[Web Services for Remote Portlets Specification v2.0]

[Public Review Draft], [8 September 2006]

Artifact Identifier:

wsrp-2.0-spec-pr-02

Location:

TC: http://www.oasis-open.org/committees/wsrp

Current: http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html

This Version: http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec-pr-02.html

Artifact Type:

Specification

Technical Committee:

OASIS Web Services for Remote Portlets TC

Chair:

Rich Thompson, IBM Corporation <richt2@us.ibm.com>

Editor:

Rich Thompson, IBM Corporation <richt2@us.ibm.com>

Contributors:

Subbu Allamaraju, BEA Systems <subbu@bea.com>
Michael Freedman, Oracle Corporation <Michael.Freedman@oracle.com>
Richard Jacob, IBM Corporation <richard.jacob@de.ibm.com>
Andre Kramer, Citrix Systems <andre.kramer@eu.citrix.com>

OASIS Conceptual Topic Model Area:

Web Services

Abstract:

Integration of remote content and application logic into an End-User presentation has been a task requiring significant custom programming effort. Typically, vendors of aggregating applications, such as a portal, write special adapters for applications and content providers to accommodate the variety of different interfaces and protocols those providers use. The goal of this specification is to enable an application designer or administrator to pick from a rich choice of compliant remote content and application providers, and integrate them with just a few mouse clicks and no programming effort. This revision of the specification adds Consumer managed coordination, additional lifecycle management and a set of related aggregation enhancements.

This specification is the effort of the OASIS Web Services for Remote Portlets (WSRP) Technical Committee which aims to simplify the effort required of integrating applications to quickly exploit new web services as they become available.

This standard layers on top of the existing web services stack, utilizing existing web services standards and will leverage emerging web service standards (such as policy) as they become available. The interfaces defined by this specification use the Web Services Description Language (WSDL).

Status:

This document was last revised or approved by the WSRP TC on the above date. The level of approval is also listed above. Check the current location noted above for possible later revisions of this document. This document is updated periodically on no particular schedule.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at www.oasis-open.org/committees/wsrp.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (www.oasis-open.org/committees/wsrp/ipr.php).

The non-normative errata page for this specification is located at www.oasis-open.org/committees/wsrp.

Notices

Short Table of Contents

Table of Contents

1 Introduction

The Web Services for Remote Portlets specification defines a web service interface for accessing and interacting with interactive presentation-oriented web services. It has been produced through the efforts of the Web Services for Remote Portlets (WSRP) OASIS Technical Committee. It is based on the requirements gathered and on the concrete proposals made to the committee.

Scenarios that motivate WSRP functionality include:

- Content hosts, such as portal servers, providing Portlets as presentation-oriented web services that can be used by aggregation engines.
- Aggregating frameworks, including portal servers, consuming presentation-oriented web services offered by

content providers and integrating them into the framework.

Many of the non-portal use cases were originally defined by the Web Services for Interactive Applications (WSIA) OASIS Technical Committee. The WSIA use cases [1] have become input to the ongoing effort in this area by the WSRP OASIS Technical Committee. For additional details and documents, refer to the committee information available at http://www.oasis-open.org/committees/wsrp/ and http://www.oasis-open.org/committees/wsia/.

This specification accounts for the fact that **Producers** (web services conforming to this specification) and **Consumers** (applications consuming Producers in a manner conforming to this specification) may be implemented on very different platforms, be it as a **[J2EE]** based web service, a web service implemented on Microsoft's **[.Net]** platform or a Portlet published directly by a portal [A100]. Special attention has been taken to ensure this platform independence. At the same time, attention has also been paid to ensure a reasonable mapping to technologies for producing markup fragments (e.g. webparts, JSR 168 portlets, etc).

These web services are built on standard technologies, including **[SSL/TLS]**, **[URI/URL]**, **[WSDL]** and **[SOAP]**, and expects to leverage future applicable Web Service standards, such as WS-Policy [A102] in future versions.

1.1 Motivation

Portals and other Web applications render and aggregate information from different sources and provide it in a compact and easily consumable form to End-Users.

