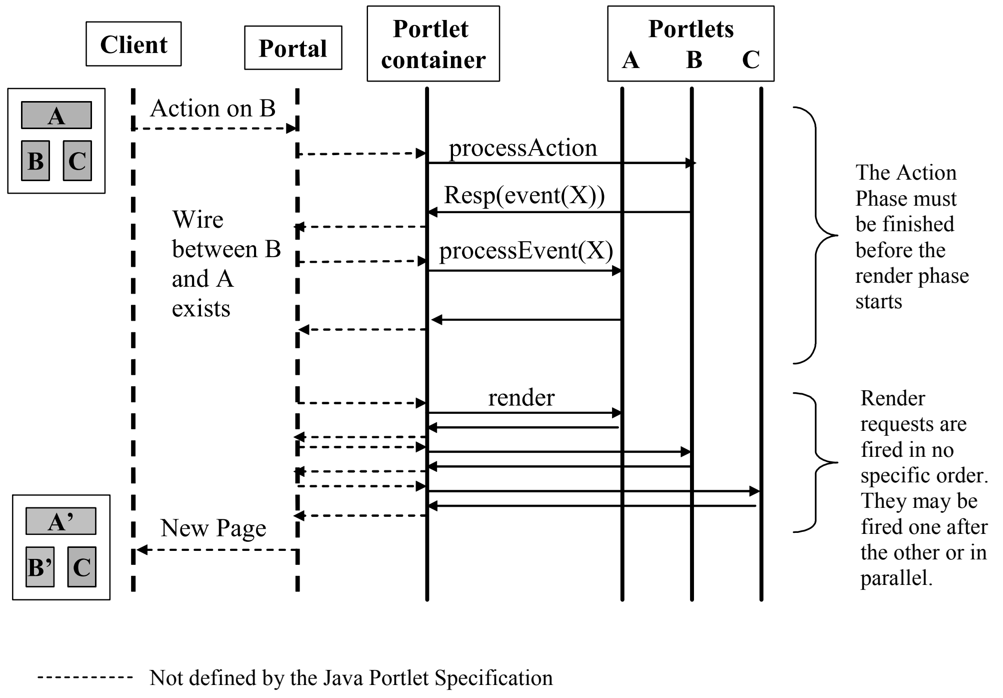
If the client request is triggered by a render URL, the portal/portlet-container must invoke the `render` method for all the portlets in the portal page with the possible exception of portlets for which their content is being cached.[xvi] The portal/portlet-container must not invoke the `processAction`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

If a portlet has caching enabled, the 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.22 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


Client
Portal
Portlet container
Portlets A B C
A
B
C
Action on B
processAction
Resp(event(X))
Wire between B and A exists
processEvent(X)
render
A'
B'
C
New Page
The Action Phase must be finished before the render phase starts
Render requests are fired in no specific order. They may be fired one after the other or in parallel.
---------- Not defined by the Java Portlet Specification

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[xviii] 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 requests 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 requests 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 delegate the action processing to a servlet via a request dispatcher call (see *PLT.19 Dispatching Requests to Servlets and JSPs*).

The portlet may publish events via the `ActionResponse setEvent` method and thus publish state changes or other notifications to other portlets. See *PLT.15* 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 requests 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 requests 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 delegate the event processing to a servlet via a request dispatcher call (see *PLT.19 Dispatching Requests to Servlets and JSPs*).

The portlet may publish events via the `EventResponse setEvent` method and thus publish state changes and other notifications to other portlets. See *PLT.15* for more details on sending and receiving events.

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.19 Dispatching Requests to Servlets and JSPs* Chapter for details on this.

The portlet should not trigger any state changes in a render request and be a safe operation as defined by the HTTP specification (see RFC 2616, http://www.w3.org/Protocols/rfc2616/rfc2616.html).

PLT.5.4.4 Resource Request

In order to serve resources or render content fragments 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 `ResourceResponse`.

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 `ResourceResponse` writer or output stream, or it may delegate the generation of content to a servlet or a JSP. Refer to *PLT.19 Dispatching Requests to Servlets and JSPs* Chapter for details on this.

More details on serving resources can be found in PLT.13.

PLT.5.4.5 GenericPortlet

The `GenericPortlet` abstract class provides default functionality and convenience methods for handling events, resource and render requests. By extending `GenericPortlet` portlets also get robust against future changes in the Java Portlet Specification as they can be mitigated in the implementation of `GenericPortlet`.

PLT.5.4.5.1 Action Dispatching

For a received action the `processAction` method in the `GenericPortlet` class tries to dispatch to methods annotated with the tag `@ProcessAction(name=<action name>)`, where the action name must be set on the `ActionURL` as value of the parameter `javax.portlet.action` (or via the constant `ActionRequest.ACTION_NAME`), and following signature:

```
void <methodname> (ActionRequest, ActionResponse) throws
                  PortletException, java.io.IOException;
```

A portlet that wants to leverage this action dispatching needs to set the parameter `ActionRequest.ACTION_NAME` on the action URL.

PLT.5.4.5.2 Event Dispatching

For a received event the `processEvent` method in the `GenericPortlet` class tries to dispatch to methods annotated with the tag `@ProcessEvent(qname=<event name>)`, where the event name must be in the format "{" + Namespace URI + "}" + local part (like used by `javax.xml.namespace.QName.toString()` method). For using only the local part of the event name and leverage the default namespace defined in the portlet deployment descriptor with the `default-namespace` element the following alternative is provided: `@ProcessEvent (name=<event name_local_part>)`, where the event name is only the local part. If the Namespace URI `.equals(javax.xml.XMLConstants.NULL_NS_URI)`, only the local part is used. The method annotated with the `@ProcessEvent` annotation must have the following signature:

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

If no such method can be found the `GenericPortlet` just sets the received render parameters as new render parameters.

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.

PLT.5.4.5.3 Resource Serving Dispatching

The `serveResource` method in the `GenericPortlet` class tries to forward the resource serving to the resource ID set on the URL triggering the request for serving the resource. If no resource ID is set, the `serveResource` method does nothing.

PLT.5.4.5.4 Rendering Dispatching

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).

First it tries to dispatch to methods annotated with the tag `@RenderMode(name=<portlet mode name>)`. The method must have the following signature:

```
  void <methodname> (RenderRequest, RenderResponse) throws
                      PortletException, java.io.IOException;
```

If no matching annotated method is found `GenericPortlet` will dispatch to the following methods:

- `doView` for handling VIEW requests[xix]
- `doEdit` for handling EDIT requests[xx]
- `doHelp` for handling HELP requests[xxi]

For any other portlet mode the `GenericPortlet` will throw a `PortletException` per default.

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.[xxii]

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

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, or any of the optional lifecycle methods, like `processEvent`, or `serveResource`, at any particular time.

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` or `processEvent` method, all operations on the ActionResponse must be ignored including set events.[xxiii] 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.[xxiv] 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`.[xxv]

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

PLT.5.4.8 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`, `processEvent`, `serveResource` 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.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, default event namespace, public render parameter names, 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.25.10 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.[xxvi]

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.25.10 Resource Bundles* Section.[xxvii]

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.6.3 Default Event Namespace

The `getDefaultNamespace` method of the `PortletConfig` interface returns the default namespace for events and public render parameters set in the portlet deployment descriptor with the default-namespace element, or the XML default namespace `XMLConstants.NULL_NS_URI` if no default namespace is provided in the portlet deployment descriptor. [xxviii]

PLT.6.4 Public Render Parameter Names

The `getPublicRenderParameterNames` method of the `PortletConfig` interface returns the public render parameter names found in the portlet definition in the deployment descriptor with the `supported-public-render-parameter` element or an empty enumeration if no public render parameters are defined for the current portlet definition. [xxix]

PLT.6.5 Publishing Event QNames

The `getPublishingEventQNames` method of the `PortletConfig` interface returns the publishing event QNames found in the portlet definition in the deployment descriptor with the `supported-publishing-event` element or an empty enumeration if no publishing events are defined for the current portlet definition. [xxx]

If the event was defined using the `name` element instead of the `qname` element the defined default namespace must be added as namespace for the returned QName. [xxxi]

PLT.6.6 Processing Event QNames

The `getProcessingEventQNames` method of the `PortletConfig` interface returns the processing event QNames found in the portlet definition in the deployment descriptor with the `supported-processing-event` element or an empty enumeration if no processing events are defined for the current portlet definition. [xxxii]

If the event was defined using the `name` element instead of the `qname` element the defined default namespace must be added as namespace for the returned QName. [xxxiii]

PLT.6.7 Supported Locales

The `getSupportedLocales` method of the `PortletConfig` interface returns the supported locales found in the portlet definition in the deployment descriptor with the `supported-locale` element or an empty enumeration if no supported locales are defined for the current portlet definition. [xxxiv]

PLT.6.8 Supported Container Runtime Options

The `getContainerRuntimeOptions` method returns an immutable Map containing portlet application level container runtime options merged with the portlet level container runtime options, containing the names as keys and the container runtime values as map values, or an empty Map if no portlet application level or portlet level container runtime options are set in the `portlet.xml` or supported by this portlet container. The map returned from `getContainerRuntimeOptions` will provide the subset the portlet container supports of the options the portlet has specified in the portlet deployment descriptor. The keys in the map are of type String. The values in the map are of type String array. If a container runtime option is set on the portlet application level and on the portlet level with the same name the setting on the portlet level takes precedence and overwrites the one set on the portal application level.

See section PLT 10.4 for a list of all predefined container runtime options.

PLT.7

Portlet URLs

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

PLT.7.1 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/ResourceURL` objects invoking the `createActionURL`, `createRenderURL` or the `createResourceURL` methods of the `PortletResponse` interface. The `createActionURL` method creates action URLs. The `createRenderURL` method creates render URLs. The `createResourceURL` method creates resource URLs. 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 of a render URL.[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, i.e. that change state, 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 may become bookmarkable.

Note that Render URLs used within forms may not work on all portal/portlet-containers as the portal/portlet-container may ignore form parameters.

A resource URL allows the portlet serving resources with access to information of the portlet request. When rendering resources the portlet has full control over the output stream and can render binary markup.

Note that portlet URLs are only valid within the current request and need to be either written to the output stream in order to allow re-writing the portlet URL token into a real URL.

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 either as a resource URL, action URL, or render URL.

Portlets can add application specific parameters to the `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 `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 an `ActionURL` or `RenderURL`. When creating a `ResourceURL` the current render parameters are automatically added to that URL by the portlet container, but are hidden to the `getParameter` calls of the portlet URL object. Setting parameters on an `ActionURL` will result in action parameters, not render parameters or public render parameters.

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

If portlet developers namespace parameter names or values before adding them to a `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 namespace them properly to avoid collisions with the parameters set and used by the portlet.[xl]

Using the `toString` method, a portlet can obtain the string representation of the `BaseURL`. If the portlet wants to include a portlet URL in the portlet content it should use the `write` method and avoid the string object creation of the `toString` method.

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.write(writer);
writer.print("\">Summary</A>");
...
```

Portlet developers should be aware that the string representation of a `PortletURL` or `ResourceURL` 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. It may even be an ECMA script method that may generate the URL at the time the user clicks on the link.

PLT.7.1.1.1 URL Properties

Properties can be used by portlets to set vendor specific information on the PortletURL object and thus use extended URL capabilities.

A portlet can set properties using the following methods of the `BaseURL` 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.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.write(writer);
writer.print("\">");
...
```

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.[xli]. The change of portlet mode must be effective for the request triggered by the portlet URL.[xlii] There are some exceptional circumstances, such as changes in 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[xliii].

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.[xliv] 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[xlv].

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 should be of the same security level of the current request. If `setSecure` is called with `true`, the transport for the request triggered with this URL must be secure (i.e. HTTPS). [xlvi] If set to `false` the portlet indicates that it does not require a secure connection for the request triggered with such a URL.

PLT.7.2 Portlet URL listeners

Portlets can register portlet URL listeners in order to filter URLs before they get generated either as a string via the `toString method` or written to the output stream via the `write` method of the `BaseURL` interface. The portlet URL listener is also called for a render URL that is added to a redirect URL via the method `sendRedirect(location, renderUrlParamName).`

For example the portlet could use URL listeners to set the caching level of resource URLs in one central piece of code (see PLT13.7).

In order to receive a callback from the portlet container before a portlet URL is generated the listener class needs to implement the `PortletURLGenerationListener` interface and register it in the deployment descriptor.

PLT.7.2.1 PortletURLGenerationListener Interface

The `PortletURLGenerationListener` interface provides callbacks for each portlet URL type. If the portlet application has specified one or more `PortletURLGenerationListener`

classes in the portlet deployment descriptor the portlet container must call

- the method `filterActionURL` method for all action URLs before executing the `write` or `toString` method of these action URLs[xlvii]
- the method `filterRenderURL` method for all render URLs before executing the `write` or `toString` method of these render URLs[xlviii]
- the method `filterResourceURL` method for all resource URLs before executing the `write` or `toString` method of these resource URLs[xlix]

The portlet container must provide the `PortletURL` or `ResourceURL` to generate to the filter methods and execute the `write` or `toString` method on the updated `PortletURL` or `ResourceURL` that is the outcome of the filter method call.[l]

PLT.7.2.2 Registering Portlet URL Listeners

Portlet applications must register Portlet URL listeners in the portlet deployment descriptor under the application section with the `listener` element and provide the class name that implements the `PortletURLGenerationListener` as value in the `listener-class` element.

If more than one listener is registered the portlet container must chain the listeners in the order of how they appear in the deployment descriptor.[li]

PLT.8

Portlet Modes

A portlet mode indicates the function a portlet is performing in the `render` method. 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.17 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 for modes that need to be managed by the portal. Portlets may define additional modes that don't need to be managed by the portal and correspond to the `VIEW` mode from a portal point of view. The portlet must declare portlet modes that are not managed by the portal via the `<portal-managed>false</portal-managed>` tag. Portlet modes are considered portal managed by default.

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 portal managed custom portlet modes defined in the deployment descriptors should be mapped to custom portlet modes supported by the portal implementation. Portlets that list custom portlet modes that are not managed by the portal may provide a localized decoration name as resource bundle entry with the key `javax.portlet.app.custom-portlet-mode.<name>.decoration-name` for this portlet mode. If no entry in the portlet resource bundle with such a name exists the portal / portlet container should use the portlet mode name as default decoration name.

If a custom portlet mode defined in the deployment descriptor is not mapped to a custom portlet mode provided by the portal or otherwise supported as non-managed portlet mode, portlets must not be invoked in that portlet mode.

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

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

```
        <portal-managed>false</portal-managed>
      </custom-portlet-mode>

      <custom-portlet-mode>
        <description>Provides administration functions</description>
        <portlet-mode>admin</portlet-mode>
        <portal-managed>true</portal-managed>
      </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. Therefore providing a decoration name or `portal-managed` element for the modes defined in *PLT.A* is not necessary.

PLT.8.5 GenericPortlet Render Handling

The `GenericPortlet` class implementation of the `render` method dispatches requests to the methods annotated with the tag `@RenderMode(name=<portlet mode name>)`. The method must have the following signature:

```
void <methodname> (RenderRequest, RenderResponse) throws
                   PortletException, java.io.IOException;
```

If no matching annotated method is found `GenericPortlet` will dispatch to the `doView`, `doEdit` or `doHelp` method depending on the portlet mode indicated in the request using the `doDispatch` method or throws a `PortletException` if the mode is not `VIEW`, `EDIT`, or `HELP`.[lii]

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 in the `render` method. As all portlets must support the `VIEW` portlet mode, `VIEW` does not have to be indicated.[liii] The portlet must not be invoked in a portlet mode that has not been declared as supported for a given markup type.[liv]

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.[lv]

PLT.8.7 Setting next possible Portlet Modes

Via the render response the portlet can set next possible portlet modes that make sense from the portlet point of view. If set, the portal should honor these enumeration of portlet modes and only provide the end user with choices to the provided portlet modes or a subset of these modes based on access control considerations. If the portlet does not set any next possible portlet modes the default is that all portlet modes that the portlet has defined supporting in the portlet deployment descriptor are meaningful new portlet modes. In order to ensure that the next possible portlet modes are honored by all portal implementations the portlet should set the `javax.portlet.renderHeaders` container runtime option and either overwrite the `getNextPossiblePortletModes` method in the `GenericPortlet` or set the next possible portlet modes in the `RENDER_HEADERS` sub-phase of the render phase (see PLT.11.1.1.4.3) via `setNextPossiblePortletModes`. This allows that the portal receives these suggested new modes before writing the portlet window decorations and thus is able to optimize the amount of buffering needed.

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 via the `render` method. 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.[lvi]

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>
    <window-state>half_page</window-state>
  </custom-window-state>
  ...
</portlet-app>
```

PLT.9.5 Defining Window State Support

Portlets may restrict within their definition, in the deployment descriptor, the custom window states they can handle for each markup type they support in the `render` method. If the portlet does not list explicitly which window states it supports, the portal / portlet container should assume that the portlet supports all pre-defined window states and all custom window states defined for this portlet application.

As all portlets must at least support the `pre-defined` window states `NORMAL`, `MAXIMIZED`, `MINIMIZED`, these window states do not have to be indicated.[lvii] The portlet should not be invoked in a custom window state that has not been declared as supported for a given markup type.

The following example shows a snippet of the window states 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>
    <window-state>half-page</window-state>
    ...
</supports>
<supports>
    <mime-type>text/vnd.wap.wml</mime-type>
    <portlet-mode>help</portlet-mode>
    ...
</supports>
...
```

For HTML markup, this portlet supports the `HALF-PAGE` window state in addition to the required pre-defined window states. For WML markup, it supports only the pre-defined window states.

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

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, application and portlet runtime options 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.[lix] 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.[lx]

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. However, the context objects themselves may be different objects.

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.[lxi]

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.[lxii]

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

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 SVR 3 Servlet Context* Chapter, `javax.servlet.context.tempdir`.[lxiv] The portlet context must follow the same behavior and functionality that the servlet context has for virtual hosting and reloading considerations. (see *Servlet Specification SVR 3 Servlet Context* Chapter)[lxv]:

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`.

PLT.10.4 Portlet Container Runtime Options

The portlet can define additional runtime behavior in the `portlet.xml` on either the portlet application level or the portlet level with the container-runtime-option element. Runtime options that are defined on the application level should be applied to all portlets in the portlet application. Runtime options that are defined on the portlet level should be applied for this portlet only and override any runtime options defined on the application level with the same name.

Container runtime options besides the `javax.portlet.actionScopedRequestAttributes` option are optional to support by the portlet container and the portlet can find out which container runtime options are supported by the portlet container running the portlet via the method `getContainerRuntimeOptions` on the `PortletContext`.

The `getContainerRuntimeOptions` method returns an enumeration of type String containing the keys of all container runtime options that the current portlet container supports.

PLT.10.4.1 Runtime Option javax.portlet.escapeXml

In the Java Portlet Specification V1.0 the behavior in regards to XML escaping URLs written by the tag library was undefined and thus portlets may have been coded with the assumption that the URLs were not XML escaped. In order to be able to run these portlets on a Java Portlet Specification V 2.0 container the specification provides the `javax.portlet.escapeXml` container runtime option. The value of this setting can either be `true` for XML escaping URLs per default, or `false` for not XML escaping URLs per default.

Portlets that require that the default behavior for URLs written to the output stream via the portlet tag library should therefore define the following container runtime option in the portlet deployment descriptor:

```
<portlet>

…

      <container-runtime-option>

            <name>javax.portlet.escapeXml</name>

            <value>false</value>

      </container-runtime-option>

</portlet>
```

If the portlet has defined the `javax.portlet.escapeXml` container runtime option the portlet container should honor this setting as otherwise the portlet may not work correctly.

PLT.10.4.2 Runtime Option javax.portlet.renderHeaders

Portlets that need to write any headers in the render phase can set the additional container-runtime-option with name `javax.portlet.renderHeaders` and value `true`. The default for this setting is `false`. When set to `true` streaming portal implementations should call the `render` method of the portlet twice with `RENDER_PART` attribute set in the render request (see PLT.11.1.4.3.). Example:

```
<portlet>

…

      <container-runtime-option>

            <name>javax.portlet.renderHeaders</name>

            <value>true</value>

      </container-runtime-option>

</portlet>
```

PLT.10.4.3 Runtime Option javax.portlet.servletDefaultSessionScope

The default for the session variable of included / forwarded servlets or JSPs is that it maps to the portlet session with application scope. Some portlets may require that the session variable of included / forwarded servlets or JSPs maps instead to the portlet session scope in order to work correctly. These portlets can indicate this via setting the `container-runtime-option` `javax.portlet.servletDefaultSessionScope` to `PORTLET_SCOPE`. The default for `javax.portlet.servletDefaultSessionScope` is `APPLICATION_SCOPE`.

Example:

```
<portlet>

…

      <container-runtime-option>

            <name>javax.portlet.servletDefaultSessionScope</name>

            <value>PORTLET_SCOPE</value>

      </container-runtime-option>

</portlet>
```

Portlet developers should note that not all portlet containers may be able to provide this feature as a portable JavaEE solution does not currently exist. Therefore, relying on this feature may restrict the numbers of portlet containers the portlet can be executed on.

PLT.10.4.4 Runtime Option javax.portlet.actionScopedRequestAttributes

The Java Portlet Specification follows a model of separating concerns in different lifecycle methods, like `processAction, processEvent, render`. This provides a clean separation of the action semantics from the rendering of the content, however, it may create some issues with servlet-based applications that don't follow this strict Model-View-Controller pattern. Such applications in some cases assume that attributes that they set in the action phase will be accessible again when starting the rendering. The Java Portlet Specification provides the render parameters for such use cases, but some applications need to transport complex objects instead of strings.

One example for such an use case is a Java Server Faces (JSF) bridge portlet that expects to be executed in a single lifecycle phase for processing actions, events and rendering from the JSF point of view and thus needs to transport attributes from action to subsequent event and render calls until the next action occurs.

For such use cases the Java Portlet Specification provides the action-scoped request attributes as container runtime option with the intent to provide portlets with these request attributes until a new action occurs. This container runtime option must be supported by portlet containers. [lxvi]

Portlets should note that using this container runtime option will result in increased memory usage and thus may have a decreased performance as the portlet container needs to maintain and store these attributes across requests.

Portlets that want to leverage the action-scoped request attributes need to set the container runtime option `javax.portlet.actionScopedRequestAttributes` to true, default is false. In addition the portlet may provide a value called `numberOfCachedScopes` where the following value element must be a positive number indicating the number of scopes the portlets wants to have cached by the portlet container. This value is a hint to the portlet container that the portlet container may not be able to honor because of resource constraints. The order of the values in the portlet deployment descriptor must be `true`, `numberOfCachedScopes`, `<number of cached scopes>`.

Example:

```
<portlet>

…

        <container-runtime-option>


<name>javax.portlet.actionScopedRequestAttributes</name>

                <value>true</value>

                <value>numberOfCachedScopes</value>

                <value>10</value>

        </container-runtime-option>

</portlet>
```

PLT.10.4.4.1 Action Scope ID Render Parameter

The portlet container must store the action scope ID as render parameter with the name "`javax.portlet.as`", defined as `PortletRequest.ACTION_SCOPE_ID`. When using the action-scoped request attribute extension the portlet must not use this render parameter name for its private render parameters.

The portlet container must provide the action scope ID render parameter and its value when calling one of the portlet lifecycle methods and is responsible for setting this action scope ID at the end of a `processAction` or `processEvent` method call. The portlet should not set a value for the render parameter named `PortletRequest.ACTION_SCOPE_ID ("javax.portlet.as")`.

If the portlet removes the `PortletRequest.ATION_SCOPE_ID` render parameter in a PortletURL listener the portlet container should honor this and create a portlet URL without this render parameter. This allows the portlet to create resource URLs that are cacheable across action scopes.

PLT.10.4.4.2 Lifetime of Action-scoped Request Attributes

The portlet can view attributes set on action, event, or resource requests in any of its lifecycle requests lasting until the next action occurs, or until some timeout or invalidation mechanism of the portlet container frees up the occupied memory, e.g. the user session has timed out.

A new action scope is started when

- receiving an action – starts a new action scope with a new scope ID, all previous attributes are no longer accessible, new attributes can be stored.
- receiving a render without an existing scope ID – starts a new scope without any scope ID, all previous attributes are no longer accessible, no new attributes can be stored.
- receiving an event without an existing scope ID - starts a new action scope with a new scope ID, all previous attributes are no longer accessible, new attributes can be stored.
- receiving an event with an existing scope ID after the first render for this scope had occurred, as this event will likely have an action semantic. All previous attributes are no longer accessible, new attributes can be stored.

The existing scope is preserved with the current scope ID and action-scoped attributes when

- receiving a `render` call with an existing scope ID

- receiving a `processEvent` call with an existing scope ID before the first render for this scope had occurred.
- receiving a `serveResouce` call with an existing scope ID

The following attributes are not stored in the action scope by the portlet container:

- all attributes starting with `javax.portlet`
- all Java Portlet Specification defined objects, like request, response, session, as they are only valid for the current request
- any other attributes the portal/portlet container provides itself for handling the lifecycle call

The portlet may also filter out attributes that should not be stored in the action-scope at the end of the request either via `removeAttribute` or via a response filter.

If portlets use non-serializable objects as attribute values they may not be provided across different requests, e.g. if the portlet container leverages mechanisms such as a session and session replication. However, portlet containers should either provide the complete set of attributes to the portlet or discard the entire set of attributes in order to allow the portlet to always run in a consistent state.

PLT.10.4.4.3 ServeResource Calls

If a `serveResource` call is triggered by a resource URL with a cache level of `FULL` the action scope ID may not be included and thus the portlet may not have access to the action-scoped attributes.

PLT.10.4.4.4 Examples

Example 1:

- portlet receives a processAction call and sets attribute foo, new scope contains foo
- portlet receives a processEvent call reads foo and sets bar, scope contains foo, bar
- portlet receives a render call, scope contains foo, bar
- portlet receives a processEvent call and sets foo2, new scope contains foo2
- portlet receives a render call, scope contains foo2

Example 2:

- portlet receives a render call, empty scope
- portlet receives a processEvent call and sets foo and bar, new scope contains foo, bar
- portlet receives a serveResource call, scope contains foo, bar and sets foo' and bar2, new scope contains foo', bar and bar 2

PLT.10.4.4.5 Semantics for Portlet Containers

In order to provide a consistent user experience for end users the portlet container should keep previous action-scoped attributes cached in order to allow the end user to navigate between different views with the browser forward and backward buttons. The portlet container should use the specified `numberOfCachedScopes` provided by the portlet or a meaningful default if the portlet has not provided this value.

In order to determine if a render has already occurred for the current action-scope it is assumed that the portlet container stores a bit invisible to the portlet in the action-scoped attributes that indicates if a render has already occurred.

PLT.11

Portlet Requests

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

PLT.11.1 PortletRequest Interface

The `PortletRequest` interface defines the common functionality for all the request interfaces.

PLT.11.1.1 Request Parameters

If a portlet receives a request from a client request targeted to the portlet itself, the parameters must be the string parameters 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.[lxvii] The parameters the request object returns must be "`x-www-form-urlencoded`" decoded.[lxviii]

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`
- `getPublicParameterMap`
- `getPrivateParameterMap`

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` [lxix]. If there is a single parameter value associated with a parameter name the method must return is an array of size one containing the parameter value.[lxx]. The `getParameterMap` method must return an unmodifiable `Map` object[lxxi]. If the request does not have any parameters, the `getParameterMap` must return an empty `Map` object[lxxii].The values in the returned `Map` object are from type `String` array.

Parameters set on the portlet URL and the post body are aggregated into the request parameter set. Portlet URL parameters are presented before post body data. [lxxiii]

If portlets 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.

PLT.11.1.1.1 Form and Query Parameters

If the portlet is performing an HTML Form submission via HTTP method POST the post form data will be populated to the portlet request parameter set if the content type is `application/x-www-form-urlencoded`.

If the post form data are populated to the portlet request parameters the post form data will no longer be available for reading directly from the request object's input stream. If the post form data is not included in the parameter set, the post data must still be available to the portlet via the `ActionRequest` / `ResourceRequest` input stream.

If the portlet is performing an HTML Form submission via the HTTP method GET the form data set is appended to the portlet URL used for the form submission and are therefore accessible as request parameters for the portlet.

Note that some portal/portlet-containers implementations may encode internal state as part of the URL query string and therefore do not support forms using the HTTP GET method.

As portlet URLs may be ECMA script functions that produce the required URL only on executing the URL the portlet should not simply add additional query parameters to a portlet URL on the client.

PLT.11.1.1.2 Action and Event Request Parameters

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

If a portlet wants to do that in either the `processAction` or `processEvent` methods, it must use the `setRenderParameter` or `setRenderParameters` methods of the `StateAwareResponse` object within the `processAction` or `processEvent` call. The set render parameters must be provided to the `processEvent` and `render` calls of at least the current client request. [lxxvi]

PLT.11.1.1.3 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, the parameters should be the same parameters as of the previous render request from this client.

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

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 or event request.[lxxvii]

If a portlet receives a render request that is the result of invoking a render URL targeting this portlet the render parameters received with the render request must be the parameters set on the render URL if these were not changed by the portlet as a result of an container event received for this render URL.[lxxviii]

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.[lxxix]

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

Note that render parameters get automatically cleared if the portlet receives a `processAction` or `processEvent` call and need to be explicitly re-set on the response of such a lifecycle call.

PLT.11.1.1.4 Resource Request Parameters

For `serveResource` requests the portlet must receive any resource parameters that were explicitly set on the `ResourceURL` that triggered the request. If the cacheability level of that resource URL (see PLT.13.7) was `PORTLET` or `PAGE`, the portlet must also receive the render parameters present in the request in which the URL was created

If a resource parameter is set that has the same name as a render parameter, the render parameter must be the last entry in the parameter value array.

PLT.11.1.2 Public Render Parameters

In order to allow coordination of render parameters with other portlets, within the same portlet application or across portlet applications, the portlet can declare public render parameters in its deployment descriptor using the `public-render-parameter` element in the portlet application section. Public render parameters are available in all lifecycle methods of the portlet: `processAction`, `processEvent`, `render`, and `serveResource`. Public render parameters can be viewed and changed by other portlets or components. In the portlet section each portlet can specify the public render parameters it would like to share via the `supported-public-render-parameter` element. The `supported-public-render-parameter` element must reference the identifier of a public render parameter defined in the portlet application section in a `public-render-parameter` element[lxxxi]. The portlet should use the defined public render parameter identifier in its code in order to access the public render parameter.

Example:

```
<public-render-parameter>
        <identifier>foo</identifier>
        <qname xmlns:x=”http://example.com/params”>x:foo2</qname>
</public-render-parameter>
<public-render-parameter>
        <identifier>bar</identifier>
        <qname xmlns:x=”http://example.com/params”>x:foobar</qname>
</public-render-parameter>
<portlet>
        <portlet-name>portletA</portlet-name>
        …
        <supported-public-render-parameter>foo</supported-public-render-parameter>
</portlet>
<portlet>
        <portlet-name>portletB</portlet-name>
        …
        <supported-public-render-parameter>bar</supported-public-render-parameter>
</portlet>
```

The portlet container must only send those public render parameters to a portlet which the portlet has defined support for using `supported-public-render-parameter` element in the portlet.xml[lxxxii]. The portlet container must only share those render parameters of a portlet which the portlet has declared as supported public render parameters using `supported-public-render-parameter` element in the portlet.xml [lxxxiii]. The portlet container is free to only provide a subset of the defined public render parameters to portlets that are not target of a render URL as storing of render parameters is only encouraged, but not mandated for portal / portlet container implementations. A public 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 public render parameter the portlet must receive at least this render parameter [lxxxiv]

A portlet can access the public render parameters in any lifecycle method via the `getPublicParameterMap` method of the portlet request. In addition the portlet can access public render parameters via the `getParameter` and `getParameterMap` methods. In the case of a `processAction` or `serveResource` call the public parameters are merged with the action / resource parameters set on the action / resource URL. If a action or resource parameter has the same name as a public render parameter the public render parameter values must be the last entries in the parameter value array. [lxxxv]

If a portlet wants to delete a public render parameter it needs to use the `removePublicRenderParameter` method on the `StateAwareResponse` or the `PortletURL`.

By default all public render parameters declared by the portlet will be provided in the current request. In order to minimize updates a portlet should only set public render parameters explicitly on a render URL, if the values in the target request should be different from the parameter values of the current request.

Portlets can access a merged set of public and private parameters via the `getParameter` methods on the `PortletRequest` or separated as maps of private parameters via the `getPrivateParameterMap` method and public parameters via the `getPublicParameterMap` method. [lxxxvi]

The `qname` element should uniquely identify the public render parameter and use the QNames as defined in the XML specifications: XML Schema Part2: Datatypes specification (http://www.w3.org/TR/xmlschema-2/#QName), Namespaces in XML (http://www.w3.org/TR/REC-xml-names/#ns-qualnames), Namespaces in XML Errata (http://www.w3.org/XML/xml-names-19990114-errata), TAG Finding: Using Qualified Names (QNames) as Identifiers in Content (http://www.w3.org/2001/tag/doc/qnameids-2002-06-17).

As an alternative the portlet can specify a default namespace via the `default-namespace` element that will be applied to all public render parameters defined only with a local name with the `name` element in the public render parameter definition section.

It is up to the portal implementation to decide which portlets may share the same public render parameters. The portal should use the information provided in the deployment descriptor, like the name, qname, alias names and description, in order to perform such a mapping between public render parameters of different portlets. It is also an implementation choice of the portal whether different portlet windows of the same portlet will receive the same public render parameters. An example where different portlet windows may not want to share the same render parameters is a generic viewer portlet that takes as public 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 parameters for administration and configuration tools, developers should provide a display name in the portlet application ResourceBundle (see the *PLT.25.10 Resource Bundles* Section). The entry for the display name should be constructed as `'javax.portlet.app.public-render-parameter.<identifier>.display-name'`.

PLT.11.1.3 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. [lxxxvii] It is the responsibility of the portal/portlet-container to properly 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 and for use by portal/portlet-container implementations.

PLT.11.1.4 Request Attributes

Request attributes are objects associated with a portlet during a single portlet request. Portlets can not assume that attributes are public 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.11.1.4.1 The User Information Request Attribute

The portlet can access a map with user information attributes via the request attribute `PortletRequest.USER_INFO`.[lxxxviii] See Chapter 20, User Information for more details.

PLT.11.1.4.2 The CC/PP Request Attribute

The portlet can access a Composite Capability/Preference Profile (CC/PP, W3C: Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies http://www.w3.org/TR/2001/WD-CCPP-struct-vocab-20010315/) javax.ccpp.profile via the request attribute `PortletRequest.CCPP_PROFILE`. The `PortletRequest.CCPP_PROFILE` request attribute must return a `javax.ccpp.Profile` based on the current portlet request.[lxxxix] It may contain additional CC/PP information set by the portal / portlet container. (See JSR 188 (CC/PP Processing, http://jcp.org/en/jsr/detail?id=188) for more details on CC/PP profile processing).

Note that once the CC/PP profile API provides a factory method taking the `PortletRequest` / `PortletResponse` as parameters this attribute may become deprecated.

PLT.11.1.4.3 The Render Part Request Attribute for Setting Headers in the Render Phase

There are cases in which the portlet may want to return header information, or other information that is required before getting the markup, like the portlet title or the next possible portlet modes, in the render phase. However, some portal implementations may choose to implement itself in a streaming manner and thus do not buffer the output of the portlet. In order to support these implementations the Java Portlet Specification provides the `javax.portlet.renderHeaders` container runtime setting and the `RENDER_PART` request attribute that these streaming portal implementations need to set. Portlets that want to ensure that they run with maximum performance on all portal implementations should leverage this mechanism for:

- Setting cookies
- Setting headers
- Setting the title
- Returning new possible portlet modes

Portlets that need to set any of the above mentioned headers should set the additional container-runtime-option with name `javax.portlet.renderHeaders` and value `true`. The default for this setting is `false`. When set to `true` streaming portal implementations should call the `render` method of the portlet twice with `RENDER_PART` attribute set in the render request. Example:

```
<portlet>

…

      <container-runtime-option>

            <name>javax.portlet.renderHeaders</name>

            <value>true</value>

      </container-runtime-option>

</portlet>
```

If the `RENDER_PART` portlet request attribute is set it indicates that the render request needs to be split into two parts:

1. The render headers part that must be indicated by setting the `RENDER_PART` request attribute with the value `RENDER_HEADERS`. In this part the portlet should only set the header related data, cookies, the next possible portlet modes and the portlet title. The portlet can set cache information for this response that may differ from the one set on the `RENDER_MARKUP` response.
2. The render markup part that must be indicated by setting the `RENDER_PART` request attribute with the value `RENDER_MARKUP`. In this part the portlet should produce only its markup.

Non-streaming portals will not set this attribute and thus the portlet should set headers, portlet title and produce its markup in a single render request.

Portlets should either extend `GenericPortlet`, which provides handling of the `RENDER_PART` request attribute in the render method, or check for the `RENDER_PART` request attribute themselves.

PLT.11.1.4.4 The Lifecycle Phase Request Attribute

The `LIFECYCLE_PHASE` request attribute of the `PortletRequest` interface allows a portlet to determine the current lifecycle phase of this request. This attribute value must be `ACTION_PHASE` if the current request is of type `ActionRequest`, `EVENT_PHASE` if the current request is of type `EventRequest`, `RENDER_PHASE` if the current request is of type `RenderRequest`, and `RESOURCE_SERVING_PHASE` if the current request is of type `ResourceRequest`. [xc]

The main intent of this attribute is to allow frameworks implemented on top of the Java Portlet Specification to perform the correct type casts from the `PortletRequest/PortletResponse` to a specific request/response pair, like `ActionRequest/ActionResponse`.

PLT.11.1.4.5 Action-scoped Request Attributes

The Java Portlet Specification follows a model of separating concerns in different lifecycle methods, like `processAction, processEvent, render`. This provides a clean separation of the action semantics from the rendering of the content, however, it may create some issues with servlet-based applications that don't follow this strict Model-View-Controller pattern. Such applications in some cases assume that attributes that they set in the action phase will be accessible again when starting the rendering. The Java Portlet Specification provides the render parameters for such use cases, but some applications need to transport complex objects instead of strings.

For such use cases the Java Portlet Specification provides the action-scoped request attributes as container runtime option with the intent to provide portlets with these request attributes until a new action occurs.

Section PLT.10.1.4.4 describes this option in more detail.

PLT.11.1.5 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.11.1.5.1 Cookies

The portlet can access cookies provided by the current request with the `getCookies` method. The returned cookie array provides the portlet with all cookie properties.

PLT.11.1.6 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.[xci] 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.[xcii]

PLT.11.1.7 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`.[xciii]

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

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

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

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

PLT.11.1.8 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.[xciv]

The returned values of the `getResponseContentType` and `getResponseContentTypes` call are the same for `processAction, processEvent` and `render` calls occurring within the same client request.

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.

If the `getResponseContentType or getResponseContentTypes` methods are exposed via an `ActionRequest, EventRequest`, or `RenderRequest` the following additional restrictions apply:

- The content type must only includes the MIME type, not the character set. [xcv] The character set of the response can be retrieved via the `RenderResponse.getCharacterEncoding`.
- The `getResponseContentTypes` method must return only the content types supported by the current portlet mode of the portlet.[xcvi]

If the `getResponseContentType or getResponseContentTypes` methods are exposed via an ResourceRequest the returned values should be based on the HTTP Accept header provided by the client.

PLT.11.1.9 Internationalization

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

PLT.11.1.10 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. Note that the `VIEW` mode is always allowed, even if not explicitly listed in the portlet definition.

PLT.11.1.11 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.11.1.12 Access to the Portlet Window ID

The `getWindowID` method of the `PortletRequest` interface provides the portlet with the current portlet window ID. The portlet window ID must be unique for this portlet window and constant for the lifetime of the portlet window. The portlet window ID retrieved with the `getWindowID` method must be the same as the one that is used by the portlet container for scoping the portlet-scope session attributes. [xcvii]

PLT.11.2 ClientDataRequest Interface

The `ClientDataRequest` interface extends the `PortletRequest` interface and it is used as base class for the `ActionRequest` and `ResourceRequest`. In addition to the functionality provided by the `PortletRequest` interface, the `ClientDataRequest` interface represents the request information of the HTTP request issued from the client to the consuming application / portal, such as the input stream.

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 `ClientDataRequest` interface also provides a `getReader` method that retrieves the HTTP POST data as character data according to the character encoding defined in the 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`.[xcviii]

To help manage the input stream, the `ClientDataRequest` 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.[xcix]

PLT.11.3 ActionRequest Interface

The `ActionRequest` interface extends the `ClientDataRequest` interface and is used in the `processAction` method of the `Portlet` interface. Currently, the `ActionRequest` interface does not define any additional methods but only the `ACTION_NAME` constant that can be used together with the `@ProcessAction` annotation.

PLT.11.4 ResourceRequest Interface

The `ResourceRequest` interface extends the `ClientDataRequest` interface and is used in the `serveResource` method of the `ResourceServingPortlet` interface. The

`ResourceRequest` interface defines in addition the `ETAG` constant and the `getETag` method for validation based caching and the `getResourceID` method for getting the resource ID set on the resource URL.

PLT.11.5 EventRequest Interface

The `EventRequest` interface extends the `PortletRequest` interface and is used in the `processEvent` method of the `EventPortlet` interface. 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 QName via `getQName`.

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 methods.

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.

PLT.12

Portlet Responses

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

PLT.12.1 PortletResponse Interface

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

PLT.12.1.1 Response Properties

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

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

- `setProperty`
- `addProperty`

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

Response properties can be viewed as header values set for the portal application. If these header values are intended to be transmitted to the client they should be set before the response is committed. When setting headers in the render lifecycle phase portlets should set the header in the render headers part or simply override the `GenericPortlet.doHeaders` method (see PLT.11.1.1.4.3).

The portlet should note that headers set on the response are not guaranteed to be transported to the client as the portal application may restrict headers due to security reasons, or they may conflict with other headers set by other portlets on the page.

PLT.12.1.2 Encoding of URLs

Portlets may generate content with URLs referring to other resources within the portlet application, 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 may return the URL unchanged.

Resources that are addressed not by an URL encoded with `encodeURL`, or directly via a ResourceURL, are not guaranteed to be accessible.

Portlet developer should be aware that the returned URL might 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. As a result, the outcome of the `encodeURL` call may be different than calling `encodeURL` in the servlet world.

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.[c] For example, the `getNamespace` method will 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.[ci]

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

PLT.12.1.4 Setting Cookies

A portlet can set HTTP cookies at the response via the `addProperty` method with a `javax.servlet.http.Cookie` as parameter. The portal application is not required to transfer the cookie to the client. Thus the portlet should not assume that it has access to the cookie on the client or that request triggered with URLs not generated by the portlet API can access the cookie.

Cookies set in the response of one lifecycle call should be available to the portlet in the subsequent lifecycle calls, e.g. setting a cookie in `processAction` should enable the portlet to retrieve the cookie in the next `render` call.

For requests triggered via portlet URLs the portlet should receive back the cookie. Cookies can be retrieved via the `request.getCookies` method.

Cookies are properties and all restrictions said above about properties also apply for cookies, i.e. to be successfully transmitted back to the client, cookies must be set before the response is committed. Cookies set in `render` or `serveResource` after the response is committed will be ignored by the portlet container.

When setting cookies in the render lifecycle phase portlets should set the cookies in the render headers part or simply override the `GenericPortlet.doHeaders` method in order to run with maximum performance on all portal implementations (see PLT.11.1.1.4.3).

PLT.12.2 StateAwareResponse Interface

The `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. [ciii] 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 occur which 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 StateAwareResponse.

The `removePublicRenderParameter` method allows the portlet to remove a public render parameter.

PLT.12.2.2 Portlet Modes and Window State Changes

The `setPortletMode` method allows a portlet to change its current portlet mode. The new portlet mode will be effective in the following `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.[civ]

The `setWindowState` method allows a portlet to change its current window state. The new window state will 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.[cv]

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.

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

PLT.12.2.3 Publishing Events

The portlet can publish events via the `setEvent` method. It is also valid to call `setEvent` multiple times in the current `processAction` or `processEvent` method and thus publish multiple events (see PLT. 15.2).

PLT.12.3 ActionResponse Interface

The `ActionResponse` interface extends the `StateAwareResponse` interface and it is used in the `processAction` method of the `Portlet` interface. This interface also allows a portlet to redirect the user to another URL.

PLT.12.3.1 Redirections

The `sendRedirect(String location)` 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.[cvi]

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

The `sendRedirect(String location, String renderUrlParamName)` 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.[cviii]

The portlet container must attach a render URL with the currently set portlet mode, window state and render parameters on the `ActionResponse` and the current public render parameters. [cix] The attached URL must be available as query parameter value under the key provided with the `renderUrlParamName` parameter. [cx]

New values for portlet mode, window state, private or public render parameters must be encoded in the attached render URL[cxi], but are not remembered after the redirect is issued.

Sending events when doing a redirect is discouraged as these events may be discarded by the portlet container / portal application as the further processing of the event may result in state changes that the portlet container would not be able to honor because of the performed redirect.

PLT.12.4 EventResponse Interface

The `EventResponse` interface extends the `StateAwareResponse` interface and adds the additional method `setRenderParameters(EventRequest request)`. One thing to note is that if a portlet receives multiple `processEvent` calls while processing one client request the new portlet mode or window state that the portlet may have set, may be not validated by the portal between these 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.12.5 MimeResponse Interface

The `MimeResponse` interface extends the `PortletResponse` interface and is used as base interface for `RenderResponse` and `ResourceResponse`. In addition to the `PortletResponse` interface the `MimeResponse` interface provides the functionality to create MIME-based content that is returned to the portal application.

PLT.12.5.1 Content Type

A portlet can set the content type of the response using the `setContentType` method of the `MimeResponse` interface in order to indicate to the portlet container which content type the portlet has chosen.

For the render response 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 `PortletRequest` object[cxii]. For the render response the portlet container should ignore any character encoding specified as part of the content type and treat the content type as if the character encoding was not specified.

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`.[cxiii]

If the portlet does not specify a content type before the `getWriter` or `getPortletOutputStream` methods the portlet container assumes the content type of the `PortletRequest.getResponseContentType()` method and resolves wildcards on a best-can-do basis.

PLT.12.5.2 Output Stream and Writer Objects

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

The termination of the `render` or `serveResource` method of the portlet indicates that the portlet has satisfied the request and that the output buffer is to be flushed.

In `render` the raw `OutputStream` is available because of some servlet container implementations requirements and for portlets that do not generate markup fragments. Portlets should only use the raw `OutputStream` for binary content and use the `Writer` for text-based markup. If a portlet utilizes the `OutputStream`, the portlet is responsible for using the proper character encoding.

PLT.12.5.3 Access to Response Headers

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

Note that it is not guaranteed that headers, like cookies, will be transmitted all the way back to the client.

For render calls, portlets should set headers in the render headers part of the render lifecycle phase or simply override the `GenericPortlet.doHeaders` method (see PLT.11.1.4.3) in order to run with maximum performance on all portal implementations.

PLT.12.5.4 Setting Markup Head Elements

A portlet can set markup head elements at the response via the `addProperty` method with `MimeResponse.MARKUP_HEAD_ELEMENT` (value: `"javax.portlet.markup.head.element"`) as property name and an `org.w3c.dom.Element` value.

This property is intended to be a hint to the portal application that the provided DOM element should be added to the markup head section of the response to the client.

Support for this property is optional and the portlet can verify if the calling portal supports this property via the `MARKUP_HEAD_ELEMENT_SUPPORT` property on the `PortalContext`.

Even if the calling portal supports this property, delivery of the DOM element to the client cannot be guaranteed, e.g. due to possible security rules of the portal application or elements that conflict with the response of other portlets.

For render calls, portlets should set head properties in the render headers part of the render lifecycle phase or simply override the `GenericPortlet.doHeaders` method (see PLT.11.1.4.3) in order to run with maximum performance on all portal implementations.

PLT.12.5.5 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 `MimeResponse` interface allow a portlet to access and set buffering information:

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

These methods are provided on the `MimeResponse` 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).[cxv]

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.[cxvi] 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.[cxvii] 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.[cxviii] The response and its associated buffer must be unchanged.[cxix]

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

PLT.12.5.6 Predefined MimeResponse Properties

The `MimeResponse` interface defines some property names that allow portlets leveraging these extensions to interoperate across different portal / portlet container implementations.

PLT.12.5.6.1 Cache properties

The `MimeResponse` defines the property names `CACHE_SCOPE`, `EXPIRATION_CACHE`, `ETAG` and `USE_CACHED_CONTENT` and the property values `PRIVATE_SCOPE` and `PUBLIC_SCOPE`, which can be used for validating expired content, setting new expiration times and cache scopes. See PLT.22 for more details.

PLT.12.5.6.2 Namespaced Response Property

The `NAMESPACED_RESPONSE` constant is intended to be a hint to the portal application that the returned content is completely namespaced. This includes all markup id elements, form fields, etc. One example where this might be used is for portal applications that are form-based and thus need to re-write any forms included in the portlet markup.

This property needs to be set using the `setProperty` method with a non-null value. The value itself is not evaluated. The value of the `NAMESPACED_RESPONSE` constant is X-

`JAVAX-PORTLET-NAMESPACED-RESPONSE` indicating that it is intended to be a header in the portlet response to the portal application.Portlets should set the namespaced property in the render headers part of the render lifecycle phase or simply override the `GenericPortlet.doHeaders` method in order to run with maximum performance on all portal implementations (see PLT.11.1.4.3).

PLT.12.6 RenderResponse Interface

The `RenderResponse` interface extends the `MimeResponse` interface and it is used in the `render` method of the `Portlet` interface. This interface allows a portlet to set its title, indicate the next possible portlet modes, and generate content.

The portlet cannot set the character encoding or the locale of the response as these are pre-set by the portal / portlet container.

PLT.12.6.1 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.[cxxi]

Portlets should set the `javax.portlet.renderHeaders` container runtime option and either set the title in the render headers part of the render lifecycle phase (see PLT.11.1.1.4.3) or simply override the `GenericPortlet.getTitle` method in order to run with maximum performance on all portal implementations.

PLT.12.6.2 Next possible portlet modes

A portlet may indicate to the portal application the next possible portlet modes that make sense from the portlet point of view via the `setNextPossiblePortletModes` method.

If set, the portal should honor these enumeration of portlet modes and only provide the end user with choices to the provided portlet modes or a subset of these modes based on access control considerations.

If the portlet does not set any next possible portlet modes the default is that all portlet modes that the portlet has defined supporting in the portlet deployment descriptor are meaningful new portlet modes.

In order to ensure that the next possible portlet modes are honored by all portal implementations, portlets should set the `javax.portlet.renderHeaders` container runtime option and either set the next possible portlet modes in the render headers part of the render lifecycle phase (see PLT.11.1.1.4.3) or simply override the `GenericPortlet.`

`getNextPossiblePortletModes` method in order to run with maximum performance on all portal implementations.

PLT.12.7 ResourceResponse Interface

The `ResourceResponse` interface extends the `MimeResponse` interface and 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.

The portlet can set the character encoding or the locale of the response. The portal / portlet container may pre-set character encoding and locale.

PLT.12.7.1 Setting the Response Character Set

The portlet can set the character encoding for a resource response in several ways:

- via the `setCharacterEncoding` method
- via the `setContentType` method. Calls to `setContentType` set the character encoding only if the given content type string provides a value for the `charset` attribute.
- via the `setLocale` method and a `locale-encoding-mapping-list` mapping in the `web.xml` deployment descriptor (see servlet specification SVR.5.4 for details). Calls to `setLocale` set the character encoding only if neither `setCharacterEncoding` nor `setContentType` has set the character encoding before.

If the portlet does not set a character encoding via one of the above listed methods before calling `getWriter` UTF-8 is applied by the portlet container as default character encoding.

PLT.12.8 Lifetime of Response Objects

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

PLT.13

Resource 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 web application. These links are constructed by the portlet and encoded with the `PortletResponse.encodeURL()` method.
   Note that this method might not return a valid URL.
   Direct links are not guaranteed to pass through the portal server and 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 via resource URLs through the portal.
2. Resource URL links pointing back to the portlet. Via these links the `serveResource` method of the `ResourceServingPortlet` interface is called and the portlet can serve the resource. Thus resources served via resource URLs 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 portal applications 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 serve the resource.

PLT.13.1 ResourceServingPortlet Interface

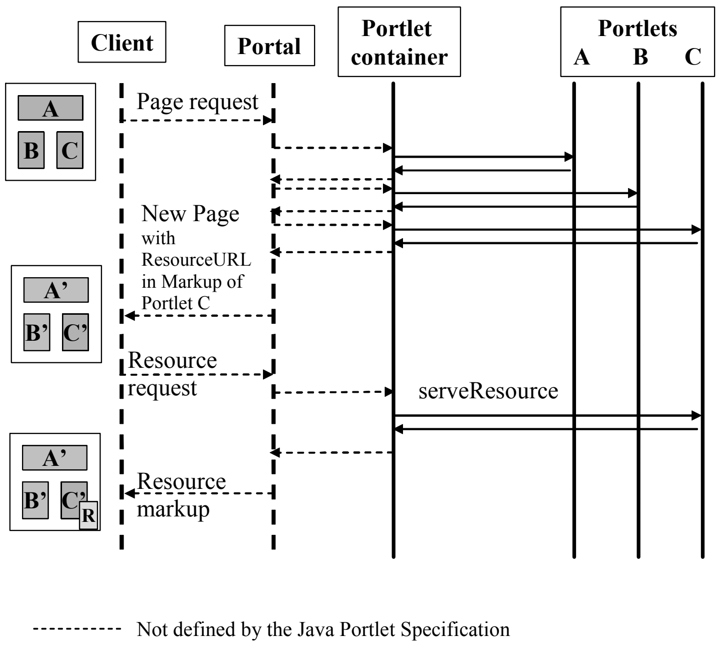
A portlet that wants to serve resources addressed via a resource URL must implement the `ResourceServingPortlet` interface with the method `serveResource`. The portlet container must not render any output in addition to the content returned by the `serveResource` call. For `serveResource` calls the portal application should just act as a proxy for accessing the resource.

The `serveResource` call normally follows a `render` call and can be viewed as a logical extension the render phase. The portlet should not change any state in the `serveResource` call that was issued via an HTTP method GET.

For use cases that require state changes the `serveResource` call should be issued via an HTTP method POST or PUT or DELETE. For `serveResource` calls only state changes to non-shared state, like the portlet session scope or portlet preferences, should be performed as otherwise portlets participating in this shared state would display stale markup. The portlet should note that such state changes impact cachability of the resource and set the cache settings accordingly.

The `serveResource` call can also be used to implement Asynchronous Javascript and XML (AJAX) use cases (see Chapter 14).

Figure 13-1 Resource Request Handling Sequence


Client
Portal
Portlet container
Portlets
A
B
C
Page request
New Page with ResourceURL in Markup of Portlet C
Resource request
serveResource
Resource markup
A'
B'
C'
R
---------- Not defined by the Java Portlet Specification

PLT.13.2 Access to Render Parameters, Portlet Mode, and Window State

The `ResourceRequest` should be provided with the current portlet mode and window state. The `ResourceRequest` call should also be provided with the current render parameters of the portlet.

PLT.13.3 Access to Request and Response Headers

Given that the portal / portlet container does not render any additional markup for a `serveResource` 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 *PLT 11.1.5*).

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.

The portlet should be aware that the portal application may filter out some headers due to the fact that it has already set these headers to a different value or because of security reasons.

PLT.13.4 Getting the HTTP Method

The portlet must be able to get the HTTP method with which this request was made, for example, GET, POST, or PUT, via the `getMethod` call on the `ResourceRequest`. [cxxii]

PLT.13.5 Access to the Resource ID

The portlet must be able to get the resource ID that was set on the resource URL with the `setResourceID` method via the `getResourceID` method from the resource request. [cxxiii] If no resource ID was set on the resource URL the `getResourceID` method must return `null`. [cxxiv]

PLT.13.6 Resource URLs

The portlet can create resource URLs pointing back to itself via the `createResourceURL` method on the `PortletResponse`. When an end user invokes such a resource URL the portlet container must call the `serveResource` method of the portlet or return a valid cached result for this resource URL[cxxv] If the portlet does not implement the `ResourceServingPortlet` 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 `processEvent` method[cxxvi]. Resource URLs should be provided with the current portlet mode, window state, and render parameters that the portlet can access via the ResourceRequest with `getPortletMode`, `getWindowState`, or one of the `getParameter` methods. ResourceURLs cannot change the current portlet mode, window state or render parameters[cxxvii]. Parameters set on a resource URL are not render parameters but parameters for serving this resource and will last only for 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 values must be the last entries in the parameter value array. [cxxviii]

PLT.13.7 Caching of Resources

The supported use cases for `serveResource` include retrieving new markup fragments based on the current portlet state and allowing the portlet to include portlet URLs in the returned markup. If portlet URLs are included in the markup, portals / portlet containers must create correct portlet URLs for all text-based markup types. [cxxix] If the returned markup of the `serveResource` call includes portlet URLs the cachability of the markup on the browser will most likely be limited as a common practice of portal application is to encode the state of the portlets in the URL.

With the `setCacheability` method on the `ResourceURL` the portlet can indicate that it only needs parts of the overall state via the cache level parameter and thus the portal application can create URLs that result in an increased likelihood of a subsequent browser access being served from a browser/web cache. With the `getCachability` method on the `ResourceURL` the portlet can retrieve the current cache level.

The following values are defined for the cache level parameter:

- FULL – The resource URL does not need to contain the current state of the page or the current render parameters, portlet mode, or window state of the portlet. Thus the portlet should not access the portlet mode, window state, or render parameters in the `serveResource` call.
  Only URLs with a cache level `FULL` are allowed in the response of the `serveResource` call triggered via a `ResourceURL` with a cache level `FULL`. The same restriction is true for all downstream URLs that result from this `serveResource` call. Setting a cachability different from `FULL` must result in an `IllegalStateException`[cxxx]. Attempts to create URLs that are not of type `FULL` or are not resource URLs in the current or a downstream response must result in an `IllegalStateException`[cxxxi].
  In order to enable sharing of the resource between different portlet applications the portlet can set a unique ID, preferable a `QName` in the `QName.toString` format, via the property key `ResourceURL.SHARED` on the resource URL. This unique ID is intended to allow the portal application identifying resource links that identify the same resource (e.g. in case of a JavaScript library it could include the namespace + name of the library + version). All downstream URLs will be assumed to have the same sharing ID if no other unique ID is specified. For resource URLs that have set the `ResourceURL.SHARED` property the portlet may not get called for serving the resource as it may already be cached on the portlet application when serving the same resource for a different portlet.
  URLs of the type `FULL` have the highest cacheability in the browser as they do not depend on any state of the portlet or page.
- PORTLET – The `serveResource` call triggered by a `PORTLET` resource URL does have access to the portlet state consisting of the render parameters, portlet mode and window state. The resource URL does not include further state of the portal page and therefore the markup returned from `serveResource`, or any

further downstream calls resulting from this URL, must only include URLs of type `FULL` or `PORLET`. Creating other URLs, e.g. resource URLs of type `PAGE` or action or render URLs, must result in an `IllegalStateException`[cxxxii]
URLs of the type `PORTLET` are cacheable on the portlet level in the browser and can be served from the browser cache for as long as the state of this portlet does not change.

- PAGE – The resource URL may contain artifacts that require knowledge of the state of the complete page, like `PortletURLs`, or resource URLs of type `PAGE`. The markup returned by such a resource URL may contain any portlet URL. Resource URLs of the type `PAGE` are only cacheable on the page level and can only be served from the browser cache as long as no state on the page changes.

The cacheability constants are ordered (from strong to weak) in the following manner: `FULL, PORTLET, PAGE`.

If no cachability is set on the resource URL, the cacheability setting of the parent resource is used. If no parent resource is available, `PAGE` is the default.

E.g. a portlet creates in render a resource URL with cachability `PORTLET`. When this resource URL is being triggered and the `serveResource` method of the portlet is being called all resource URLs created in this `serveResource` call will have per default `PORTLET` cacheability. The portlet can only further restrict the cacheability, e.g. set it to `FULL`, but not lessen it, like trying to set it to `PAGE`.

PLT.13.8 Generic Portlet Support

The `serveResource` method in the `GenericPortlet` class tries to forward the resource serving to the resource ID set on URL triggering the request for serving the resource. If no resource ID is set, the `serveResource` method does nothing.

PLT.14

Serving Fragments through Portlets

Through the `render` method of the Portlet interface the Portlet produces its complete markup that is embedded as a fragment into the overall page by the portal application. However, there are use cases where the portlet would like to only replace a part of its markup, e.g. via an AJAX call.

1. There are two different scenarios: Perform operations that don't need coordination features or change shared state, like portlet application session scope data, or any navigational state, like render parameters, portlet mode or window state.
2. Perform operations that want to leverage coordination features or need to change shared state like portlet application session scope data, render parameters, portlet mode or window state.

For scenario 1 the Java Portlet Specification provides the `serveResource` method.

Scenario 2 requires coordination between the portlet and the portal application as changing shared state or state that may be stored on the client, like render parameters, affects not only the portlet markup itself, but also other parts of the page. Thus the portal application needs to provide these updates and the portlet needs to have some means to allow the portal performing these updates. Version 2.0 of the Java Portlet Specification does not address this coordinated scenario that requires defining client side interfaces and thus reaches beyond the Java space.

The remainder of this chapter explains how to serve portlet fragments by using the `serveResource` method. In this context a portlet fragment is a response that impacts in most cases only parts of the portlet markup. A fragment response will be commonly in a HTML format but it can also be XML, JSON, etc.

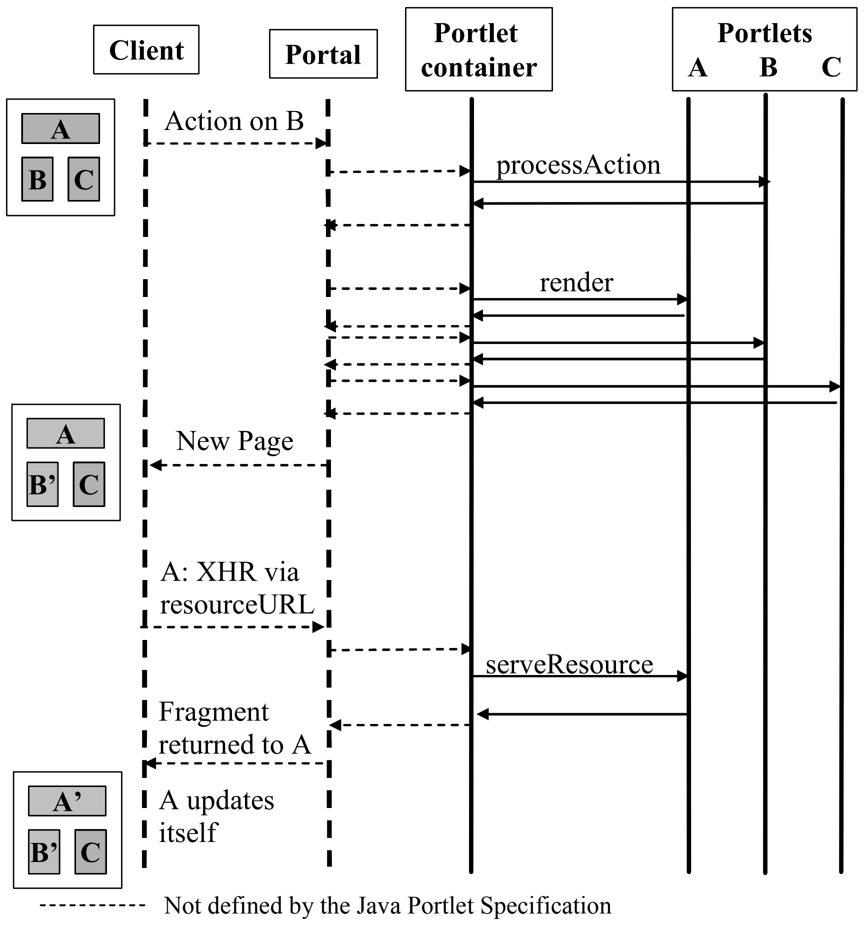
PLT.14.1 Serving Fragments via serveResource Method

Serving fragments via `serveResource` is under the complete control of the portlet. Typically a portlet would issue an XMLHttpRequest with a resource URL and provide either markup or data as response in the `serveResource` method. The ECMA client side code of the portlet is then responsible for inserting either the markup or otherwise update the page DOM in a non-disruptive manner for the other components on the page.

Due to the fact that the portal application is not involved in serving the fragment several restrictions apply for serving fragments via `serveResource`:

- No support for coordination like events or shared render parameters. The portlet will only receive the current shared render parameter values, cannot change these values.
- The `serveResource` call cannot set new render parameters, a new portlet mode or window state.
- The `serveResource` call cannot issue redirects.
- The `serveResource` call should not change application-scoped session state, as other parts of the page will not see these session updates and thus represent an inconsistent user experience.

The portlet should note that such state changes impact cachability of the resource response and set the cache settings accordingly. The following figure shows how a request flow using `serveResource` for serving portlet fragments will look like. Figure 2: Request flow when serving fragments via the `serveResource` method



The top part of the picture shows a normal action request that results in a complete page re-rendering. In portlet A's markup is a resource URL that gets triggered by the user and results in an asynchronous `XMLHttpRequest` to the portlet, which then results in calling the `serveResource` method on portlet A. Portlet A returns a portlet fragment that gets delivered all the way back to the client and is evaluated and processed by some script code of portlet A on the client. This could then result in portlet A updating itself via direct manipulation of the browser DOM.

PLT.15

Coordination between portlets

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

- sharing data between artifacts in the same web application via the session in the application scope (see *PLT.17.2)*
- public render parameters in order to share render state between portlets (see *PLT.11.1.2*)
- portlet events that a portlet can receive and send

In this chapter we'll cover briefly the public render parameters and the portlet events in detail.

Note that it is not in the scope of this specification to define how portlets are wired together, nor how a set of portlets relate to each other or to a portal page. All this is done on portal application level and is not reflected in the Java Portlet API or `portlet.xml`.

PLT.15.1 Public Render Parameters

Public render parameters are intended for sharing view state across portlets. Using public render parameters instead of events avoids the additional process 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 public render parameters are useful is the following: a weather portlet wants to display the weather of a selected city. It therefore uses the public 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 public render parameters, like a map portlet displaying the location of the 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 even bookmark these information for specific cities.

For more details on public render parameters see *PLT.11.1.1.2*.

PLT.15.2 Portlet Events

Portlet events are intended to allow portlets to react to 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 JavaEE 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.

An example where a portlet may want to offer receiving events is for state changes triggered by simple user interactions, e.g. adding an item to a shopping cart. By offering this as an event to other portlets these can trigger adding items to the shopping cart based on the user interactions happing inside these portlets. In contrast to using the portlet application scope session this will work across portlet application boundaries.

PLT.15.2.1 EventPortlet Interface

In order to receive events the portlet must implement the `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 client request.

Events are a lifecycle operation that occurs before the rendering phase. The portlet may issue events via the `setEvent` method during the action processing which will be processed by the portlet container after the action processing has finished. 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, e.g. a portlet mode changed event, or events from other portlets, e.g. an item was added to the shopping cart event.

PLT.15.2.2 Receiving 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 encapsulating the current event name and value. The event must always have a name and may optionally have a value.[cxxxiii]

Event names are represented as QNames in order to make them uniquely identifiable. The event name can be either retrieved with the `getQName` method that returns the complete QName of the event, or with the `getName` method that only returns the local part of the event name.

If the event has a value it must be based on the type defined in the deployment descriptor.[cxxxiv] The default XML to Java mapping that every container should support is the JAXB mapping (see *PLT.27*). Portlet containers are free to support additional mapping mechanisms beyond the JAXB mapping. 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:

```
<default-namespace>http:example.com/events</default-namespace>
<event-definition>
  <name>foo</name>
  <value-type>java.lang.String</value-type>
</event-definition>
....
<portlet>
...
<supported-processing-event>
   <name>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("foo") )
  {
    String payload = (String) event.getValue();
    ...
  }
```

PLT.15.2.3 Sending Events

The portlet can publish events via the `StateAwareResponse.setEvent` method.[cxxxv] The `StateAwareReponse` methods are exposed via the `ActionResponse` and `EventResponse` interfaces. It is also valid to call `StateAwareResponse.setEvent` multiple times in the current `processAction` or `processEvent` method. [cxxxvi]

Events can be published either with their full QName with the `setEvent(QName, Serializable)` or by only specifying their local part with the `setEvent(String, Serializable)` method. If only the local part is specified the namespace must be the default namespace defined in the portlet deployment descriptor with the `default-namespace` element. [cxxxvii] If no such element is provided in the portlet deployment descriptor the XML default namespace `javax.xml.XMLConstants.NULL_NS_URI` must be assumed. [cxxxviii]

The event payload must have a valid JAXB binding, or be in the list of Java primitive types / standard classes of the JAXB 2.0 specification section 8.5.1 or 8.5.2 (except `java.lang.Object`), and implement `java.io.Serializable`. Otherwise the `setEvent` method on the `StateAwareResponse` must throw a `java.lang.IllegalArgumentException`. [cxxxix]

Example for sending an event:

event defined in the DD:

```
<event-definition>
  <qname xmlns:x="http:example.com/events">x:foo.bar</qname>
  <value-type>com.example.Address</value-type>
</event-definition>
....
<portlet>
...
<supported-publishing-event>
 <qname
xmlns:x="http:example.com/events">x:foo.bar</qname></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");
QName name = new QName ("http:example.com/events", "foo.bar");
resp.setEvent(name, sampleAddress);
}
```

PLT.15.2.4 Event declaration

The portlet should declare all events that it would like to receive and the ones it would like to initiate. Typically portlets only receive events that the portlet has declared as processing events.

PLT.15.2.4.1 Declaration in the deployment descriptor

The portlet should declare events in the `portlet.xml` deployment descriptor (see *PLT.24 Deployment Descriptor*). On the application level the portlet should define the basic event definition with the `event-definition` element. The event definition must contain an event name. [cxl] The portlet container must use the event name entry in the portlet deployment descriptor as event name when submitting an event to the portlet. [cxli] The portlet can specify additional alias names in order to enable portals performing an automatic wiring between events. When publishing an event the portlet should also use the event name entry in the deployment descriptor as event name, otherwise the container may ignore this event.

The event definition should be referenced on the portlet level where the portlet can define the processing events with the `supported-processing-event` element and the events being published with the `supported-publishing-event` element. The referenced event name should either be the full QName provided with the `qname` element and referencing the QName of the event definition provided by the `qname` element, or the local part of the QName provided with the `name` element and referencing the local part of the event definition provided by the `name` element.

Event definitions are valid for all 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 QNames as defined in the XML specifications: XML Schema Part2: Datatypes specification (http://www.w3.org/TR/xmlschema-2/#QName), Namespaces in XML (http://www.w3.org/TR/REC-xml-names/#ns-qualnames), Namespaces in XML Errata (http://www.w3.org/XML/xml-names-19990114-errata), TAG Finding: Using Qualified Names (QNames) as Identifiers in Content (http://www.w3.org/2001/tag/doc/qnameids-2002-06-17).

As an alternative the portlet can specify a default namespace via the `default-namespace` element that will be applied to all events defined only with a local name with the `name` element in the event definition section.

The portlet is encouraged to organize the local part of the event names in the `event-definition` element in a hierarchical manner using the dot '.' as separator. A trailing '.' tells the Consumer that this is not the end of the hierarchy and the Portlet is interested in all events with names in this branch of the hierarchy. The portlet must not specify events with the same name but different types. Event names in the `event-definition` element should not end with a trailing "." character as wildcards are not supported in the event definition level. Wildcards should only be used in the `supported-processing-event` or `supported-publishing-event` elements and should be able to be resolved by the portlet container to an event definition without wildcards in the `event-definition` element by matching event names ending with a "." character to any event whose local name starts with the characters before the "." character and also specifies the same namespace. If the wildcard string should match a part of a hierarchy two dots are required at the end of the wildcard string: one to denote the hierarchy and one for the wildcard: "foo.bar..".

A localized display name for the portlet event definition should be provided in the application level resource bundle (see *PLT.25.10*) with an entry of the name `javax.portlet.app.event-definition.<name>.display-name`.

PLT.15.2.4.2 Events not declared in the Deployment Descriptor

The portlet can send events which are not declared in the portlet deployment descriptor at runtime using the `setEvent` method on either the `ActionResponse` or `EventResponse`. [cxlii] The portlet should note that by not declaring these events in the deployment descriptor, the abilities of the portal for distributing the event to other portlets may be limited or even non-existent.

PLT.15.2.5 Event processing

Events are valid only in the current client request and the portlet container must therefore deliver all events within the current client request. [cxliii] 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. However, portal applications should distribute events returned by a single portlet in the order the portlet called the `setEvent` method while executing the `processAction` or `processEvent` method, but ordering of distribution is not guaranteed. Thus portlet developers should rely on other mechanisms, like adding the ordering in the event payload, if ordering of the events is required.

Event distribution is non-blocking and can happen in parallel for different portlet windows.

Event distribution must be serialized for a specific portlet window per client request so that at any given time a portlet window is only processing one event in the `processEvent` method for the current client request. [cxliv] The portlet container should therefore queue the events for one portlet 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.

Note that event processing for different portlets within the current client request may happen in parallel and that therefore for state changes on shared data, like public render parameters or the application session, the last state change wins.

Portlet event processing may occur after the processing of the action, if the portlet was target of an action URL, and must be finished before the render phase. [cxlv]

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

PLT.15.2.6 Exceptions during event processing

A portlet may throw a `PortletException,` a `PortletSecurityException` or a `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. [cxlvi] The portal/portlet-container should continue processing other events targeted to the portlet and 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. [cxlvii] 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.15.2.7 GenericPortlet support

The `GenericPortlet` implements the`EventPortlet` interface and provides a default event handling. For a received event the `GenericPortlet` tries to dispatch to methods annotated with the tag `@ProcessEvent.` The event name can be either specified as QName or local part only.

For using QNames as event name the syntax is the following: `@ProcessEvent (qname=<event name>),` where the event name must be in the format "{" + Namespace URI + "}" + local part (like used by `javax.xml.namespace.QName.toString()` method). If the Namespace URI is equal to the `javax.xml.XMLConstants.NULL_NS_URI` only the local part is used.

For using only the local part of the event name and leverage the default namespace defined in the portlet deployment descriptor with the `default-namespace` element the following alternative is provided: `@ProcessEvent (name=<event name_local_part>),` where the event name is only the local part. If no default namespace is defined in the deployment descriptor the XML default namespace `XMLConstants.NULL_NS_URI` is used.

If the local part of the event name has a wildcard at the end (".") the `GenericPortlet` will try to match the received event either to the same wildcard event name or to the longest matching event name for this wildcard. E.g. if an event with the local part of the event name of "a.b.c.d" is being received and there are methods annotated for handling "a.b." and "a.b.c." events in this portlet, the `GenericPortlet` will dispatch the event to the method annotated with "a.b.c.".

The method annotated with the `@ProcessEvent` annotation must have the following signature:

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

If no such method can be found the `GenericPortlet` just sets the received render parameters as new render parameters. If multiple annotations match the current event it is indeterministic which method will be called for handling this event.

Example:

@ProcessEvent(qname="{http://com.example/events}foo.bar")

public void processFoo(EventRequest request, EventResponse response) throws PortletException, java.io.IOException {

// process event foo.bar

}

PLT.15.3 Predefined Container Events

The Web Service for Remote Portlets (WSRP) specification predefines some common events that should be leveraged when requiring an event for one of the following scenarios:

- Event handling failed (`wsrp:eventHandlingFailed`) –This is a portal application generated event which signals to the portlet that the portal application detected that errors occurred while distributing events. As a simple notification, this event carries no predefined payload, but does use an open content definition.
- Navigations context changed (`wsrp:newNavigationalContextScope`)– allowing the portlet to manage its own navigational context in a consistent manner with the navigational context managed by the portal application.
- New portlet mode (`wsrp:newMode`) – indicating to the portlet that it has been put into a new portlet mode and allowing the portlet to pre-set some state before getting rendered in this new mode.
- New window state(`wsrp:newWindowState`) – indicating to the portlet that it has been put into a new window state and allowing the portlet to pre-set some state before getting rendered in this window state.

See section 5.11 of the Web Services for Remote Portlets specification V2.0 for more details and the QNames for these events.

Portals / portlet containers supporting one of the above predefined events should deliver these events to all portlets having declared receiving event support for these events in the portlet deployment descriptor.

PLT.16

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.16.1 Support for Markup Head Elements

Portals should indicate if they support the `MimeResponse` property `MimeResponse.MARKUP_HEAD_ELEMENT` (value: `"javax.portlet.markup.head.element"`) by providing the `PortalContext.HTML_HEAD_ELEMENT_SUPPORT` (value: `"javax.portlet.markup.head.element.support"`) property on the `PortalContext`.

A non-null value of `MARKUP_HEAD_ELEMENT_SUPPORT` indicates that the portal application supports the `MARKUP_HEAD_ELEMENT` property.

PLT.17

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.17.1 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`, `processEvent`, or `serveResource` invocation.

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

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.[cxlix] 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.[cl] 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.[cli]

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

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

PLT.17.2 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, but enables sharing of preferences and different levels of portlet entities (see Section 5.3.1). 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.17.3 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`).[clvii] 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. [clviii]

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 constrain the modifiability of preferences values.

PLT.17.3.1 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.25.10 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.17.4 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. 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.[clix] 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.[clx] If the validation is successful, the store operation must be completed.[clxi] Portlet preferences should not be modified when they are being validated by a `PreferencesValidator` object.

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.18

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.18.1 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.[clxii] 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.[clxiii]

PLT.18.2 Session Scope

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

Each portlet application has its own distinct `PortletSession` object per user session. Note that the `PortletSession` object is only valid within the current client request and thus should be retrieved via `getPortletSession` for each client request and not stored by the portlet across client requests. The portlet container must not share the `PortletSession` object or the attributes stored in it among different portlet applications or among different user sessions.[clxv]

PLT.18.3 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.[clxvi] 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.

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.[clxvii] 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.[clxviii] `<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.18.4 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. Note that the session objects may be different, but access to objects stored in the application session scope is available to any portlet, servlet or JSPs within the same portlet application.

The container must ensure that all attributes placed in the `PortletSession` are also available in the `HttpSession` of the portlet application. A direct consequence of this is that data stored in the `HttpSession` by servlets or JSPs of the Portlet Application is accessible to portlets through the `PortletSession` in the portlet application scope.[clxix] Conversely, data stored by portlets in the `PortletSession` in the portlet application scope is accessible to servlets and JSPs through the `HttpSession`. [clxx]

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

PLT.18.4.1 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.[clxxiii]
- The attribute name has to conform with the specified prefixing if PORTLET_SCOPE is used.[clxxiv]

- The variant of these methods that does not receive a scope must be treated as PORTLET_SCOPE.[clxxv]

PLT.18.5 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.18.5.1 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. Concurrency issues may occur if the end user interacts at the same time with multiple browser windows with this portlet window or triggers request to the portlet window with a faster rate than the requests get processed.

Setting attributes in the APPLICATION_SCOPE 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.

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

PLT.18.5.2 Rendering phase

The portlet API does not prevent 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 as different portlets share these attributes.

PLT.18.6 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.18.7 Session Timeouts

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

PLT.18.8 Last Accessed Times

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

PLT.18.9 Important Session Semantics

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

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

PLT.19

Dispatching Requests to Servlets and JSPs

Portlets can delegate the execution of logic or creation of content to servlets and JSPs. This is useful for implementing the Model-View-Controller pattern where the portlet may act as controller and dispatch to different JSPs for rendering the views.

The `PortletRequestDispatcher` interface provides a mechanism to accomplish this dispatching.

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

PLT.19.1 Obtaining a PortletRequestDispatcher

`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. [clxvii]

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`.[clxviii]

PLT.19.1.1 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.[clxxix]

PLT.19.2 Using a Request Dispatcher

To include a servlet or a JSP, a portlet calls the `include` method of the `PortletRequestDispatcher` interface. To forward the request processing to a servlet or JSP the portlet calls the `forward` method of the `PortletRequestDispatcher` interface.

The parameters to these methods must be the request and response arguments that were passed in via the corresponding lifecycle method (e.g. `processAction`, `processEvent`, `serveResource`, `render)` , or the request and response arguments must be instances of the corresponding subclasses of the request and response wrapper classes that were introduced for version 2.0 of the specification. [clxxx] In the latter case, the wrapper instances must wrap the request or response objects that the container passed into the lifecycle method.

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.[clxxxi]

PLT.19.3 The Include Method

The include method of the `PortletRequestDispatcher` interface may be called at any time and multiple times within the current portlet lifecycle method. 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 in the `render` method must be handled as HTTP GET requests.[clxxxii]

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.19.3.1 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[clxxxiii]:

```
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.19.3.2 Included Request Attributes

In addition to the request attributes specified in *Servlet Specification, 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

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

Request Attribute	Type
javax.portlet.request	javax.portlet.ActionRequest
javax.portlet.response	javax.portlet.ActionResponse

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

Request Attribute	Type
javax.portlet.request	javax.portlet.EventRequest
javax.portlet.response	javax.portlet.EventResponse

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

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

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

Request Attribute	Type
`javax.portlet.request`	`javax.portlet.ResourceRequest`
`javax.portlet.response`	`javax.portlet.ResourceResponse`

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

PLT.19.3.3 Request and Response Objects for Included Servlets/JSPs from within the Action and Event processing Methods

The target servlet or JSP of the 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 `processAction or processEvent` method in order to keep the action semantic intact.

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

The following methods of the `HttpServletRequest` must return `'0'`: `getRemotePort` and `getLocalPort`.[clxxxvi]

The response of `HttpUtils.getRequestURL` is undefined and should not be used.

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`.[clxxxvii]

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`, `getCookies`.[clxxxviii]

The following methods of the `HttpServletRequest` must be equivalent to the methods of the `PortletRequest` of similar name with the provision defined in *PLT.19.1.1 Query Strings in Request Dispatcher Paths* Section: `getParameter`, `getParameterNames`, `getParameterValues` and `getParameterMap`.[clxxxix]

In case of an include from `processAction`, the following methods of the `HttpServletRequest` must be based on the corresponding methods of the `ActionRequest`: `getCharacterEncoding`, `setCharacterEncoding`, `getContentType`, `getInputStream`, `getContentLength`, `getMethod` and `getReader`.[cxc]

In case of an include from `processEvent`, the following methods of the `HttpServletRequest` must do no operations and/or return `null`: `getCharacterEncoding`, `setCharacterEncoding`, `getContentType`, `getInputStream` and `getReader`.[cxci] The `getContentLength` method of the `HttpServletRequest` must return `0`.[cxcii] The `getMethod` method of the `HTTPServletRequest` must be based on the corresponding method of the `EventRequest`, which must provide the name of the HTTP method with which the original action request was made.[cxciii]

The following methods of the `HttpServletRequest` must be based on the properties provided by the `getProperties` method of the `PortletRequest` interface: `getHeader`, `getHeaders`, `getHeaderNames`, `getDateHeader` and `getIntHeader`.[cxciv]

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

The `getProtocol` method of the `HttpServletRequest` must always return '`HTTP/1.1`'.[cxcvi]

The following methods of the `HttpServletResponse` must return `null`: `encodeRedirectURL`, `encodeRedirectUrl`, `getCharacterEncoding`, `getContentType`, `getLocale`, `resetBuffer`, `reset`.[cxcvii]

The following method of the `HttpServletResponse` must return 0: `getBufferSize`.[cxcviii]

The following methods of the `HttpServletResponse` must return an outputstream / writer that ignores any output written to it: `getOutputStream` and `getWriter`.[cxcix]

The following methods of the `HttpServletResponse` must be equivalent to the methods of the `ActionResponse/EventResponse` of similar name: `encodeURL` and `encodeUrl`.[cc]

The following methods of the `HttpServletResponse` must perform no operations: `setContentType`, `setCharacterEncoding`, `setContentLength`, `setLocale`, `addCookie`, `sendError`, `sendRedirect`, `setDateHeader`, `addDateHeader`, setHeader, addHeader, setIntHeader, addIntHeader, setStatus, setBufferSize and flushBuffer.[cci]

The containsHeader method of the HttpServletResponse must return false.[ccii]

The isCommitted method of the HttpServletResponse must return true.[cciii]

PLT.19.3.4 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: getRemoteAddr, getRemoteHost, getLocalAddress, getLocalName, getRealPath, and getRequestURL.[cciv]

The following methods of the HttpServletRequest must return '0': getRemotePort and getLocalPort.[ccv]

The response of HttpUtils.getRequestURL is undefined and should not be used.

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.[ccvi]

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, getCookies.[ccvii]

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.[ccviii]

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

The following methods of the `HttpServletRequest` must be based on the properties provided by the `getProperties` method of the `PortletRequest` interface: `getHeader`, `getHeaders`, `getHeaderNames`, `getDateHeader` and `getIntHeader`.[ccxi].

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

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

The `getProtocol` method of the `HttpServletRequest` must always return '`HTTP/1.1`'.[ccxiv]

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

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

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

PLT.19.3.5 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`: `getRemoteAddr`, `getRemoteHost`, `getLocalAddress`, `getLocalName`, `getRealPath`, and `getRequestURL`.[ccxx]

The following methods of the `HttpServletRequest` must return `'0'`: `getRemotePort` and `getLocalPort`.[ccxxi]

The response of `HttpUtils.getRequestURL` is undefined and should not be used.

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`.[ccxxii]

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`, `getCookies`.[ccxxiii]

The following methods of the `HttpServletRequest` must be equivalent to the methods of the `ResourceRequest` of similar name: `getCharacterEncoding`, `setCharacterEncoding`, `getContentType`, `getMethod`, `getContentLength` and `getReader`. [ccxxiv] 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`.[ccxxv]

The following methods of the `HttpServletRequest` must be based on the properties provided by the `getProperties` method of the `PortletRequest` interface: `getHeader`, `getHeaders`, `getHeaderNames`, `getDateHeader` and `getIntHeader`.[ccxxvi].

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

The `getProtocol` method of the `HttpServletRequest` must always return '`HTTP/1.1`'.[ccxxviii]

The following methods of the `HttpServletResponse` must return `null`: `encodeRedirectURL` and `encodeRedirectUrl`.[ccxxix]The following methods of the `HttpServletResponse` must be equivalent to the methods of the `ResourceResponse` of similar name: `getCharacterEncoding`, `setBufferSize`, `flushBuffer`, `resetBuffer`, `reset`, `getBufferSize`, `isCommitted`, `getOutputStream`, `getWriter`, `getLocale`, `encodeURL` and `encodeUrl`.[ccxxx]

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

PLT.19.3.6 Comparison of the different Request Dispatcher Includes

HttpServletRequest method	ActionRequest mapping	EventRequest mapping	RenderRequest mapping	ResourceRequest mapping
getAuthType	getAuthType	getAuthType	getAuthType	getAuthType
getContextPath	getContextPath	getContextPath	getContextPath	getContextPath
getCookies	getCookies	getCookies	getCookies	getCookies
getDateHeader	getProperties	getProperties	getProperties	getProperties
getHeader	getProperties	getProperties	getProperties	getProperties
getHeaderNames	getPropertyNames	getPropertyNames	getPropertyNames	getPropertyNames
getHeaders	getProperties	getProperties	getProperties	getProperties
getIntHeader	getProperties	getProperties	getProperties	getProperties
getMethod	getMethod	getMethod	'GET'	getMethod
getPathInfo	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`
getPathTranslated	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`
getQueryString	query string information used to obtain the `PortletRequestDispatcher`	query string information used to obtain the `PortletRequestDispatcher`	query string information used to obtain the `PortletRequestDispatcher`	query string information used to obtain the `PortletRequestDispatcher`
getRemoteUser	getRemoteUser	getRemoteUser	getRemoteUser	getRemoteUser
getRequestedSessionId	getRequestedSessionId	getRequestedSessionId	getRequestedSessionId	getRequestedSessionId
getRequestURI	path and query string information used to obtain the `PortletRequestDispatcher`	path and query string information used to obtain the `PortletRequestDispatcher`	path and query string information used to obtain the `PortletRequestDispatcher`	path and query string information used to obtain the `PortletRequestDispatcher`
getRequestURL	null	null	null	null
getServletPath	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`
getSession	getPortletSes	getPortletSessi	getPortletSes	getPortletSessi

	sion(APPLICATION_SCOPE)	on(APPLICATION_SCOPE)	sion(APPLICATION_SCOPE)	on(APPLICATION_SCOPE)
getUserPrincipal	getUserPrincipal	getUserPrincipal	getUserPrincipal	getUserPrincipal
isRequestedSessionIdFromCookie	N/A	N/A	N/A	N/A
isRequestedSessionIdFromUrl	N/A	N/A	N/A	N/A
isRequestedSessionIdFromURL	N/A	N/A	N/A	N/A
isRequestedSessionIdValid	isRequestedSessionIdValid	isRequestedSessionIdValid	isRequestedSessionIdValid	isRequestedSessionIdValid
isUserInRole	isUserInRole	isUserInRole	isUserInRole	isUserInRole
getAttribute	getAttribute	getAttribute	getAttribute	getAttribute
getAttributeNames	getAttributeNames	getAttributeNames	getAttributeNames	getAttributeNames
getCharacterEncoding	getCharacterEncoding	null	null	getCharacterEncoding
getContentLength	getContentLength	0	0	getContentLength
getContentType	getContentType	null	null	getContentType
getInputStream	getPortletInputStream	null	null	getPortletInputStream
getLocalAddr	null	null	null	null
getLocale	getLocale	getLocale	getLocale	getLocale
getLocales	getLocales	getLocales	getLocales	getLocales
getLocalName	null	null	null	null
getLocalPort	0	0	0	0
getParameter	getParameter	getParameter	getParameter	getParameter
getParameterMap	getParameterMap	getParameterMap	getParameterMap	getParameterMap
getParameterNames	getParameterNames	getParameterNames	getParameterNames	getParameterNames
getParameterValues	getParameterValues	getParameterValues	getParameterValues	getParameterValues
getProtocol	HTTP/1.1	HTTP/1.1	HTTP/1.1	HTTP/1.1
getReader	getReader	null	null	getReader
getRealPath	null	null	null	null
getRemoteAddr	null	null	null	null
getRemoteHost	null	null	null	null
getRemotePort	0	0	0	0
getRequestDispatcher	N/A	N/A	N/A	N/A
getScheme	getScheme	getScheme	getScheme	getScheme
getServerName	getServerName	getServerName	getServerName	getServerName
getServerPort	getServerPort	getServerPort	getServerPort	getServerPort
isSecure	isSecure	isSecure	isSecure	isSecure
removeAttribute	removeAttribute	removeAttribute	removeAttribute	removeAttribute
setAttribute	setAttribute	setAttribute	setAttribute	setAttribute

setCharacterEncoding	setCharacterEncoding	no-op	no-op	setCharacterEncoding

Note: no-op indicates that this method does not perform any operation and N/A indicates that such a method is not available in the portlet interface and the functionality defined by the Servlet Specification must be provided for this call.

5

HttpServletResponse method	ActionResponse mapping	EventResponse mapping	RenderResponse mapping	ResourceResponse mapping
addCookie	no-op	no-op	no-op	no-op
addDateHeader	no-op	no-op	no-op	no-op
addHeader	no-op	no-op	no-op	no-op
addIntHeader	no-op	no-op	no-op	no-op
containsHeader	false	false	false	false
encodeRedirectUrl	null	null	null	null
encodeRedirectURL	null	null	null	null
encodeUrl	encodeURL	encodeURL	encodeURL	encodeURL
encodeURL	encodeURL	encodeURL	encodeURL	encodeURL
sendError	no-op	no-op	no-op	no-op
sendRedirect	no-op	no-op	no-op	no-op
setDateHeader	no-op	no-op	no-op	no-op
setHeader	no-op	no-op	no-op	no-op
setIntHeader	no-op	no-op	no-op	no-op
setStatus	no-op	no-op	no-op	no-op
flushBuffer	no-op	no-op	flushBuffer	flushBuffer
getBufferSize	0	0	getBufferSize	getBufferSize
getCharacterEncoding	null	null	getCharacterEncoding	getCharacterEncoding
getContentType	null	null	getContentType	getContentType
getLocale	null	null	getLocale	getLocale
getOutputStream	null stream	null stream	getPortletOutputStream	getPortletOutputStream
getWriter	null writer	null writer	getWriter	getWriter
isCommitted	true	true	isCommitted	isCommitted
reset	no-op	no-op	reset	reset
resetBuffer	no-op	no-op	resetBuffer	resetBuffer
setBufferSize	no-op	no-op	setBufferSize	setBufferSize
setCharacterEncoding	no-op	no-op	no-op	no-op

setContentLength	no-op	no-op	no-op	no-op
setContentType	no-op	no-op	no-op	no-op
setLocale	no-op	no-op	no-op	no-op

PLT.19.3.7 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.[ccxxxiii] 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.[ccxxxiv]

PLT.19.3.8 Path and Query Information in Included / Forwarded Servlets

As mentioned in the previous sections the methods of the `HttpServletRequest` of an included servlet that deal with path and query information (`getPathInfo`, `getPathTranslated`, `getQueryString`, `getRequestURI` and `getServletPath`) must return the path and query string information used to obtain the `PortletRequestDispatcher` object. This is different than in the Servlet API, where these values are based on the path and query string of the client request. This makes sense from the servlet programming model point of view where you want to run the included / forwarded code as if it really where running in the servlet issuing the request dispatcher include or forward call.