Among typical sources of information are web services. Traditional data-oriented web services, however, require aggregating applications to provide specific presentation logic for each of these web services. Furthermore, each aggregating application communicates with each web service via its unique interface. This approach is not well suited to dynamic integration of business applications and content as a plug-and-play solution.

This specification solves this problem by introducing a set of presentation-oriented web service interfaces that allow the inclusion of and interaction with content from a web service. Such a presentation-oriented web service provides both application logic and presentation logic. This specification provides a common protocol and a set of interfaces for presentation-oriented web services. Thus, aggregating applications can easily incorporate content from these web services using code which is not specific to the content source.

1.2 Actors

This specification uses four roles (End-User, Consumer, Producer and Portlet) to help delineate responsibilities. Implementations are free to combine multiple roles provided all resulting responsibilities are met.

The protocol defined by this specification describes the conversation between Producers and Consumers on behalf of End-Users (clients of the Consumer). Producers are presentation-oriented web services that host Portlets which are able to render markup fragments and process user interaction requests. Consumers use these web services as part of presenting markup to End-Users and managing the End-User's interaction with the markup.

1.2.1 Portlet

Portlets are hosted by Producer web services and generate markup as well as processing interactions with that markup. In general a Portlet includes both logic conforming to some specification of the Producer's environment and

a particular configuration of any settings or properties the Portlet exposes.

1.2.2 Producer

Producers are modeled as containers of Portlets. The Producer provides a set of web service interfaces, including:

- **Service Description:** A required interface that allows Consumers to find out the capabilities of the Producer web service and about the Portlets it hosts, including the metadata necessary for a Consumer to properly interact with each Portlet.
- **Markup:** A required interface used to request and interact with markup fragments.
- **Registration:** An optional interface used to establish a relationship between a Producer and a Consumer (e. g. for billing or book-keeping purposes).
- **Portlet Management:** An optional interface that grants access to the life-cycle of the hosted Portlets. This interface also includes **Property Management**, which enables programmatic access to a Portlet's enduring state (defined in **[Section 3.7])**.

In order to allow different levels of sophistication for both the Producer and Consumer, parts of this functionality are optional. Various examples of how a Producer might implement particular functionality for varying levels of sophistication and with regards to implementing some of the optional portions of the protocol are contained throughout this document.

The Producer optionally manages Consumer *registrations*. The Producer may require Consumers to register prior to discovering and interacting with Portlets. A registration represents a relationship (often including both technical and business aspects) between the Consumer and Producer which provides the scope for a set of interactions between them.

1.2.2.1 Portlet Management

A particular Portlet is identified with a `portletHandle`. The Consumer uses `portletHandles` throughout the communication to address and interact with Portlets through the Producer's web service interfaces. The Portlets a Producer publishes as available for all Consumers to interact with are called "Producer Offered Portlets". Producer Offered Portlets are pre-configured and not modifiable by Consumers.

If the Producer chooses to expose the *Portlet Management* interface, it is allowing Consumers to clone the Portlets offered by the Producer and customize those cloned Portlets, the details of which are discussed later. Such a uniquely configured Portlet is called a "Consumer Configured Portlet". Like Producer Offered Portlets, a `portletHandle` is used to address Consumer Configured Portlets. This `portletHandle` is both; 1) invariant until released and 2) unique within and scoped to the Consumer registration.

1.2.3 Consumer

A Consumer is an application that incorporates, in part or whole, an intermediary function that communicates with presentation-oriented web services (i.e. Producers and the Portlets they host) on behalf of its End-Users. It gathers and aggregates the markup delivered by the Portlets and other view components for presentation to the End-User. Because of this intermediary role, Consumers are often compared to "message switches" that route messages between various parties. One typical Consumer is a portal, which mediates the markup and the interaction with this markup between End-Users and presentation-oriented web services. Another typical Consumer is an e-Commerce application that aggregates manufacturer-provided content with its own content. Since the Consumer is an intermediary, aggregating system, the markup sent for display to the End-User and most interactions with that

markup flow through the Consumer. This often results in situations where the End-User implicitly trusts the Consumer to respect their privacy and security concerns with regards to this information flow. Additionally, the event distribution and public navigational state portions of this specification cause Consumers to also have the role of a coordination broker, providing inter-Portlet and End-User / Consumer to Portlet coordination.

While this specification is neutral as to the markup used to represent the user interface to the End-User, we note that general performance concerns favor markup technologies that push the processing of user interface logic, such as the validation of End-User input, as far toward the user agent as possible. Client-side scripting and XForms[2] represent technologies that can be leveraged to address these performance concerns. Note that use of such technologies does not relieve the need for a Portlet to validate the input data it receives.

1.2.4 End-User

The main purpose of a Consumer acting as a content intermediary for various Producer/Portlets is the preparation and presentation of aggregated markup to an End-User. In addition, the Consumer needs to manage the processing of interactions with that markup in order to properly correlate the interactions with the (potentially stateful) environment that produced the markup.

1.3 Typical Process Flow

While some of the following steps are optional, the typical flow of interactions between these actors is:

1. Consumer "discovers" the Producer. This involves the Consumer learning the URL of the web service end-point for the Producer and getting the Producer's metadata with its description of the registration requirements and possibly an indication of the Portlets the Producer is exposing.
2. Establishment of a relationship between the Consumer and Producer. This may involve the exchange of information regarding capabilities, security requirements or other business and/or technical aspects of the relationship.
3. Consumer learning the full capabilities and services of the Producer based on the now established relationship.
4. Establishment of a relationship between the Consumer and End-User. This permits the Consumer to authenticate the End-User and may allow the End-User to customize the aggregated pages presented by the Consumer.
5. Production of aggregated pages. This typically involves the Consumer defining some base level of page design (often with customized Portlets) and may involve further customization of those pages by the End-User.
6. Request for a page. This typically results when the End-User directs a user-agent (e.g. browser) to the Consumer's URL, but also occurs indirectly as a result of processing an interaction with the markup of a previous page.
7. Processing interactions. Some End-User interactions with the markup of a page will result in an invocation on the Consumer to provide some logical function. The Consumer will process this invocation to determine the Producer/Portlet that the interaction has targeted and the nature of the invocation requested for that Portlet. The Portlet's response may trigger the Consumer to mediate coordination activity among the Portlets. Since the resulting invocation of that Portlet is likely to change its state (and may also change the state of other Portlets), the Consumer must also treat this as an indirect request for a page and thereby loop back to step 6.
8. Destruction of relationships. Producers and Consumers may choose to end a registration relationship at any time, potentially due to the expiration of a lease on the item used to refer to the relationship. The protocol provides means by which the Producer and Consumer may inform each other that the relationship (or some portion of it) has ended and that related items may be cleaned up.

2 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]**.

Cross references to the **[Requirements]** developed by both the WSIA and WSRP technical committees are designated throughout this specification by a hyperlink to the requirement contained where the requirement number is enclosed in square brackets (e.g. [A100]).

Whenever this specification uses the prefix "wsrp:" in what appears to be a QName, it is referring to the URI `urn:oasis:names:tc:wsrp:v2:types`.

The normative definitions for all data structures defined by this specification are contained in the WSDL referenced in **[Section 15]**. For the convenience of the reader, the types are also declared in a non-normative manner just prior to the first use of the structure. These non-normative declarations use an IDL like syntax to describe the structures, where the leading [R] indicates a field is required and [O] indicates it is optional. These indicators are followed by an indication of the type for the field and then its name. If the type is one defined by this specification, then the type declaration also serves as a hyperlink to the definition of that type. All types not defined by this specification come from the XML Schema Datatype definitions in http://www.w3c.org/TR/xmlschema-2/. An example of a type being declared in this IDL like style is:

```
NewTypeDefinition
 [R] WSRPDefinedType someRequiredField
 [O] xsdDefinedType  someOptionalField
```

3 General Considerations

The major design goals for the protocol defined by this specification are simplicity, extensibility and efficiency. This specification seeks to accomplish these goals whenever possible by leveraging other standards. It also seeks to accomplish these goals in a manner that is neutral as to the platform any particular actor may be using to participate in interactions governed by this specification.

3.1 Related Standards

This specification seeks to leverage both existing and emerging web service standards whenever possible. The following are particularly noted as relevant standardization efforts:

3.1.1 Existing Standards

CC/PP - Defines syntax for describing a device's capabilities and user preferences.