On the other hand, the portlet does not have direct access to the path and query information of the client request as it is one component rendered on the page. Thus the portlet acts as starting point of the include chain and the included / forwarded servlet must get the path and query string information used to obtain the `PortletRequestDispatcher` object. [ccxxxv] Note that when doing additional includes or forwards from the included or forwarded servlet it will have the same semantics as in the plain servlet case: all further included / forwarded servlets or JSPs will get the path and query string information used to obtain the `PortletRequestDispatcher` object as this is viewed as the initial path and query information.

PLT.19.4 The forward Method

The `forward` method of the `RequestDispatcher` interface may be called by the calling portlet only when no output has been committed to the response. The request dispatcher `forward` allows setting the response content type by the servlet or JSP the forward call is made to. If output data exists in the response buffer that has not been committed, the content must be cleared before the target servlet's service method is called. [ccxxxvi] If the response has been committed, an `IllegalStateException` must be thrown. [ccxxxvii]

Information like cookies, properties, portlet mode, window state, render parameters, or the portlet title that the portlet may have set before calling the request dispatcher forward method should still be valid.

The path elements of the request object exposed to the target servlet must reflect the path used to obtain the RequestDispatcher.

Before the forward method of the RequestDispatcher interface returns, the response content must be sent and committed, and closed by the portlet container. [ccxxxviii]

When using a RequestDispatcher in a servlet that was target of a forward from a portlet, the servlet must request the RequestDispatcher via the ServletRequest and not the ServletContext. Using a RequestDispatcher that was retrieved via the ServletContext may behave in a way that does not comply with this specification.

PLT.19.4.1 Query String

The request dispatching mechanism is responsible for aggregating query string parameters when forwarding or including requests.

PLT.19.4.2 Forwarded Request Parameters

Except for servlets obtained by using the `getNamedDispatcher` method, a servlet that has been invoked by a portlet using the `forward` method of `RequestDispatcher` has access to the path used to obtain the `PortletRequestDispatcher.`

The following request attributes must be set: [ccxxxix]

```
javax.servlet.forward.request_uri

javax.servlet.forward.context_path

javax.servlet.forward.servlet_path

javax.servlet.forward.path_info

javax.servlet.forward.query_string
```

The values of these attributes must be equal to the return values of the `HttpServletRequest` methods `getRequestURI, getContextPath, getServletPath, getPathInfo, getQueryString` respectively, invoked on the request object passed to the first servlet object in the forward call chain. [ccxl]

These attributes are accessible from the forwarded servlet via the `getAttribute` method on the request object. Note that these attributes must always reflect the information in the target of the first forward servlet in the situation that multiple forwards and subsequent includes are called. [ccxli]

If the forwarded servlet was obtained by using the `getNamedDispatcher` method, these attributes must not be set. [ccxlii]

PLT.19.4.3 Request and Response Objects for Forwarded Servlets/JSPs from within the Action and Event processing Methods

The target servlet of the portlet request dispatcher has access to a limited set of methods of the request and the response objects when the forward is done from within the `processAction or processEvent` method in order to keep the action semantic intact.

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

The following methods of the `HttpServletRequest` must return `'0'`: `getRemotePort` and `getLocalPort`.[ccxliv]

The response of `HttpUtils.getRequestURL` is undefined and should not be used.

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`.[ccxlv]

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`, `getCookies`.[ccxlvi]

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`.[ccxlvii]

In case of a forward from `processAction`, the following methods of the `HttpServletRequest` must be based on the corresponding methods of the `ActionRequest`: `getCharacterEncoding`, `setCharacterEncoding`, `getContentType`, `getInputStream`, `getContentLength`, `getMethod` and `getReader`.[ccxlviii]

In case of a forward from `processEvent`, the following methods of the `HttpServletRequest` must do no operations and/or return `null`: `getCharacterEncoding`, `setCharacterEncoding`, `getContentType`, `getInputStream` and `getReader`.[ccxlix] The `getContentLength` method of the `HttpServletRequest` must return `0`.[ccl] The `getMethod` method of the `HttpServletRequest` must be based on the corresponding method of the `ActionRequest` triggering this event.[ccli]

The following methods of the `HttpServletRequest` must be based on the properties provided by the `getProperties` method of the `PortletRequest` interface: `getHeader`, `getHeaders`, `getHeaderNames`, `getDateHeader` and `getIntHeader`.[cclii].

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

The `getProtocol` method of the `HttpServletRequest` must always return '`HTTP/1.1`'.[ccliv]

The following methods of the `HttpServletResponse` must return `null`: `encodeRedirectURL`, `encodeRedirectUrl`, `getCharacterEncoding`, `getContentType`, `getLocale`, and `getBufferSize`.[cclv]

The following methods of the `HttpServletResponse` must return an outputstream / writer that ignores any output written to it: `getOutputStream` and `getWriter`.[cclvi]

The following methods of the `HttpServletResponse` must be equivalent to the methods of the `ActionResponse/EventResponse` of similar name: `encodeURL` and `encodeUrl`.[cclvii]

The following methods of the `HttpServletResponse` must perform no operations: `resetBuffer`, `reset`, `setContentType`, `setContentLength`, `setCharacterEncoding`, `setLocale`, `sendError`, `setDateHeader`, `addDateHeader`, `setHeader`, `addHeader`, `setIntHeader`, `addIntHeader`, `setStatus`, `setBufferSize` and `flushBuffer`.[cclviii]

The `sendRedirect` method of the `HttpServletResponse` must be mapped to the `ActionResponse.sendRedirect` in the `processAction` call and to a no-op for the `processEvent` call.

The `addCookie` method of the `HttpServletResponse` must be based on `addProperty` method of the `ActionResponse/EventResponse` interface. [cclix]

The `containsHeader` method of the `HttpServletResponse` must return `false`.[cclx]

The `isCommitted` method of the `HttpServletResponse` must return `false`.[cclxi]

PLT.19.4.4 Request and Response Objects for Forwarded 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 forward is done from within the `render` method.

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

The following methods of the `HttpServletRequest` must return `'0'`: `getRemotePort` and `getLocalPort`.[cclxiii]

The response of `HttpUtils.getRequestURL` is undefined and should not be used.

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`.[cclxiv]

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`, `getCookies`.[cclxv]

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`.[cclxvi]

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

The following methods of the `HttpServletRequest` must be based on the properties provided by the `getProperties` method of the `PortletRequest` interface: `getHeader`, `getHeaders`, `getHeaderNames`, `getDateHeader` and `getIntHeader`.[cclxix].

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

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

The `getProtocol` method of the `HttpServletRequest` must always return '`HTTP/1.1`'.[cclxxii]

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

The following methods of the `HttpServletResponse` must perform no operations: `setContentLength`, `setLocale`, `sendError`, `sendRedirect`, and `setStatus`.[cclxxv] The `containsHeader` method of the `HttpServletResponse` must return `false`.[cclxxvi]

The following methods of the `HttpServletResponse` must be based on the properties provided by the `setProperties/addProperties` method of the `RenderResponse` interface: `addCookie`, `setDateHeader`, `addDateHeader`, `setHeader`, `addHeader`, `setIntHeader`, `addIntHeader`.[cclxxvii]

PLT.19.4.5 Request and Response Objects for Forwarded 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`: `getRemoteAddr`, `getRemoteHost`, `getLocalAddress`, `getLocalName`, `getRealPath`, and `getRequestURL`.[cclxxviii]

The following methods of the `HttpServletRequest` must return '`0`': `getRemotePort` and `getLocalPort`.[cclxxix]

The response of `HttpUtils.getRequestURL` is undefined and should not be used.

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`.[cclxxx]

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, getCookies.[cclxxxi]

The following methods of the HttpServletRequest must be equivalent to the methods of the ResourceRequest of similar name: getCharacterEncoding, setCharacterEncoding, getContentType, getMethod and getReader. [cclxxxii] 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.[cclxxxiii]

The following methods of the HttpServletRequest must be based on the properties provided by the getProperties method of the PortletRequest interface: getHeader, getHeaders, getHeaderNames, getDateHeader and getIntHeader.[cclxxxiv].

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

The getProtocol method of the HttpServletRequest must always return 'HTTP/1.1'. [cclxxxvi]

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

The following methods of the HttpServletResponse must be equivalent to the method defined in the Servlet Specification for HttpServletResponse: setContentLength, setCharacterEncoding, and setLocale.

The following methods of the HttpServletRequest must be based on the properties provided by the setProperties/addProperties method of the ResourceResponse interface: addCookie, setDateHeader, addDateHeader, setHeader, addHeader, setIntHeader, addIntHeader.

If the portlet want to set a response status code it should do this via setProperty with the key ResourceResponse.HTTP_STATUS_CODE.

The following methods of the `HttpServletResponse` must perform no operations: `sendError`, `sendRedirect`.[cclxxxix] The `containsHeader` method of the `HttpServletResponse` must return `false`.[ccxc]

PLT.19.4.6 Comparison of the different Request Dispatcher Forwards

HttpServletRequest method	ActionRequest mapping	EventRequest mapping	RenderRequest mapping	ResourceRequest mapping
getAuthType	getAuthType	getAuthType	getAuthType	getAuthType
getContextPath	getContextPath	getContextPath	getContextPath	getContextPath
getCookies	getCookies	getCookies	getCookies	getCookies
getDateHeader	getProperties	getProperties	getProperties	getProperties
getHeader	getProperties	getProperties	getProperties	getProperties
getHeaderNames	getPropertyNames	getPropertyNames	getPropertyNames	getPropertyNames
getHeaders	getProperties	getProperties	getProperties	getProperties
getIntHeader	getProperties	getProperties	getProperties	getProperties
getMethod	getMethod	getMethod of ActionRequest	'GET'	getMethod
getPathInfo	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`
getPathTranslated	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`	path used to obtain the `PortletRequestDispatcher`
getQueryString	query string information used to obtain the `PortletRequestDispatcher`	query string information used to obtain the `PortletRequestDispatcher`	query string information used to obtain the `PortletRequestDispatcher`	query string information used to obtain the `PortletRequestDispatcher`
getRemoteUser	getRemoteUser	getRemoteUser	getRemoteUser	getRemoteUser
getRequestedSessionId	getRequestedSessionId	getRequestedSessionId	getRequestedSessionId	getRequestedSessionId
getRequestURI	path and query string information used to obtain the `PortletRequ`	path and query string information used to obtain the `PortletRequ`	path and query string information used to obtain the `PortletRequ`	path and query string information used to obtain the `PortletRequ`

	estDispatcher	estDispatcher	estDispatcher	estDispatcher
getRequestURL	null	null	null	null
getServletPath	path used to obtain the PortletRequestDispatcher	path used to obtain the PortletRequestDispatcher	path used to obtain the PortletRequestDispatcher	path used to obtain the PortletRequestDispatcher
getSession	getPortletSession(APPLICATION_SCOPE)	getPortletSession(APPLICATION_SCOPE)	getPortletSession(APPLICATION_SCOPE)	getPortletSession(APPLICATION_SCOPE)
getUserPrincipal	getUserPrincipal	getUserPrincipal	getUserPrincipal	getUserPrincipal
isRequestedSessionIdFromCookie	N/A	N/A	N/A	N/A
isRequestedSessionIdFromUrl	N/A	N/A	N/A	N/A
isRequestedSessionIdFromURL	N/A	N/A	N/A	N/A
isRequestedSessionIdValid	isRequestedSessionIdValid	isRequestedSessionIdValid	isRequestedSessionIdValid	isRequestedSessionIdValid
isUserInRole	isUserInRole	isUserInRole	isUserInRole	isUserInRole
getAttribute	getAttribute	getAttribute	getAttribute	getAttribute
getAttributeNames	getAttributeNames	getAttributeNames	getAttributeNames	getAttributeNames
getCharacterEncoding	getCharacterEncoding	null	null	getCharacterEncoding
getContentLength	getContentLength	0	0	getContentLength
getContentType	getContentType	null	null	getContentType
getInputStream	getPortletInputStream	null	null	getPortletInputStream
getLocalAddr	null	null	null	null
getLocale	getLocale	getLocale	getLocale	getLocale
getLocales	getLocales	getLocales	getLocales	getLocales
getLocalName	null	null	null	null
getLocalPort	0	0	0	0
getParameter	getParameter	getParameter	getParameter	getParameter
getParameterMap	getParameterMap	getParameterMap	getParameterMap	getParameterMap
getParameterNames	getParameterNames	getParameterNames	getParameterNames	getParameterNames
getParameterValues	getParameterValues	getParameterValues	getParameterValues	getParameterValues
getProtocol	HTTP/1.1	HTTP/1.1	HTTP/1.1	HTTP/1.1
getReader	getReader	null	null	getReader

getRealPath	null	null	null	null
getRemoteAddr	null	null	null	null
getRemoteHost	null	null	null	null
getRemotePort	0	0	0	0
getRequestDispatcher	N/A	N/A	N/A	N/A
getScheme	getScheme	getScheme	getScheme	getScheme
getServerName	getServerName	getServerName	getServerName	getServerName
getServerPort	getServerPort	getServerPort	getServerPort	getServerPort
isSecure	isSecure	isSecure	isSecure	isSecure
removeAttribute	removeAttribute	removeAttribute	removeAttribute	removeAttribute
setAttribute	setAttribute	setAttribute	setAttribute	setAttribute
setCharacterEncoding	setCharacterEncoding	no-op	no-op	setCharacterEncoding

Note: no-op indicates that this method does not perform any operation and N/A indicates that such a method is not available in the portlet interface and the functionality defined by the Servlet Specification must be provided for this call.

5

HttpServletResponse method	ActionResponse mapping	EventResponse mapping	RenderResponse mapping	ResourceResponse mapping
addCookie	addProperty	addProperty	addProperty	addProperty
addDateHeader	no-op	no-op	addProperties	addProperties
addHeader	no-op	no-op	addProperties	addProperties
addIntHeader	no-op	no-op	addProperties	addProperties
containsHeader	false	false	false	false
encodeRedirectUrl	null	null	null	null
encodeRedirectURL	null	null	null	null
encodeUrl	encodeURL	encodeURL	encodeURL	encodeURL
encodeURL	encodeURL	encodeURL	encodeURL	encodeURL
sendError	no-op	no-op	no-op	no-op
sendRedirect	sendRedirect	no-op	no-op	no-op
setDateHead	no-op	no-op	setProperties	setProperties

er				
setHeader	no-op	no-op	setProperties	setProperties
setIntHeader	no-op	no-op	setProperties	setProperties
setStatus	no-op	no-op	no-op	setProperties
flushBuffer	no-op	no-op	flushBuffer	flushBuffer
getBufferSize	null	null	getBufferSize	getBufferSize
getCharacterEncoding	null	null	getCharacterEncoding	getCharacterEncoding
getContentType	null	null	getContentType	getContentType
getLocale	null	null	getLocale	getLocale
getOutputStream	null stream	null stream	getPortletOutputStream	getPortletOutputStream
getWriter	null writer	null writer	getWriter	getWriter
isCommitted	false	false	isCommitted	isCommitted
reset	no-op	no-op	reset	reset
resetBuffer	no-op	no-op	resetBuffer	resetBuffer
setBufferSize	no-op	no-op	setBufferSize	setBufferSize
setCharacterEncoding	no-op	no-op	no-op	setCharacterEncoding
setContentLength	no-op	no-op	no-op	setContentLength
setContentType	no-op	no-op	setContentType	setContentType
setLocale	no-op	no-op	no-op	setLocale

PLT.19.5 Servlet filters and Request Dispatching

Since the Java Servlet Specification V2.4 you can specify servlet filters for request dispatcher include calls. Portlet containers must support this capability for included servlets via the `PortletRequestDispatcher`. [ccxci] 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.19.6 Changing the Default Behavior for Included / Forwarded Session Scope

The default for the session variable named "session" of included / forwarded servlets or JSPs is that it maps to the portlet session with application scope. Some portlets may require that the session variable for included / forwarded servlets or JSPs maps instead to the portlet session scope in order to work correctly. These portlets can indicate this via setting the `container-runtime-option` `javax.portlet.servletDefaultSessionScope` to `PORTLET_SCOPE`. The default for `javax.portlet.servletDefaultSessionScope` is `APPLICATION_SCOPE`.

Example:

```
<portlet>
…
      <container-runtime-option>
            <name>javax.portlet.servletDefaultSessionScope</name>
            <value>PORTLET_SCOPE</value>
      </container-runtime-option>
</portlet>
```

Portlet developers should note that not all portlet container may be able to provide this feature as a portable JavaEE solution does not currently exist. Therefore, relying on this feature may restrict the numbers of portlet containers the portlet can be executed on.

PLT.20

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.20.1 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 JavaEE.

PLT.20.2 Main Concepts

The main concepts of this filtering model are described in this section. The application developer creates a filter by implementing one of the `javax.portlet.filter.XYZFilter` interfaces 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.20.2.1 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[ccxcii]. 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[ccxciii]. 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[ccxciv]:

- `ActionRequest` and `ActionResponse` for `processAction` calls
- `EventRequest` and `EventResponse` for `processEvent` calls
- `RenderRequest` and `RenderResponse` for `render` calls
- `ResourceRequest` and `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, ResourceResponse`) 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 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. [ccxcv]

PLT.20.2.2 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 has the ability to override existing methods on the request and response objects. The portlet should not add additional methods to the wrapper as further downstream wrappers may not honor these. 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. [ccxcvi]

PLT.20.2.3 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.20.2.4 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
- `lifecycle`: used to determine for which lifecycles the filter should be applied
- `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[ccxcvii]. 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>

      <lifecycle>ACTION_PHASE</lifecycle>

</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. Each filter mapping matching the portlet should be applied for this portlet, even if that result in one filter being applied more than once.
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[ccxcviii]. 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 within the 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.[ccxcix].

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.20.2.5 Defining the Target Lifecycle Method for a Portlet Filter

A portlet filter can be applied to different lifecycle method calls: `processAction, processEvent, render, serveResource`[ccc]. Thus the filter must define the lifecycle method for which the filter is written in the `<lifecycle>` element of the `<filter>` element. [ccci] A filter can be applied to one or more lifecycle methods. The following constants are valid values for the `<lifecycle>` element:

- `ACTION_PHASE` requesting that the portlet container processes this filter for the `processAction` lifecycle method. The filter implementation must implement the `ActionFilter` interface.
- `EVENT_PHASE` requesting that the portlet container processes this filter for the `processEvent` lifecycle method. The filter implementation must implement the `EventFilter` interface.
- `RENDER_PHASE` requesting that the portlet container processes this filter for the `render` lifecycle method. The filter implementation must implement the `EventFilter` interface.
- `RESOURCE_PHASE` requesting that the portlet container processes this filter for the `serveResource` lifecycle method. The filter implementation must implement the `ResourceFilter` interface.

If the lifecycle declaration and portlet filter type do not match the portlet container is free to either reject the portlet at deployment time or ignore this filter.

Example:

```
<filter>
      <filter-name>Sample Filter</filter-name>
      <filter-class>com.acme.SampleFilter</filter-class>
      <lifecycle>ACTION_PHASE</lifecycle>
      <lifecycle>RENDER_PHASE</lifecycle>
</filter>
```

In this example the portlet filter is applied to the action and render phase.

PLT.21

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.21.1 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 should not be exposed to portlets.

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

PLT.21.2 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`.[cccii]. 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.