Character Sets - Character set encoding

JSR168 - Java Community Process effort defining the Java Portlet Specification.

P3P - Defines how a Producer/Portlet may publish its privacy policy so that a Consumer could enforce End-User privacy preferences.

Namespaces - Defines how XML Namespaces are declared and used.

SAML - Defines how authentication and authorization information may be exchanged.

Schema - Defines how types are defined and associated with each other.

SOAP - Defines how to invoke web service interfaces.

SSL/TLS - Defines secure transport mechanisms.

URL - Defines URI (includes URL) syntax and encoding. We note that this is being superceded by the definition of an IRI (see http://www.ietf.org/rfc/rfc3987.txt), but are waiting for greater support in the marketplace before changing the descriptions to be IRI oriented. Those using IRIs locally will need to properly encode their IRIs into URIs during this transition period.

WSDL - Defines how abstract interfaces and their concrete realizations are defined.

WS-I.org - Has defined a base profile for use of the WSDL, SOAP and UDDI web services standards such that interoperability is maximized.

WS-Security - Defines how document level security standards apply to SOAP messages.

XACML - Defines syntax for expressing authorization rules.

XCBF - Defines how to exchange biometric data.

XML Digital Signatures - Defines how portions of an XML document are digitally signed.

XML Encryption - Defines how to encrypt/decrypt portions of an XML document.

XOP/MTOM - Defines how to move binary elements of an XML InfoSet into and out of separate parts of a multipart mime message such that the XML InfoSet may be transported efficiently.

3.1.2 Emerging Standards

JSR286 - Java Community Process effort defining v2 of the Java Portlet Specification.

WS-I.org - Defining additional profiles (e.g. Security) for use of web service standards such that interoperability is maximized.

WS-Notification - Defining how to discover, subscribe to and receive notifications using web services.

3.2 Foundations

As a specification that enables aggregating applications to use generic proxy code to easily integrate compliant web services, the foundations for the specification become critical. The text of this specification uses an IDL-like syntax to describe the interface in a non-normative manner. The ONLY normative description of the interface itself is contained in the WSDL referenced by **[Section 15]**. The textual portion of this specification does not contain normative statements regarding the exact syntax of XML passed between Consumer and Producer, but it does contain normative statements about the manner and order in which messages must be exchanged in order to be conformant. The chapters below are organized along the lines of the portTypes defined in the WSDL and the IDL descriptions of the data types reflect the normative schema definitions from the WSDL.

Since there are multiple ways to secure a message exchange between the Consumer and Producer (there are transport means such as SSL/TLS and message level means such as WS-Security) this specification uses the term "secure communication" to mean that at least one of these means will be used to secure messages. Note that this specification does not include mechanisms to reach agreement about which means to use as other efforts are addressing that issue in a manner that applies to all web services.

3.3 Data Objects

It is often necessary to pass data to operations. Typed data objects are defined as the transport mechanism wherever possible. The schema definitions of these structures includes the <any namespace="##other"/> construct as a standard means for data extensions. Producers/Portlets employing these extensions are encouraged to provide typing information for the extended data items [A505]. The preferred means for this typing information includes using the schema defined[3] "type" attribute to reference the correct schema on each such extension element, and use of either the Producer's WSDL (default), or the WSRP defined "wsrp-extra" (`urn:oasis:names:tc:wsrp:extra:v1`) namespace or a "schemaLocation" attribute as per standard schema usage to declare the details of all non-simple types. This allows Consumers to provide type checking outside of that done by typical interface layers. This specification introduces various data structures as they are needed for operations and has a list of all these data structures in Appendix C.

3.4 Lifecycles

"Lifecycle" is a term used to describe how items become available, are interacted with, and finally are destroyed. The two lifecycles included in this specification are:

Enduring: This lifecycle starts with an explicit operation to create the item and ends either via a lifetime expiring (when leasing is in use) or via an explicit operation to destroy the item. Examples include the `registrationHandle` and context of a Consumer Configured Portlets.

Transient: This lifecycle can either start with an explicit operation OR as a side effect of some other operation [A204]. The item created is transient and no explicit operation is required to destroy it. This specification generally includes an `expires` element when a transient item is created so that anything at the Consumer related to the item

may be reclaimed at an appropriate time. An example of items using this lifecycle is session creation.