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(PortletRequest.P3PUserInfos.USER_NAME_GIVEN) : "";
String lastName  = (userInfo!=null)
      ? (String)
userInfo.get(PortletRequest.P3PUserInfos.USER_NAME_FAMILY) : "";
...
```

PLT.21.3 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.22

Caching

Caching content helps improving the Portal response time for users. It also helps reducing the load on servers.

The Portlet Specification defines an expiration based caching mechanism. This caching mechanism is per portlet. Cached content must not be shared across different user clients displaying the same portlet for the private cache scope.

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.22.1 Expiration Cache

Portlets that want their content to be cached using expiration cache 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) and must not be shared across users.

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

A portlet may programmatically alter the expiration time or caching scope by setting a property in the `RenderResponse` or `ResourceResponse` object using the `EXPIRATION_CACHE` or `CACHE_SCOPE` constant defined in the `MimeResponse` interface in forwarded or included servlets/JSPs. Inside the portlet the CacheControl object is available via the `MimeResponse` for setting the expiration time or caching scope via the calls `setExpirationTime` or `setScope`.

The portlet should set the expiration time or caching scope before writing to the output stream as otherwise portals / portlet containers may ignore the values.

If the expiration property is set to 0, 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 should be used. If the caching scope is set to `PRIVATE_SCOPE` the cached data must not be shared across users. If the caching scope is set to `PUBLIC_SCOPE` the cached data may be shared across users. The private scope is the default scope if no scope is provided in the deployment descriptor or via the `RenderResponse` or `ResourceResponse`.

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 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 the portlet is target of request with an action-type semantic (e.g. an action or event call), the portlet container should discard the cache and invoke the corresponding request handling methods of the portlet like `processAction`,or `processEvent`.

PLT.22.2 Validation Cache

As an extension of the expiration-based caching mechanism portlets may use validation caching. Validation-based caching allows portlets to return a validation token together with the markup response and expiration time. The portlet can set the validation token on the `RenderResponse` or `ResourceResponse` via the `ETAG` property from within servlets/JSPs or via the `CacheControl setETag` method from within the portlet. If no expiration time is set, the content should be viewed by the portlet container as expired.

After the content is expired the portlet container should send a `render` or `serveResource` request to the portlet with the validation token (called ETag in HTTP) of the expired content. The portlet can access the validation token provided by the portlet container either via the property `ETAG` of the `RenderRequest` or `ResourceRequest`, or the `getETag` method of the `RenderRequest` or `ResourceRequest`. The portlet can validate if the cached content for the given ETag is still valid or not. If the content is still valid the portlet should not render any output but either set the property `USE_CACHED_CONTENT RenderResponse` or `ResourceResponse` and a new expiry time, or `setUseCachedContent` on the `CacheControl` of the `RenderResponse` or `ResourceResponse` and set a new expiry time. The portlet should set the validation token, expiry time or caching scope before writing to the output stream as otherwise portals / portlet containers may ignore the values.

Example:

```
protected void doView (RenderRequest request, RenderResponse response)
    throws PortletException, java.io.IOException
 {
     …
    if ( request.getETag() != null ) {  // validation request
      if ( markupIsStillValid(request.getETag()) ) {
            // markup is still valid
            response.getCacheControl().setExpirationTime(30);
            response.getCacheControl().setUseCachedContent(true);
            return;
      }
    }
    // create new content with new validation tag
    response.getCacheControl().setETag(someID);
    response.getCacheControl().setExpirationTime(60);
    PortletRequestDispatcher                    rd                    =
getPortletContext().getPortletRequestDispatcher("jsp/view.jsp");
    rd.include(request, response);
}
```

PLT.23

Portlet Applications

A portlet application is a web application, as defined in *Servlet Specification, 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.23.1 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.23.2 Relationship to PortletContext

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

PLT.23.3 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^{TM} 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.23.4 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.23.5 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.[cccv]

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

PLT.23.6 Portlet Application Archive File

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

PLT.23.7 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.23.8 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.23.9 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.23.10 Portlet Application Environment

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

PLT.24

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.24.1 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, SRV.12.1* Section.

PLT.24.2 Roles

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

PLT.24.3 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.[cccvii] Refer to the *Servlet Specification, 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.[cccviii]

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</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.[cccix] 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.24.4 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 confidential 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.24.5 Propagation of Security Identity in EJB^{TM} 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^{TM} 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^{TM} 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.[cccx] The security role name must be one of the security role names defined for the `web.xml` deployment descriptor.[cccxi] Alternatively, portlet application code may be the sole processor of the signon into the EJB^{TM} container.

PLT.25

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, 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.25.1 Portlet and Web Application Deployment Descriptor

In the Portlet Specification 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,* 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 can be set in the `web.xml` deployment descriptor:

- portlet application description using the `<description>` element
- portlet application name using the `<display-name>` element
- portlet application security role mapping using the `<security-role>` element
- portlet application locale-character set mapping for serving resources using the `<locale-encoding-mapping-list>`.

PLT.25.2 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, SRV.9 Web Application* Chapter.

In addition to the resources described in the *Servlet Specification, 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.25.2.1 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.25.2.2 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.25.3 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.25.4 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.25.5 Portlet Deployment Descriptor

Portlet deployment descriptor schema:

```
<?xml version="1.0" encoding="UTF-8"?>

<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>
    This is the XML Schema for the Portlet 2.0 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. In order to be a valid JSR 286 portlet application the version
      must have the value "2.0".
      </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>
    <unique name="filter-name-uniqueness">
      <annotation>
        <documentation>
        The filter element contains the name of a filter.
        The name must be unique within the portlet application.
        </documentation>
      </annotation>
      <selector xpath="portlet:filter"/>
      <field xpath="portlet:filter-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"/>
       <element name="resource-bundle" type="portlet:resource-bundleType"
        minOccurs="0"/>
       <element name="filter" type="portlet:filterType" minOccurs="0"
        maxOccurs="unbounded"/>
       <element name="filter-mapping" type="portlet:filter-mappingType"
        minOccurs="0" maxOccurs="unbounded"/>
       <element name="default-namespace" type="xs:anyURI" minOccurs="0"/>
       <element name="event-definition" type="portlet:event-definitionType"
        minOccurs="0" maxOccurs="unbounded"/>
       <element name="public-render-parameter"
        type="portlet:public-render-parameterType" minOccurs="0"
        maxOccurs="unbounded"/>
       <element name="listener" type="portlet:listenerType" minOccurs="0"
        maxOccurs="unbounded"/>
       <element name="container-runtime-option"
        type="portlet:container-runtime-optionType" minOccurs="0"
        maxOccurs="unbounded"/>
     </sequence>
     <attribute name="version" type="portlet:string" use="required"/>
     <attribute name="id" type="portlet:string" use="optional"/>
   </complexType>
   <complexType name="cache-scopeType">
     <annotation>
```

```
      <documentation>
      Caching scope, allowed values are "private" indicating that the content
      should not be shared across users and "public" indicating that the
      content may be shared across users.
      The default value if not present is "private".
      Used in: portlet
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet:string"/>
    </simpleContent>
  </complexType>
  <complexType name="custom-portlet-modeType">
    <annotation>
      <documentation>
      A custom portlet mode that one or more portlets in
      this portlet application supports.
      If the portal does not need to provide some management functionality
      for this portlet mode, the portal-managed element needs to be set
      to "false", otherwise to "true". Default is "true".
      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"/>
      <element name="portal-managed" type="portlet:portal-managedType"
       minOccurs="0"/>
    </sequence>
    <attribute name="id" type="portlet: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="portlet:string" use="optional"/>
  </complexType>
  <complexType name="expiration-cacheType">
    <annotation>
      <documentation>
      Expiration-time defines 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="portlet: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="portlet: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="portlet: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="portlet: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="cache-scope" type="portlet:cache-scopeType"
       minOccurs="0"/>
      <element name="supports" type="portlet:supportsType"
       maxOccurs="unbounded"/>
      <element name="supported-locale" type="portlet:supported-localeType"
```

```
            minOccurs="0" maxOccurs="unbounded"/>
          <element name="resource-bundle" type="portlet:resource-bundleType"
            minOccurs="0"/>
          <element name="portlet-info" type="portlet:portlet-infoType"
            minOccurs="0"/>
          <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:event-definition-referenceType" minOccurs="0"
            maxOccurs="unbounded"/>
          <element name="supported-publishing-event"
            type="portlet:event-definition-referenceType" minOccurs="0"
            maxOccurs="unbounded"/>
          <element name="supported-public-render-parameter" type="portlet:string"
            minOccurs="0" maxOccurs="unbounded"/>
          <element name="container-runtime-option"
            type="portlet:container-runtime-optionType" minOccurs="0"
            maxOccurs="unbounded"/>
        </sequence>
        <attribute name="id" type="portlet: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="container-runtime-optionType">
   <annotation>
     <documentation>
      The container-runtime-option element contains settings
      for the portlet container that the portlet expects to be honored
      at runtime. These settings may re-define default portlet container
      behavior, like the javax.portlet.escapeXml setting that disables
      XML encoding of URLs produced by the portlet tag library as
      default.
      Names with the javax.portlet prefix are reserved for the Java
      Portlet Specification.
     Used in: portlet-app, portlet
     </documentation>
   </annotation>
   <sequence>
     <element name="name" type="portlet:nameType"/>
     <element name="value" type="portlet:valueType" minOccurs="0"
      maxOccurs="unbounded"/>
   </sequence>
 </complexType>
 <complexType name="filter-mappingType">
```

```
  <annotation>
    <documentation>
    Declaration of the filter mappings in this portlet
    application is done by using filter-mappingType.
    The container uses the filter-mapping
    declarations to decide which filters to apply to a request,
    and in what order. To determine which filters to
    apply it matches filter-mapping declarations on the
    portlet-name and the lifecyle phase defined in the
    filter element. The order in which filters are invoked
    is the order in which filter-mapping declarations
    that match appear in the list of filter-mapping elements.
    Used in: portlet-app
    </documentation>
  </annotation>
  <sequence>
    <element name="filter-name" type="portlet:filter-nameType"/>
    <element name="portlet-name" type="portlet:portlet-nameType"
     maxOccurs="unbounded"/>
  </sequence>
</complexType>
<complexType name="filterType">
  <annotation>
    <documentation>
      The filter element specifies a filter that can transform the
      content of portlet requests and portlet responses.
      Filters can access the initialization parameters declared in
```

```
        the deployment descriptor at runtime via the FilterConfig
        interface.
        A filter can be restricted to one or more lifecycle phases
        of the portlet. Valid entries for lifecycle are:
        ACTION_PHASE, EVENT_PHASE, RENDER_PHASE,
        RESOURCE_PHASE
        Used in: portlet-app
        </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
       maxOccurs="unbounded"/>
      <element name="display-name" type="portlet:display-nameType"
       minOccurs="0" maxOccurs="unbounded"/>
      <element name="filter-name" type="portlet:filter-nameType"/>
      <element name="filter-class" type="portlet:fully-qualified-classType"/>
      <element name="lifecycle" type="portlet:string" maxOccurs="unbounded"/>
      <element name="init-param" type="portlet:init-paramType" minOccurs="0"
       maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="portlet-collectionType">
    <annotation>
      <documentation>
      The portlet-collectionType is used to identify a subset
      of portlets within a portlet application to which a
      security constraint applies.
      Used in: security-constraint
      </documentation>
    </annotation>
    <sequence>
      <element name="portlet-name" type="portlet:portlet-nameType"
       maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <complexType name="event-definitionType">
    <annotation>
      <documentation>
      The event-definitionType is used to declare events the portlet can either
      receive or emit.
      The name must be unique and must be the one the
      portlet is using in its code for referencing this event.
      Used in: portlet-app
      </documentation>
    </annotation>
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
       maxOccurs="unbounded"/>
      <choice>
        <element name="qname" type="xs:QName"/>
        <element name="name" type="xs:NCName"/>
      </choice>
      <element name="alias" type="xs:QName" minOccurs="0"
       maxOccurs="unbounded"/>
      <element name="value-type" type="portlet:fully-qualified-classType"
       minOccurs="0"/>
    </sequence>
    <attribute name="id" type="portlet:string" use="optional"/>
  </complexType>
  <complexType name="event-definition-referenceType">
    <annotation>
      <documentation>
      The event-definition-referenceType is used to reference events
      declared with the event-definition element at application level.
      Used in: portlet
      </documentation>
    </annotation>
    <choice>
      <element name="qname" type="xs:QName"/>
      <element name="name" type="xs:NCName"/>
    </choice>
    <attribute name="id" type="portlet:string" use="optional"/>
  </complexType>
  <complexType name="listenerType">
    <annotation>
      <documentation>
      The listenerType is used to declare listeners for this portlet
      application.
      Used in: portlet-app
      </documentation>
    </annotation>
```

```
    <sequence>
      <element name="description" type="portlet:descriptionType" minOccurs="0"
       maxOccurs="unbounded"/>
      <element name="display-name" type="portlet:display-nameType"
       minOccurs="0" maxOccurs="unbounded"/>
      <element name="listener-class" type="portlet:fully-qualified-classType"/>
    </sequence>
    <attribute name="id" type="portlet:string" use="optional"/>
  </complexType>
  <complexType name="portlet-infoType">
    <sequence>
      <element name="title" type="portlet:titleType" minOccurs="0"/>
      <element name="short-title" type="portlet:short-titleType"
       minOccurs="0"/>
      <element name="keywords" type="portlet:keywordsType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="portlet:string" use="optional"/>
  </complexType>
  <simpleType name="portal-managedType">
    <annotation>
      <documentation>
      portal-managed indicates if a custom portlet mode
      needs to be managed by the portal or not.
      Per default all custom portlet modes are portal managed.
      Valid values are:
      - true for portal-managed
      - false for not portal managed
      Used in: custom-portlet-modes
      </documentation>
    </annotation>
    <restriction base="portlet:string">
      <enumeration value="true"/>
      <enumeration value="false"/>
    </restriction>
  </simpleType>
  <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="portlet: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, filter-mapping
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet: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="portlet: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="portlet: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>
      Name of the resource bundle containing the language specific
      portlet informations in different languages (Filename without
      the language specific part (e.g. _en) and the ending (.properties).
      Used in: portlet-info
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet: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="portlet: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="portlet: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="portlet:string" use="optional"/>
```

```
</complexType>
<complexType name="public-render-parameterType">
  <annotation>
    <documentation>
    The public-render-parameters defines a render parameter that is allowed
    to be public and thus be shared with other portlets.
    The identifier must be used for referencing this public render parameter
    in the portlet code.
    Used in: portlet-app
    </documentation>
  </annotation>
  <sequence>
    <element name="description" type="portlet:descriptionType" minOccurs="0"
     maxOccurs="unbounded"/>
    <element name="identifier" type="portlet:string"/>
    <choice>
      <element name="qname" type="xs:QName"/>
      <element name="name" type="xs:NCName"/>
    </choice>
    <element name="alias" type="xs:QName" minOccurs="0"
     maxOccurs="unbounded"/>
  </sequence>
  <attribute name="id" type="portlet: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="portlet: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"/>
    <element name="window-state" type="portlet:window-stateType"
     minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="id" type="portlet:string" use="optional"/>
</complexType>
<complexType name="supported-localeType">
```

```
    <annotation>
      <documentation>
      Indicated the locales the portlet supports.
      Used in: portlet
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet:string"/>
    </simpleContent>
  </complexType>
  <complexType name="titleType">
    <annotation>
      <documentation>
      Locale specific static title for this portlet.
      Used in: portlet-info
      </documentation>
    </annotation>
    <simpleContent>
      <extension base="portlet: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="portlet: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="portlet: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="portlet: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="portlet: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="portlet: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 JavaEE
      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>
  <simpleType name="filter-nameType">
    <annotation>
      <documentation>
      The logical name of the filter is declare
      by using filter-nameType. This name is used to map the
      filter.  Each filter name is unique within the portlet
      application.
      Used in: filter, filter-mapping
      </documentation>
    </annotation>
    <restriction base="portlet:string"/>
  </simpleType>
</schema>
```

PLT.25.6 Pictures of the structure of a Deployment Descriptor

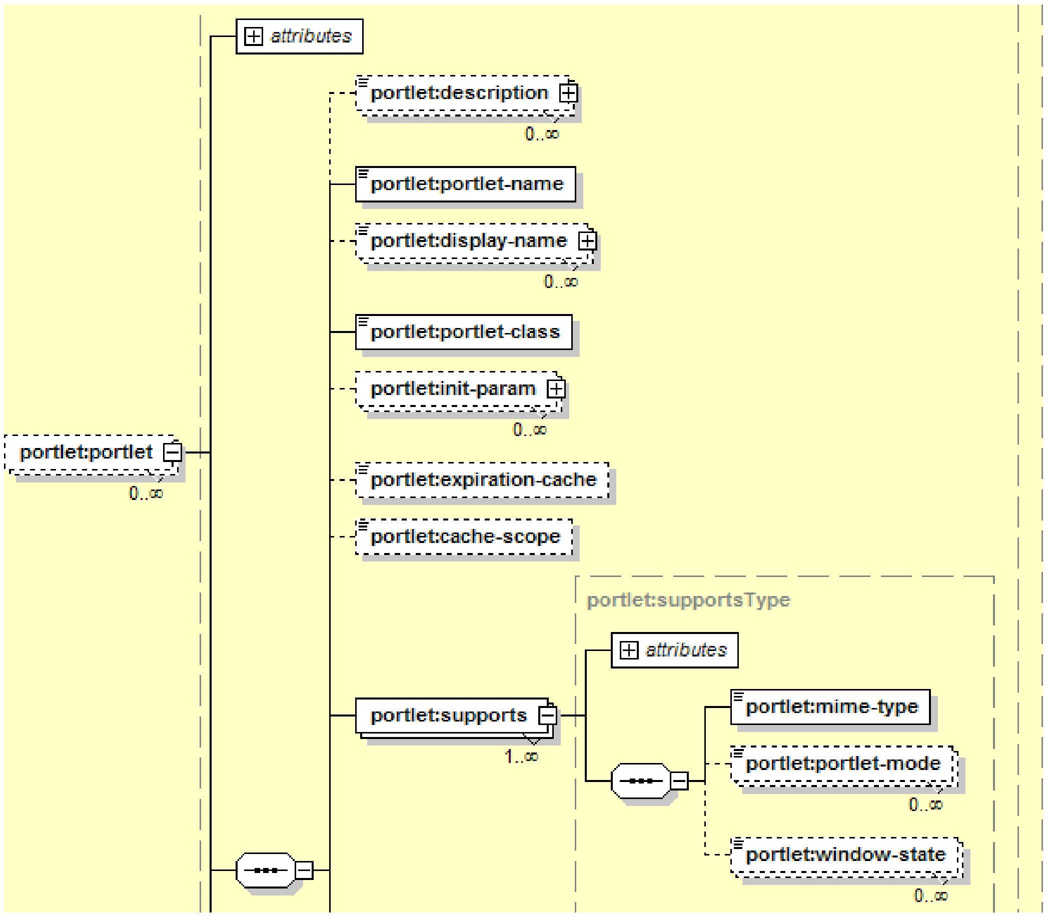


Figure 3: Part one of the portlet element

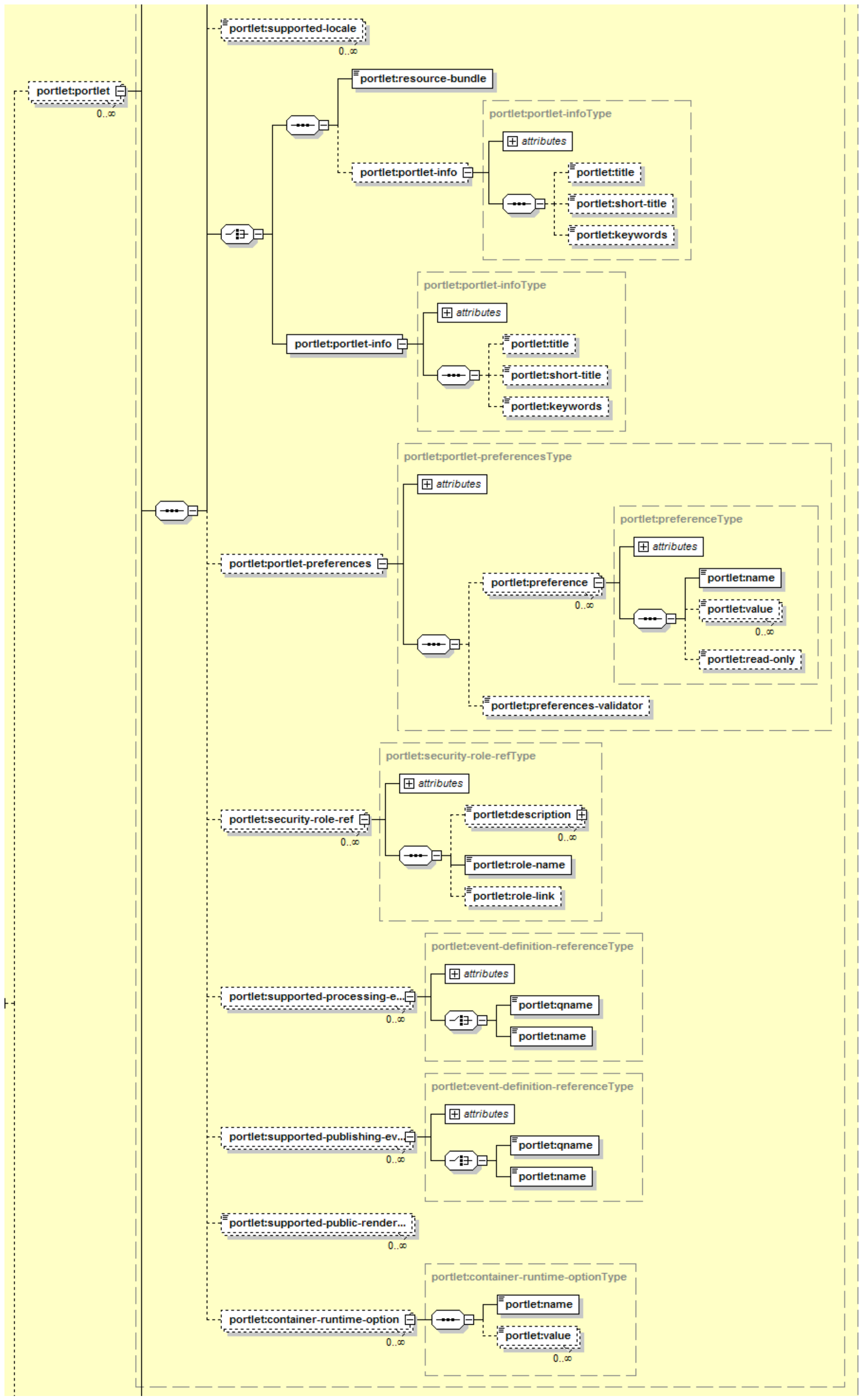


Figure 4: Part 2 of the portlet element

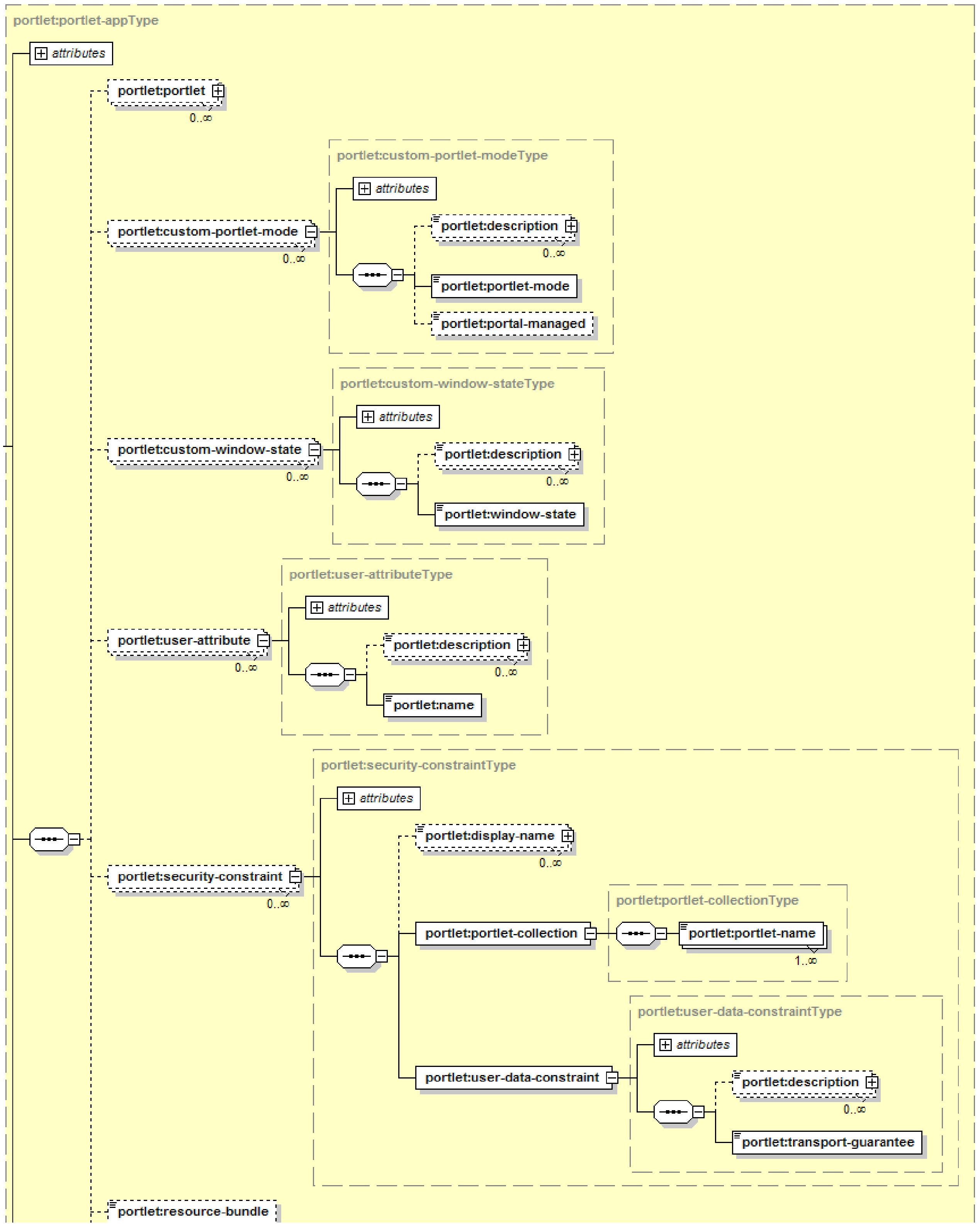


Figure 5: Part 1 of the portlet-app element

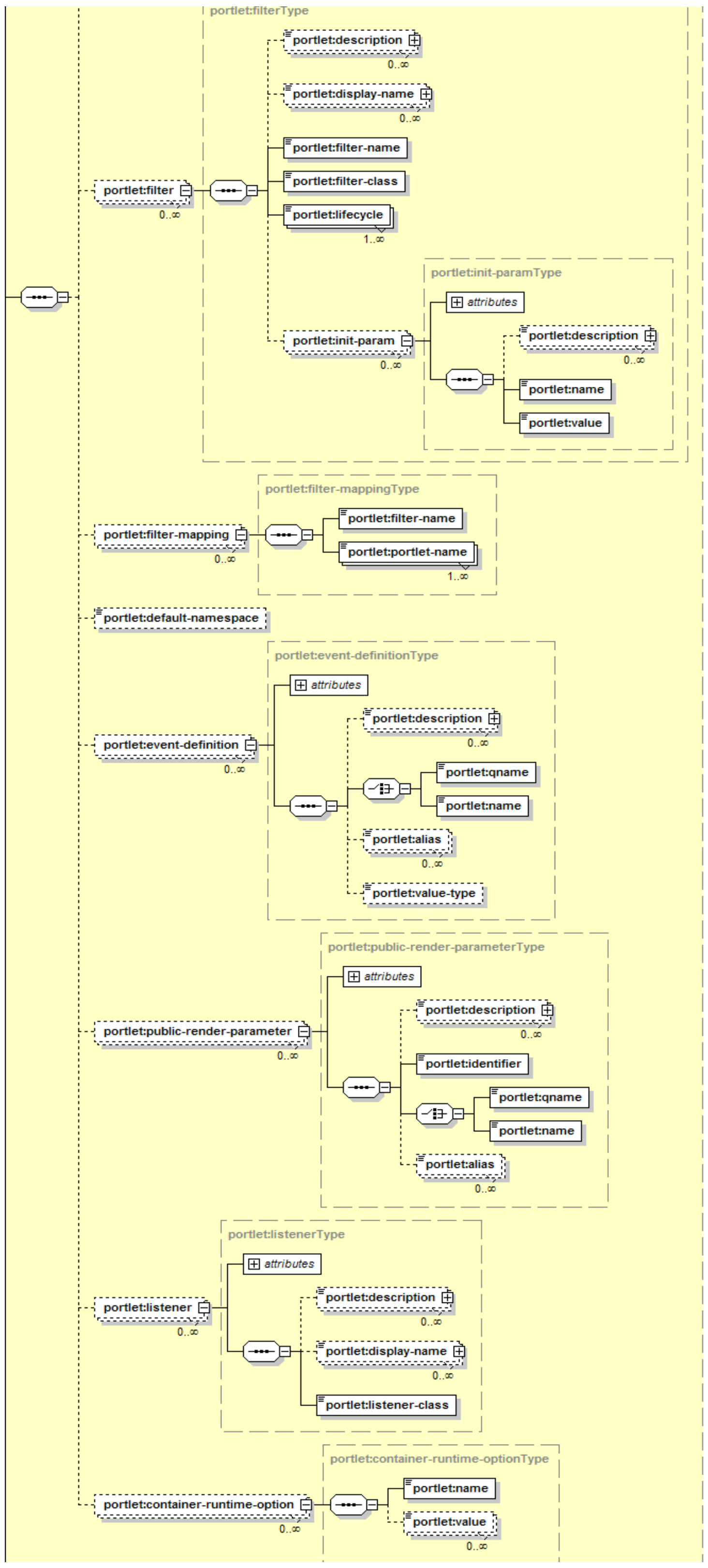


Figure 6: Part 2 of the portlet-app element

PLT.25.7 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>` and `<qname>`
- public-render-parameter `<name>` and `<qname>`
- filter `<filter-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>`
- `<supported-processing-event>`
- `<supported-publishing-event>`
- `<supported-public-render-parameter>`

PLT.25.8 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.25.8.1 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.

The preferred method for localization of values in the deployment descriptor is providing a resource bundle via the `<resource-bundle>` element on the portlet application level (see Resource Bundle section below).

PLT.25.8.2 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).

The supported locales are meta information intended to be used by the portal application.

PLT.25.9 Deployment Descriptor Example

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
version="2.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
                       http://java.sun.com/xml/ns/portlet/portlet-app_2_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.25.10 Resource Bundles

As an alternative to embedding all localized values in the deployment descriptor the portlet can provide a separate resource bundle containing the localized values. Providing localized values via resource bundles is the preferred way, as it allows the separation of deployment descriptor values from 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 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>. <name> uses the string representation of the Java `QName` class with `{namespace}localpart`. If the namespace is missing the defined default namespace is assumed. Note that the resource bundle name needs to comply with the `java.util.Property.store` method, i.e. the ":" must be escaped.
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. <name> uses the string representation of the Java `QName` class with `{namespace}localpart`. If the namespace is missing the defined default namespace is assumed. Note that the resource bundle name needs to comply with the `java.util.Property.store` method, i.e. the ":" must be escaped.
javax.portlet.app. public-render-parameter. <name>.description	Description of the public render parameter <name>.
javax.portlet.app. public-render-parameter. <name>.display-name	Name under which this public 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` element.

The Java Portlet Specification defines the following constants for 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.app.custom-portlet-mode.<name>.decoration-name	Decoration name for the portlet managed custom portlet mode <name>.

PLT.25.11 Resource Bundle Example

This section shows the resource bundles for the world population clock portlet from the 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.26

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 request, like `RenderRequest` or `ResourceRequest` and response, like `ActionResponse` or `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.[cccxii] 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*.

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_2_0" prefix="portlet"
%>
```