3.5 Scopes

Scope is a term used to describe when something is valid. An item often scopes both the usage and lifecycle of other items. Scopes that are referenced in this specification are:

Registration scope: This scope is initiated when a Consumer registers with a Producer and ends when the handle referring to that registration is released. As such it encompasses any Portlets the Consumer configures and any interactions with the Portlets of the Producer. From the Producer's perspective, this scope has an enduring lifecycle. This scope is referenced throughout the protocol using a `registrationHandle`. The `registrationHandle` is created and destroyed using either the in band mechanism, i.e. by declaring support for the `Registration` portType, or by an out of band mechanism, whereby the `registrationHandle` is created and destroyed by means outside this specification [R354]. If a Producer supports the leasing feature, then this scope can also be ended in a scheduled manner.

Portlet scope: This scope is initiated for a Producer Offered Portlet when the Portlet is added to the set returned in the metadata of the Producer. This scope is initiated for a Consumer Configured Portlet when the Portlet is cloned and as such will be encapsulated by a registration scope, if one exists. This scope ends for Consumer Configured Portlets when the reference to the Portlet is explicitly released or, if the Producer supports the leasing feature, released in a scheduled manner. As such it encompasses all interactions with the Portlet. This scope has an enduring lifecycle and is referenced using a `portletHandle`. The Producer optionally exposes this scope by declaring support for the `PortletManagement` portType. If the Producer exposes the `PortletManagement` portType, then the Consumer can clone the Producer Offered Portlets and uniquely configure them for its own use. The Consumer can also choose to directly use the Producer Offered Portlets.

Session scope: This scope is initiated when a Portlet needs to store transient state on the Producer and is always encapsulated by the Portlet's scope. This scope ends when the session holding that state is released (either via an explicit operation on the Producer or via a timeout mechanism). As such it encompasses a set of operation invocations in which the Consumer has supplied the session's identifier. This scope has a transient lifecycle and is established by the Producer returning a new `sessionID`. The Consumer MUST respect this new session scope as described in **[Section 6.1.1]**.

3.6 Leasing

WSRP defines a number of items which can use a leasing concept (scheduled destruction of the item with opportunity for extending the scheduled termination time). All such items go through three stages of availability; namely:

- **Active:** A leased item in this stage is available for normal usage. The termination time on the lease refers to the time expected for the item to exit this stage.
- **Suspended:** A leased item in the suspended stage is not available for normal use, but is available for renewal. Renewal could be as simple as renewing the lease or as complicated as negotiating an updated business relationship. Producers are free to not support the suspended stage or to automatically move a suspended item to the active stage upon use.
- **Expunged:** An expunged item is no longer available for reference or renewal.

3.7 Types of Stateful Information

Because the WSRP protocol operates over connectionless technologies, the Producer must be able to return information to the Consumer, with the understanding that this information will be sent back to it [A200]. Three types of stateful information exist:

Navigational state: This is the state that allows the current page fragment to be correctly generated multiple times. Web applications typically store this type of state in the URL so that both page refresh and bookmarked pages will generate what the End-User expects. To supply the bookmarking and page refresh capabilities End-Users expect, the Consumer MAY store this type of state, or a reference to it, in the URL.

Transient state: This is state that applies to a restricted set of operations. This specification defines two kinds of transient state; namely:

- **Interaction State:** This opaque state is supplied to the processing of an interaction with a Portlet's markup and is often used as the equivalent of input parameters to that processing.
- **Session State:** This state is stored on the Producer and is related to a sequence of operations (for example, an e-Commerce site may store a shopping cart in its session state). Once a session is generated, the Producer returns a reference to it and the Consumer must return this reference on future invocations (as described in **[Section 6.1.2]**) in order for a Portlet to reconnect to this type of state and process End-User interactions in an expected manner.