Since Java Portlet Specification V2.0 JSP V2.0 is supported and thus the Portlet Tag Library implementation should support the JSP 2.0 Expression Language (EL) for the tags in the Portlet Tag Library.

In order to support Java Portlet Specification V1.0 portlets that references the V1.0 tag library via

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

the portlet container must also support the V1.0 tag library defined in JSR 168.

PLT.26.1 defineObjects Tag

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

- `RenderRequest renderRequest` when included from within the `render` method, `null` or not defined otherwise
- `ResourceRequest resourceRequest` when included from within the `serveResource` method, `null` or not defined otherwise
- `ActionRequest actionRequest` when included from within the `processAction` method, `null` or not defined otherwise

- `EventRequest eventRequest` when included from within the `processEvent` method, `null` or not defined otherwise
- `RenderResponse renderResponse` when included from within the `render` method, `null` or not defined otherwise
- `ResourceResponse resourceResponse` when included from within the `serveResource` method, `null` or not defined otherwise
- `ActionResponse actionResponse` when included from within the `processAction` method, `null` or not defined otherwise
- `EventResponse eventResponse when included from within the processEvent method, null or not defined otherwise`
- `PortletConfig portletConfig`
- `PortletSession portletSession`, providing access to the portletSession, does not create a new session, only returns an existing session or `null` if no session exists.
- `Map<String, Object> portletSessionScope`, providing access to the portletSession attributes as a Map equivalent to the `PortletSession.getAttributeMap()` call, does not create a new session, only returns an existing session. If no session attributes exist this method returns an empty Map.
- `PortletPreferences portletPreferences`, providing access to the portlet preferences.
- `Map<String, String[]> portletPreferencesValues`, providing access to the portlet preferences as a Map, equivalent to the `PortletPreferences.getMap()` call. If no portlet preferences exist this method returns an empty Map.

These variables must reference the same Portlet API objects stored in the request object of the JSP as defined in the *PLT.19.3.2 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.[cccxiv]

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

```
<portlet:defineObjects/>

<%=renderResponse.getCacheControl().setExpirationTime(10)%>
```

After using the `defineObjects` tag, the JSP invokes the getCacheControl() method of the renderResponse to set the expiration time of the response to 10 seconds.

PLT.26.2 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.[cccxv]

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.[cccxvi] 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.[cccxvii] 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.[cccxviii] 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. [cccxix]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. [cccxx]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`.[cccxxi]

  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.[cccxxii]

- **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.[cccxxiii] If the security is not set for a URL, it must stay the same as the security setting of the current request.

- **copyCurrentRenderParameters** (Type: boolean, non-required) – if set to true requests that the private render parameters of the portlet of the current request must be attached to this URL. cccxxiv It is equivalent to setting each of the current private render parameters via the `<portlet:param>` tag. If additional `<portlet:param>` tags are specified parameters with the same name as an existing render parameter will get merged and the value defined in additional `<portlet:param>` tags must be pre-pended. cccxxv
  The default for this attribute is false.
- **escapeXml** (Type: boolean, non-required) – determines whether characters <,>,&,',” in the resulting output should be converted to their corresponding character entity codes ('<' gets converted to '&lt;', '>' gets converted to '&gt;' '&' gets converted to '&amp;', ''' gets converted to '&#039;', '”' gets converted to '&#034;'). cccxxvi Default value is true.
- **name** (Type: String, non-required) – specifies the name of the action that can be used by `GenericPortlet` to dispatch to methods annotated with `ProcessAction`. Setting this name will result in adding a parameter to this action URL with the name `javax.portlet.action`.

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.

A `JspException` with the `java.lang.IllegalStateException` that caused this error as root cause is thrown in the following cases:

- If this tag is used in markup provided by a `serveResource` call that was directly or indirectly triggered via a resource URL of type `FULL` or `PORTLET`.

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

```
<portlet:actionURL copyCurrentRenderParameters=”true”
windowState=”maximized” portletMode=”edit” name=”editStocks”>
   <portlet:param name=”page” value=”1”/>
</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.26.3 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. cccxxvii

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.[cccxxviii] 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.[cccxxix] 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.[cccxxx] 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.[cccxxxi] 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.[cccxxxii] 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`.[cccxxxiii]

  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.[cccxxxiv]

- **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.[cccxxxv]
- **copyCurrentRenderParameters** (Type: boolean, non-required) – if set to true requests that the private render parameters of the portlet of the current request must attached to this URL. [cccxxxvi] It is equivalent to setting each of the current private render parameters via the `<portlet:param>` tag. If additional `<portlet:param>` tags are specified parameters with the same name as an existing render parameter will get merged and the value defined in additional

`<portlet:param>` tags must be pre-pended. [cccxxxvii]
The default for this attribute is false.

- **escapeXml** (Type: boolean, non-required) – deterrmines whether characters <,>,&,',” in the resulting output should be converted to their corresponding character entity codes (‘<’ gets converted to ‘&lt;’, ‘>’ gets converted to ‘&gt;’ ‘&’ gets converted to ‘&amp;’, ‘'’ gets converted to ‘&#039;’, ‘”’ gets converted to ‘&#034;’). [cccxxxviii] Default value is true

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.

A `JspException` with the `java.lang.IllegalStateException` that caused this error as root cause is thrown in the following cases:

- If this tag is used in markup provided by a `serveResource` call that was directly or indirectly triggered via a resource URL of type `FULL` or `PORTLET`.

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.26.4 resourceURL Tag

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

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

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. [cccxli]

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

- **var** (Type: String, non-required) – name of the exported scoped variable for the resource 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`.[cccxlii]

  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.[cccxliii]

- **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.[cccxliv]
- **escapeXml** (Type: boolean, non-required) – determines whether characters <,>,&,’,” in the resulting output should be converted to their corresponding character entity codes (‘<’ gets converted to ‘&lt;’, ‘>’ gets converted to ‘&gt;’ ‘&’ gets converted to ‘&amp;’, ‘’’ gets converted to ‘&#039;’, ‘”’ gets converted to ‘&#034;’).[cccxlv] Default value is true
- **id** (type:String, non-required) – sets the ID for this resource. The ID can be retrieved in the `serveResource` call from the request via the `getResourceID` method.
- **cacheability** (type: String, non-required) – defines the cacheability of the markup returned by this resource URL. Valid values are: “`FULL`”, “`PORTLET`”, and “`PAGE`”. See Section PLT 13.6 for more details on the semantic of these constants.
  If cacheability is not set the default is `PAGE` cachability.

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.

A `JspException` with the `java.lang.IllegalStateException` that caused this error as root cause is thrown in the following cases:

- If this tag is used in markup provided by a `serveResource` call that was directly or indirectly triggered via a resource URL of a weaker cacheability type.

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

```
<portlet:resourceURL id=”icons/mypict.gif” var=”iconsURL”/>
<img src="<%=iconsURL%>” >
```

The example creates a URL to provide a link that renders the icon named `mypict.gif` via the default `GenericPortlet` resource serving mechanism.

PLT.26.5 namespace Tag

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

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.26.6 param Tag

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

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

If the `param` tag has an empty value the specified parameter name must be removed from the URL. [cccxlix] In the case of a resource URL an empty value does not alter the render parameters automatically added by the portlet container to resource URLs.

If the same name of a parameter occurs more than once within 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. [cccl]

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.26.7 property Tag

This tag defines a property that may be added to an `actionURL`, `renderURL` or `resourceURL` and is equivalent to the API call `addProperty()`.

The `property` tag should not contain any body content.

If the same name of a property occurs more than once within an `actionURL`, `renderURL` or `resourceURL` the values should be delivered as properties value array with the values in the order of the declaration within the URL tag.

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

- **name** (Type: String, required) – the name of the property to add to the URL. If `name` is null or empty, no action is performed.
- **value** (Type: String, required) – the value of the property 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:actionURL>
        <portlet:property name="myProperty" value="someValue"/>
</portlet:actionURL>
```

PLT.26.8 Changing the Default Behavior for escapeXml

In the Java Portlet Specification V1.0 the behavior in regards to XML escaping URLs written by the tag library was undefined and thus portlets may have been coded with the assumption that the URLs where not XML escaped. In order to be able to run these portlets on a Java Portlet Specification V 2.0 container the specification provides the `javax.portlet.escapeXml` container runtime option. The value of this setting can either be `true` for XML escaping URLs per default, or `false` for not XML escaping URLs per default.

Portlet that require that the default behavior for URLs written to the output stream via the portlet tag library should therefore define the following container runtime option in the portlet deployment descriptor:

```
<portlet>

…

      <container-runtime-option>
```

```
                <name>javax.portlet.escapeXml</name>

                <value>false</value>

          </container-runtime-option>

    </portlet>
```

If the portlet has defined the `javax.portlet.escapeXml` container runtime option the portlet container should honor this setting as otherwise the portlet may not work correctly.

PLT.27

Leveraging JAXB for Event payloads

The Java Portlet Specification 2.0 leverages the Java Architecture for

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

The event payload must be defined using the JAXB annotations in the Java object and defining the Java object class name in the deployment descript via the `value-type` element. The event payload must have a valid JAXB binding, or be in the list of Java primitive types / standard classes of the JAXB 2.0 specification section 8.5.1 or 8.5.2, and implement `java.io.Serializable`, otherwise a `java.lang.IllegalArgumentException` must be thrown. The primitive type `xsd:anyURI` must be mapped to `java.net.URI` and not `java.lang.String`, which is the default in JAXB, in order to not loose semantics.

PLT.28

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.28.1 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.28.2 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.28.2.1 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.28.2.1.1 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.28.2.1.2 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.28.2.2 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.28.2.3 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.28.2.4 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 using the `@RenderMode(name=”about”)` annotation supported by the `GenericPortlet` class.

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>
    <portlet-mode>about</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 using the `@RenderMode(name="config")` annotation supported by the `GenericPortlet` class.

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>
    <portlet-mode>config</portlet-mode>
  </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 using the `@RenderMode(name="edit_defaults")` annotation supported by the `GenericPortlet` class.

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>
    <portlet-mode> edit_defaults </portlet-mode>
  </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 using the `@RenderMode(name=”preview”)` annotation supported by the `GenericPortlet` class.

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>
    <portlet-mode> preview </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 using the `@RenderMode(name=”print”)` annotation supported by the `GenericPortlet` class.

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

PLT.B

Markup Fragments

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

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

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

PLT.C

CSS Style Definitions

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

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

PLT.C.1 Links (Anchor)

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

PLT.C.2 Fonts

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

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

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

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

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

PLT.C.3 Messages

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

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

PLT.C.4 Sections

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 Tables

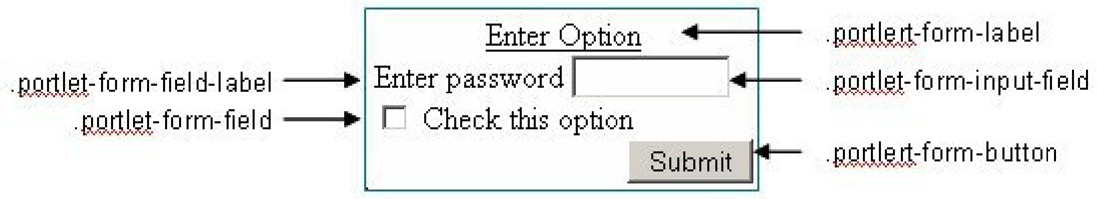
Table style definitions affect the rendering (i.e. alignment, borders, background color, etc.) as well as their text attributes.

Style	Description
portlet-table-header	Table header
portlet-table-body	Normal text in a table cell
portlet-table-alternate	Text in every other row in the table
portlet-table-selected	Text in a selected cell range
portlet-table-subheader	Text of a subheading
portlet-table-footer	Table footer
portlet-table-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 table).

5 ## PLT.C.6 Forms

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

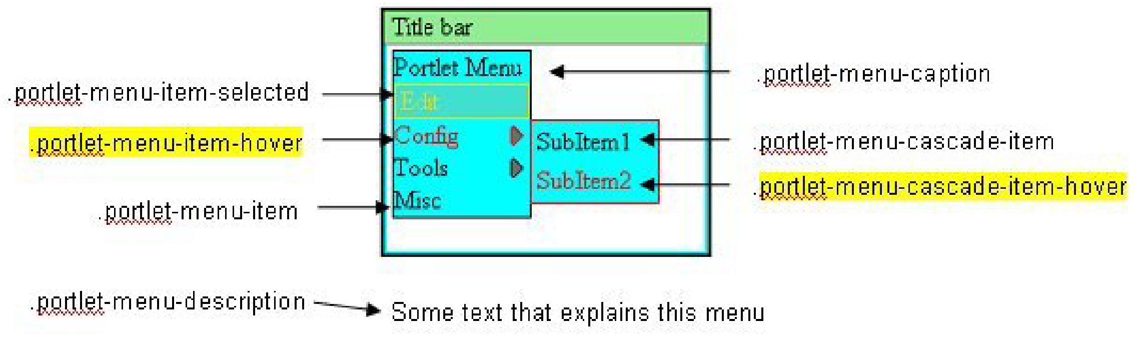
Style	Description
portlet-form-label	Text used for the descriptive label of the whole form (not the labels for fields.
portlet-form-input-field	Text of the user-input in an input field.
portlet-form-button	Text on a button
portlet-icon-label	Text that appears beside a context dependent action icon.
portlet-dlg-icon-label	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.7 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


Title bar
Portlet Menu
Edit
Config
Tools
Misc
SubItem1
SubItem2
.portlet-menu-item-selected
.portlet-menu-item-hover
.portlet-menu-item
.portlet-menu-caption
.portlet-menu-cascade-item
.portlet-menu-cascade-item-hover
.portlet-menu-description
Some text that explains this menu

.

PLT.D

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.ymd.year`
`user.bdate.ymd.month`
`user.bdate.ymd.day`
`user.bdate.hms.hour`
`user.bdate.hms.minute`
`user.bdate.hms.second`
`user.bdate.fractionsecond`
`user.bdate.timezone`
`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.login.id`
`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

The P3P user attribute constants can be accessed in the portlet via the `P3PUserInfos` enum on the `PortletRequest`.

PLT.D.1 Example

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

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

PLT.E

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.E.1.1 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.F

TCK Assertions

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

Assertions marked as Testable=false are not verifiable.

[i] SPEC:1	Testable=true	Section=PLT.2.5
[ii] SPEC:2	Testable=false	Section=PLT.5.1
[iii] SPEC:3	Testable=false	Section=PLT.5.1
[iv] SPEC:4	Testable=false	Section=PLT.5.2.1
[v] SPEC:5	Testable=true	Section=PLT.5.2.2
[vi] SPEC:6	Testable=true	Section=PLT.5.2.2.1
[vii] SPEC:7	Testable=true	Section=PLT.5.2.2.1
[viii] SPEC:8	Testable=true	Section=PLT.5.2.2.1
[ix] SPEC:9	Testable=true	Section=PLT.5.2.2.1
[x] SPEC:10	Testable=false	Section=PLT.5.2.3
[xi] SPEC:11	Testable= false	Section=PLT.5.2.3
[xii] SPEC:12	Testable=false	Section=PLT.5.2.3
[xiii] SPEC:13	Testable= false	Section=PLT.5.2.3
[xiv] SPEC:14	Testable=true	Section=PLT 5.4

[xv] SPEC:15	Testable=true	Section=PLT 5.4
[xvi] SPEC:16	Testable=true	Section=PLT 5.4
[xvii] SPEC:17	Testable=true	Section=PLT 5.4
[xviii] SPEC:18	Testable= true	Section=PLT.5.4.1
[xix] SPEC:19	Testable= true	Section=PLT.5.4.5.4
[xx] SPEC:20	Testable= true	Section=PLT.5.4.5.4
[xxi] SPEC:21	Testable= true	Section=PLT.5.4.5.4
[xxii] SPEC:22	Testable=true	Section=PLT 5.4.5.4
[xxiii] SPEC:23	Testable= true	Section=PLT.5.4.7
[xxiv] SPEC:24	Testable=false	Section=PLT.5.4.7
[xxv] SPEC:25	Testable= true	Section=PLT.5.4.7
[xxvi] SPEC:26	Testable= true	Section=PLT.6.2
[xxvii] SPEC:27	Testable= true	Section=PLT.6.2
[xxviii] SPEC:28	Testable= true	Section=PLT.6.3
[xxix] SPEC:29	Testable= true	Section=PLT.6.4
[xxx] SPEC:30	Testable= true	Section=PLT.6.5
[xxxi] SPEC:31	Testable= true	Section=PLT.6.5
[xxxii] SPEC:32	Testable= true	Section=PLT.6.6
[xxxiii] SPEC:33	Testable= true	Section=PLT.6.6
[xxxiv] SPEC:34	Testable= true	Section=PLT.6.7
[xxxv] SPEC:35	Testable= true	Section=PLT.7.1
[xxxvi] SPEC:36	Testable= true	Section=PLT.7.1
[xxxvii] SPEC:37	Testable= true	Section=PLT.7.1.1
[xxxviii] SPEC:38	Testable= true	Section=PLT.7.1.1

[xxxix] SPEC:39	Testable= true	Section=PLT.7.1.1
[xl] SPEC:40	Testable= true	Section=PLT.7.1.1
[xli] SPEC:41	Testable= true	Section=PLT.7.1.2
[xlii] SPEC:42	Testable= true	Section=PLT.7.1.2
[xliii] SPEC:43	Testable= true	Section=PLT.7.1.2
[xliv] SPEC:44	Testable= true	Section=PLT.7.1.2
[xlv] SPEC:45	Testable= true	Section=PLT.7.1.2
[xlvi] SPEC:46	Testable= true	Section=PLT.7.1.3
[xlvii] SPEC:47	Testable= true	Section=PLT.7.2.1
[xlviii] SPEC:48	Testable= true	Section=PLT.7.2.1
[xlix] SPEC:49	Testable= true	Section=PLT.7.2.1
[l] SPEC:50	Testable= true	Section=PLT.7.2.1
[li] SPEC:51	Testable= true	Section=PLT.7.2.2
[lii] SPEC:52	Testable=true	Section=PLT.8.5
[liii] SPEC:53	Testable=true	Section=PLT.8.6
[liv] SPEC:54	Testable=true	Section=PLT.8.6
[lv] SPEC:55	Testable=false	Section=PLT.8.6
[lvi] SPEC:56	Testable=true	Section=PLT.9.4
[lvii] SPEC:57	Testable=true	Section=PLT.9.5
[lviii] SPEC:58	Testable=false	Section=PLT.9.5
[lix] SPEC:59	Testable=false	Section=PLT.10.1
[lx] SPEC:60	Testable=false	Section=PLT.10.1
[lxi] SPEC:61	Testable=true	Section=PLT.10.3
[lxii] SPEC:62	Testable=true	Section=PLT.10.3

lxiii SPEC:63	Testable=true	Section=PLT.10.3
lxiv SPEC:64	Testable=true	Section=PLT.10.3
lxv SPEC:65	Testable=true	Section=PLT.10.3(servlet spec)
lxvi SPEC:66	Testable=true	Section=PLT.10.4.4
lxvii SPEC:67	Testable=true	Section=PLT.11.1.1
lxviii SPEC:68	Testable= true	Section=PLT.11.1.1
lxix SPEC:69	Testable=true	Section=PLT.11.1.1
lxx SPEC:70	Testable=true	Section=PLT.11.1.1
lxxi SPEC:71	Testable=true	Section=PLT.11.1.1
lxxii SPEC:72	Testable=true	Section=PLT.11.1.1
lxxiii SPEC:73	Testable=true	Section=PLT.11.1.1
lxxiv SPEC:74	Testable= true	Section=PLT.11.1.1.2
lxxv SPEC:75	Testable= true	Section=PLT.11.1.1.2
lxxvi SPEC:76	Testable=true	Section=PLT.11.1.1.2
lxxvii SPEC:77	Testable=true	Section=PLT.11.1.1.3
lxxviii SPEC:78	Testable=true	Section=PLT.11.1.1.3
lxxix SPEC:79	Testable= true	Section=PLT.11.1.1.3
lxxx SPEC:80	Testable=true	Section=PLT.11.1.1.3
lxxxi SPEC:81	Testable= false	Section=PLT.11.1.2
lxxxii SPEC:82	Testable= true	Section=PLT.11.1.2
lxxxiii SPEC:83	Testable= true	Section=PLT.11.1.2
lxxxiv SPEC:84	Testable= true	Section=PLT.11.1.2
lxxxv SPEC:85	Testable= true	Section=PLT.11.1.2
lxxxvi SPEC:86	Testable= true	Section=PLT.11.1.2

[lxxxvii] SPEC:87	Testable=false	Section=PLT.11.1.3
[lxxxviii] SPEC:88	Testable=true	Section=PLT.11.1.4.1
[lxxxix] SPEC:89	Testable=true	Section=PLT.11.1.4.2
[xc] SPEC:90	Testable=true	Section=PLT.11.1.4.4
[xci] SPEC:91	Testable= true	Section=PLT.11.1.6
[xcii] SPEC:92	Testable=true	Section=PLT.11.1.6
[xciii] SPEC:93	Testable=true	Section=PLT.11.1.7
[xciv] SPEC:94	Testable=true	Section=PLT.11.1.8
[xcv] SPEC:95	Testable=true	Section=PLT.11.1.8
[xcvi] SPEC:96	Testable=true	Section=PLT.11.1.8
[xcvii] SPEC:97	Testable=true	Section=PLT.11.1.12
[xcviii] SPEC:98	Testable=true	Section=PLT.11.2.1
[xcix] SPEC:99	Testable=true	Section=PLT.11.2.1
[c] SPEC:100	Testable= true	Section=PLT.12.1.3
[ci] SPEC:101	Testable= true	Section=PLT.12.1.3
[cii] SPEC:102	Testable=true	Section=PLT.12.1.3
[ciii] SPEC:103	Testable= true	Section=PLT.12.2.1
[civ] SPEC:104	Testable=true	Section=PLT.12.2.2
[cv] SPEC:105	Testable=true	Section=PLT.12.2.2
[cvi] SPEC:106	Testable=true	Section=PLT.12.3.1
[cvii] SPEC:107	Testable=true	Section=PLT.12.3.1
[cviii] SPEC:108	Testable=true	Section=PLT.12.3.1
[cix] SPEC:109	Testable=true	Section=PLT.12.3.1
[cx] SPEC:110	Testable=true	Section=PLT.12.3.1

[cxi] SPEC:111	Testable=true	Section=PLT.12.2.1
[cxii] SPEC:112	Testable=true	Section=PLT.12.5.1
[cxiii] SPEC:113	Testable= true	Section=PLT.12.5.1
[cxiv] SPEC:114	Testable= true	Section=PLT.12.5.2
[cxv] SPEC:115	Testable=true	Section=PLT.12.5.5
[cxvi] SPEC:116	Testable=true	Section=PLT.12.5.5
[cxvii] SPEC:117	Testable=true	Section=PLT.12.5.5
[cxviii] SPEC:118	Testable=true	Section=PLT.12.5.5
[cxix] SPEC:119	Testable=true	Section=PLT.12.5.5
[cxx] SPEC:120	Testable=true	Section=PLT.12.5.5
[cxxi] SPEC:121	Testable=false	Section=PLT.12.6.1
[cxxii] SPEC:122	Testable=true	Section=PLT.13.1.4
[cxxiii] SPEC:123	Testable=true	Section=PLT.13.1.5
[cxxiv] SPEC:124	Testable=true	Section=PLT.13.1.5
[cxxv] SPEC:125	Testable=true	Section=PLT.13.1.6
[cxxvi] SPEC:126	Testable=true	Section=PLT.13.1.6
[cxxvii] SPEC:128	Testable=true	Section=PLT.13.1.6
[cxxviii] SPEC:129	Testable=true	Section=PLT.13.1.6
[cxxix] SPEC:130	Testable=true	Section=PLT.13.1.7
[cxxx] SPEC:131	Testable=true	Section=PLT.13.1.7
[cxxxi] SPEC:132	Testable=true	Section=PLT.13.1.7
[cxxxii] SPEC:133	Testable=true	Section=PLT.13.1.7
[cxxxiii] EVENT:134	Testable= true	Section=PLT.15.2.2
[cxxxiv] EVENT:135	Testable= true	Section=PLT.15.2.2

[cxxxv] SPEC:137	Testable= true	Section=PLT.15.2.3
[cxxxvi] SPEC:138	Testable= true	Section=PLT.15.2.3
[cxxxvii] SPEC:139	Testable= true	Section=PLT.15.2.3
[cxxxviii] SPEC:140	Testable= true	Section=PLT.15.2.3
[cxxxix] SPEC:141	Testable= true	Section=PLT.15.2.3
[cxl] SPEC:142	Testable= true	Section=PLT.15.2.4.1
[cxli] EVENT:143	Testable= true	Section=PLT.15.2.4.1
[cxlii] SPEC:144	Testable= true	Section=PLT.15.2.4
[cxliii] EVENT:145	Testable= true	Section=PLT.15.2.5
[cxliv] EVENT:146	Testable= true	Section=PLT.15.2.5
[cxlv] EVENT:147	Testable= true	Section=PLT.15.2.5
[cxlvi] EVENT:148	Testable= true	Section=PLT.15.2.6
[cxlvii] EVENT:149	Testable= true	Section=PLT.15.2.6
[cxlviii] SPEC:150	Testable= true	Section=PLT.17.1
[cxlix] SPEC:151	Testable= true	Section=PLT.17.1
[cl] SPEC:152	Testable=true	Section=PLT.17.1
[cli] SPEC:153	Testable=true	Section=PLT.17.1
[clii] SPEC:154	Testable=true	Section=PLT.17.1
[cliii] SPEC:155	Testable=true	Section=PLT.17.1
[cliv] SPEC:156	Testable=true	Section=PLT.17.1
[clv] SPEC:157	Testable= true	Section=PLT.17.1(change)
[clvi] SPEC:158	Testable=true	Section=PLT.17.1
[clvii] SPEC:159	Testable=true	Section=PLT.17.3
[clviii] SPEC:160	Testable=true	Section=PLT.17.3

[clix] SPEC:161	Testable=true	Section=PLT.17.4
[clx] SPEC:162	Testable=true	Section=PLT.17.4
[clxi] SPEC:163	Testable=true	Section=PLT.17.4
[clxii] SPEC:164	Testable=true	Section=PLT.18.1
[clxiii] SPEC:165	Testable=true	Section=PLT.18.1
[clxiv] SPEC:166	Testable=true	Section=PLT.18.2
[clxv] SPEC:167	Testable=true	Section=PLT.18.2
[clxvi] SPEC:168	Testable=true	Section=PLT.18.3
[clxvii] SPEC:169	Testable=true	Section=PLT.18.3
[clxviii] SPEC:170	Testable=true	Section=PLT.18.3
[clxix] SPEC:171	Testable=true	Section=PLT.18.4
[clxx] SPEC:172	Testable=true	Section=PLT.18.4
[clxxi] SPEC:173	Testable=true	Section=PLT.18.4
[clxxii] SPEC:174	Testable=true	Section=PLT.18.4
[clxxiii] SPEC:175	Testable=true	Section=PLT.18.4.1
[clxxiv] SPEC:176	Testable=true	Section=PLT.18.4.1
[clxxv] SPEC:177	Testable=true	Section=PLT.18.4.1
[clxxvi] SPEC:178	Testable=true	Section=PLT.18.9(servlet spec)
[clxxvii] SPEC:179	Testable=true	Section=PLT.19.1
[clxxviii] SPEC:180	Testable=true	Section=PLT.19.1
[clxxix] SPEC:181	Testable= true	Section=PLT.19.1.1
[clxxx] SPEC:182	Testable=true	Section=PLT.19.2
[clxxxi] SPEC:183	Testable=true	Section=PLT.19.2
[clxxxii] SPEC:184	Testable=true	Section=PLT.19.3

[clxxxiii] SPEC:185	Testable=true	Section=PLT.19.3.1
[clxxxiv] SPEC:186	Testable=true	Section=PLT.16.3.2
[clxxxv] SPEC:187	Testable=true	Section=PLT.19.3.3
[clxxxvi] SPEC:188	Testable=true	Section=PLT.19.3.3
[clxxxvii] SPEC:189	Testable=true	Section=PLT.19.3.3
[clxxxviii] SPEC:190	Testable=true	Section= PLT.19.3.3
[clxxxix] SPEC:191	Testable=true	Section=PLT.19.3.3
[cxc] SPEC:192	Testable=true	Section=PLT.19.3.3
[cxci] SPEC:193	Testable=true	Section=PLT.19.3.3
[cxcii] SPEC:194	Testable=true	Section=PLT.19.3.3
[cxciii] SPEC:195	Testable=true	Section=PLT.19.3.3
[cxciv] SPEC:196	Testable=true	Section= PLT.19.3.3
[cxcv] SPEC:197	Testable=true	Section= PLT.19.3.3
[cxcvi] SPEC:198	Testable=true	Section= PLT.19.3.3
[cxcvii] SPEC:199	Testable=true	Section= PLT.19.3.3
[cxcviii] SPEC:200	Testable=true	Section= PLT.19.3.3
[cxcix] SPEC:201	Testable=true	Section= PLT.19.3.3
[cc] SPEC:202	Testable=true	Section= PLT.19.3.3
[cci] SPEC:203	Testable=false(impl)	Section= PLT.19.3.3
[ccii] SPEC:204	Testable=true	Section= PLT.19.3.3
[cciii] SPEC:205	Testable=true	Section= PLT.19.3.3
[cciv] SPEC:206	Testable=true	Section=PLT.19.3.4
[ccv] SPEC:207	Testable=true	Section=PLT.19.3.4
[ccvi] SPEC:208	Testable=true	Section=PLT.19.3.4

[ccvii] SPEC:209	Testable=true	Section= PLT.19.3.4
[ccviii] SPEC:210	Testable=true	Section=PLT.19.3.4
[ccix] SPEC:211	Testable=true	Section=PLT.19.3.4
[ccx] SPEC:212	Testable=true	Section=PLT.19.3.4
[ccxi] SPEC:213	Testable=true	Section= PLT.19.3.4
[ccxii] SPEC:214	Testable=true	Section= PLT.19.3.4
[ccxiii] SPEC:215	Testable=true	Section= PLT.19.3.4
[ccxiv] SPEC:216	Testable=true	Section= PLT.19.3.4
[ccxv] SPEC:217	Testable=true	Section= PLT.19.3.4
[ccxvi] SPEC:218	Testable=true	Section= PLT.19.3.4
[ccxvii] SPEC:219	Testable=false(impl)	Section= PLT.19.3.4
[ccxviii] SPEC:220	Testable=true	Section= PLT.19.3.4
[ccxix] SPEC:221	Testable=true	Section= PLT.19.3.4
[ccxx] SPEC:222	Testable=true	Section=PLT.19.3.5
[ccxxi] SPEC:223	Testable=true	Section=PLT.19.3.5
[ccxxii] SPEC:224	Testable=true	Section=PLT.19.3.5
[ccxxiii] SPEC:225	Testable=true	Section= PLT.19.3.5
[ccxxiv] SPEC:226	Testable=true	Section= PLT.19.3.5
[ccxxv] SPEC:227	Testable=true	Section=PLT.19.3.5
[ccxxvi] SPEC:228	Testable=true	Section= PLT.19.3.5
[ccxxvii] SPEC:229	Testable=true	Section= PLT.19.3.5
[ccxxviii] SPEC:230	Testable=true	Section= PLT.19.3.5
[ccxxix] SPEC:231	Testable=true	Section= PLT.19.3.5
[ccxxx] SPEC:232	Testable=true	Section= PLT.19.3.5

ccxxxi SPEC:233	Testable=false(impl)	Section= PLT.19.3.5
ccxxxii SPEC:234	Testable=true	Section= PLT.19.3.5
ccxxxiii SPEC:235	Testable=true	Section=PLT.19.3.7
ccxxxiv SPEC:236	Testable=true	Section=PLT.19.3.7
ccxxxv SPEC:237	Testable=true	Section=PLT.19.3.8
ccxxxvi SPEC:238	Testable=true	Section= PLT.19.4
ccxxxvii SPEC:239	Testable=true	Section= PLT.19.4
ccxxxviii SPEC:240	Testable=true	Section= PLT.19.4
ccxxxix SPEC:241	Testable=true	Section= PLT.19.4.2
ccxl SPEC:242	Testable=true	Section= PLT.19.4.2
ccxli SPEC:243	Testable=true	Section= PLT.19.4.2
ccxlii SPEC:244	Testable=true	Section= PLT.19.4.2
ccxliii SPEC:245	Testable=true	Section=PLT.19.4.3
ccxliv SPEC:246	Testable=true	Section=PLT.19.4.3
ccxlv SPEC:247	Testable=true	Section=PLT.19.4.3
ccxlvi SPEC:248	Testable=true	Section= PLT.19.4.3
ccxlvii SPEC:249	Testable=true	Section=PLT.19.4.3
ccxlviii SPEC:250	Testable=true	Section=PLT.19.4.3
ccxlix SPEC:251	Testable=true	Section=PLT.19.4.3
ccl SPEC:252	Testable=true	Section=PLT.19.4.3
ccli SPEC:253	Testable=true	Section=PLT.19.4.3
cclii SPEC:254	Testable=true	Section= PLT.19.4.3
ccliii SPEC:255	Testable=true	Section= PLT.19.4.3
ccliv SPEC:256	Testable=true	Section= PLT.19.4.3

cclv SPEC:257	Testable=true	Section= PLT.19.4.3
cclvi SPEC:258	Testable=true	Section= PLT.19.4.3
cclvii SPEC:259	Testable=true	Section= PLT.19.4.3
cclviii SPEC:260	Testable=false(impl)	Section= PLT.19.4.3
cclix SPEC:261	Testable=false(impl)	Section= PLT.19.4.3
cclx SPEC:262	Testable=true	Section= PLT.19.4.3
cclxi SPEC:263	Testable=true	Section= PLT.19.4.3
cclxii SPEC:264	Testable=true	Section=PLT.19.4.4
cclxiii SPEC:265	Testable=true	Section=PLT.19.4.4
cclxiv SPEC:266	Testable=true	Section=PLT.19.4.4
cclxv SPEC:267	Testable=true	Section= PLT.19.4.4
cclxvi SPEC:268	Testable=true	Section=PLT.19.4.4
cclxvii SPEC:269	Testable=true	Section=PLT.19.4.4
cclxviii SPEC:270	Testable=true	Section=PLT.19.4.4
cclxix SPEC:271	Testable=true	Section= PLT.19.4.4
cclxx SPEC:272	Testable=true	Section= PLT.19.4.4
cclxxi SPEC:273	Testable=true	Section= PLT.19.4.4
cclxxii SPEC:274	Testable=true	Section= PLT.19.4.4
cclxxiii SPEC:275	Testable=true	Section= PLT.19.4.4
cclxxiv SPEC:276	Testable=true	Section= PLT.19.4.4
cclxxv SPEC:277	Testable=false(impl)	Section= PLT.19.4.4
cclxxvi SPEC:278	Testable=false(impl)	Section= PLT.19.4.4
cclxxvii SPEC:279	Testable=true	Section= PLT.19.4.4
cclxxviii SPEC:280	Testable=true	Section=PLT.19.4.5

[cclxxix] SPEC:281	Testable=true	Section=PLT.19.4.5
[cclxxx] SPEC:282	Testable=true	Section=PLT.19.4.5
[cclxxxi] SPEC:283	Testable=true	Section= PLT.19.4.5
[cclxxxii] SPEC:284	Testable=true	Section= PLT.19.4.5
[cclxxxiii] SPEC:285	Testable=true	Section=PLT.19.4.5
[cclxxxiv] SPEC:286	Testable=true	Section= PLT.19.4.5
[cclxxxv] SPEC:287	Testable=true	Section= PLT.19.4.5
[cclxxxvi] SPEC:288	Testable=true	Section= PLT.19.4.5
[cclxxxvii] SPEC:289	Testable=true	Section= PLT.19.4.5
[cclxxxviii] SPEC:290	Testable=true	Section= PLT.19.4.5
[cclxxxix] SPEC:291	Testable=false(impl)	Section= PLT.19.4.5
[ccxc] SPEC:292	Testable=false(impl)	Section= PLT.19.4.5
[ccxci] SPEC:293	Testable=true	Section=PLT.19.5
[ccxcii] SPEC:294	Testable=true	Section=PLT.20.2.1
[ccxciii] SPEC:295	Testable=true	Section=PLT.20.2.1
[ccxciv] SPEC:296	Testable=true	Section=PLT.20.2.1
[ccxcv] SPEC:297	Testable=true	Section=PLT.20.2.1
[ccxcvi] SPEC:298	Testable=true	Section=PLT.20.2.2
[ccxcvii] SPEC:299	Testable=true	Section=PLT.20.2.4
[ccxcviii] SPEC:300	Testable=true	Section=PLT.20.2.4
[ccxcix] SPEC:301	Testable=true	Section=PLT.20.2.4
[ccc] SPEC:302	Testable=true	Section=PLT.20.2.5
[ccci] SPEC:303	Testable=true	Section=PLT.20.2.5
[cccii] SPEC:305	Testable=false(impl)	Section=PLT.21.2

[ccciii]SPEC:306	Testable= false	Section= PLT.23.2
[ccciv] SPEC:307	Testable= false	Section= PLT.23.2
[cccv] SPEC:308	Testable=false	Section= PLT.23.5
[cccvi] SPEC:309	Testable=true	Section=PLT.23.5(servlet spec)
[cccvii] SPEC:310	Testable=true	Section= PLT.24.2
[cccviii] SPEC:311	Testable=true	Section= PLT.24.2
[cccix] SPEC:312	Testable=true	Section= PLT.24.2
[cccx] SPEC:313	Testable=true	Section= PLT.24.4
[cccxi] SPEC:314	Testable=true	Section= PLT.24.4
[cccxii] SPEC:315	Testable= true	Section=PLT.26
[cccxiii] SPEC:316	Testable=true	Section= PLT.26.1
[cccxiv] SPEC:317	Testable=false	Section= PLT.26.1
[cccxv] SPEC:318	Testable=true	Section= PLT.26.2
[cccxvi] SPEC:319	Testable=true	Section= PLT.26.2
[cccxvii] SPEC:320	Testable=true	Section= PLT.26.2
[cccxviii] SPEC:321	Testable=true	Section= PLT.26.2
[cccxix] SPEC:322	Testable=true	Section= PLT.26.2
[cccxx] SPEC:323	Testable=true	Section= PLT.26.2
[cccxxi] SPEC:324	Testable=true	Section= PLT.26.2
[cccxxii] SPEC:325	Testable= true	Section=PLT.26.2
[cccxxiii] SPEC:326	Testable=false	Section= PLT.26.2
[cccxxiv] SPEC:327	Testable=false	Section= PLT.26.2
[cccxxv] SPEC:328	Testable=false	Section= PLT.26.2
[cccxxvi] SPEC:329	Testable=false	Section= PLT.26.2

[cccxxvii] SPEC:330	Testable=true	Section= PLT.26.3
[cccxxviii] SPEC:331	Testable=true	Section= PLT.26.3
[cccxxix] SPEC:332	Testable=true	Section= PLT.26.3
[cccxxx] SPEC:333	Testable=true	Section= PLT.26.3
[cccxxxi] SPEC:334	Testable=true	Section= PLT.26.3
[cccxxxii] SPEC:335	Testable=true	Section= PLT.26.3
[cccxxxiii] SPEC:336	Testable=true	Section= PLT.26.3
[cccxxxiv] SPEC:337	Testable= true	Section=PLT.26.3
[cccxxxv] SPEC:338	Testable=false	Section= PLT.26.3
[cccxxxvi] SPEC:339	Testable=false	Section= PLT.26.3
[cccxxxvii] SPEC:340	Testable=false	Section= PLT.26.3
[cccxxxviii] SPEC:341	Testable=false	Section= PLT.26.3
[cccxxxix] SPEC:342	Testable=true	Section= PLT.26.4
[cccxl] SPEC:343	Testable=true	Section= PLT.26.4
[cccxli] SPEC:344	Testable=true	Section= PLT.26.4
[cccxlii] SPEC:345	Testable=true	Section= PLT.26.4
[cccxliii] SPEC:346	Testable= true	Section=PLT.26.4
[cccxliv] SPEC:347	Testable=false	Section= PLT.26.4
[cccxlv] SPEC:348	Testable=false	Section= PLT.26.4
[cccxlvi] SPEC:349	Testable=true	Section= PLT.26.5
[cccxlvii] SPEC:350	Testable=true	Section= PLT.26.6
[cccxlviii] SPEC:351	Testable=false	Section= PLT.26.6
[cccxlix] SPEC:352	Testable=true	Section= PLT.26.6
[cccl] SPEC:353	Testable=true	Section= PLT.26.6