Enduring state: This is state that has an enduring lifecycle, that is, exists until either the Consumer or Producer explicitly discards it, potentially as part of the expiration of a lease. This specification defines two kinds of enduring state with each referred to via a handle that MUST remain invariant once the Producer supplies it to the Consumer:

- **Consumer Registration:** Represents a relationship between a Consumer and Producer (also a registration scope). Data that is part of the Consumer registration state impacts all invocations within the scope of the registration (see **[Section 5.1.27]**). The opaque reference to Consumer registration state is referred to as a `registrationHandle`.
- **Portlet:** In addition to the Portlets a Producer offers for all Consumers to use, the ability of a Consumer to create a unique configuration of one of those Portlets for its own use is defined. The opaque reference to a configured Portlet is referred to as a `portletHandle` (also correlates to a Portlet scope), see **[Section 6.1.4]**.

While each of these types of state can have portions which are opaque to the Consumer, each can also have portions which are described to the Consumer such that programmatic impacts, such as state-based coordination, can be effected by the Consumer.

3.8 Statefulness

This specification does not mandate that either the Producer or the Consumer is stateful [A201]. In the **getMarkup**, **handleEvents** and **performBlockingInteraction** calls, the `navigationalContext` field carries the state necessary for the Portlet to render the current markup to be returned to the Consumer. This enables the Consumer to reasonably support page refresh and bookmarking by the End-User. In order to support the URL length restrictions of some browsers, the Consumer may need to cache the navigational type of state and just supply a reference to it on the URL. If the Producer utilizes local state, storing this state in an implementation-dependent manner, then it will return a `sessionID` to the Consumer for use during the lifetime of the session.

If the Consumer is operating in a stateless manner, then it may choose the way to achieve this. In the case of HTTP transport the Consumer may employ standard HTTP mechanisms (cookies or URL-rewriting) to propagate the

navigational state or `sessionID` out to its client. If operating in a stateful manner, the Consumer may employ any number of persistence/caching mechanisms [A202].

The nature of the conversation between the client and the Consumer, for purposes of this section, is out of scope [A304]. This does not mean that information about the client, including user profile data, is opaque to the Producer. There are many use cases for which user identity must be conveyed to the Producer [A501] [A606].

3.9 Producer Mediated Data Sharing

Producers may implement data sharing mechanisms through techniques such as a shared area within sessions for Portlets to use. The Producer indicates which Portlets share such data areas via the `groupID` parameter in the Portlet metadata. **[Section 5.1.20]** specifies how the Consumer is to process Producer's grouping, which are defined by the groupID parameters.

Shared data areas introduce implementation challenges in clustered environments. In such an environment, multiple concurrent requests may be routed to different cluster nodes. The Producer must ensure that Portlets with a common shared data area have access to the shared data even in such situations.

3.10 Consumer Mediated Coordination

Consumers may implement mechanisms through which Portlets, and any other view components the Consumer aggregates, react in a coordinated manner. Each of these mechanisms enables event-driven Consumer applications. This specification defines two such mechanisms:

1. **Event Distribution**: This mechanism provides for Portlets generating and consuming events that carry both semantic meaning, through their name, and data, in their payload. In addition, Consumers can also generate events for distribution to Portlets. All events are distributed by the Consumer such that any current Portlet state which is stored at the Consumer can be supplied along with the event to the Portlet. This also allows the Consumer to apply whatever policies it chooses to control the event distribution.
2. **State Distribution**: This mechanism allows Portlets to expose portions of its state which the Consumer can then manipulate and/or transfer to other Portlets or Consumer constituents. This specification defines a state distribution mechanism for navigational state. This mechanism exposes a portion of the Portlet's navigational state to the Consumer such that coordinated updates across multiple Portlets can be effected (see **[Section 6.1.12]**).

3.11 Information Passing Mechanisms

All information passing enabled by this specification is between exactly one Producer and one Consumer. Implementation of data sharing, including both policy and side effects, within a particular Producer service is outside the scope of this specification.

3.12 Three-step protocol

This specification attempts to account for both isolated interactions between a Consumer and a Producer, and also those interactions that may cause state changes in other Portlets the Consumer aggregates from the same Producer [A503]. Common causes of such shared state include use of a common backend system (e.g. database) and Producer-mediated data sharing. In addition, the Consumer can selectively distribute events, including ones it

generates itself, among the Portlets. For these reasons, there is a "three-step" capability built into the protocol. Note that this is an extension of the WSRP v1 two-step protocol as the additional step, event distribution, is entirely optional. This enables Consumers to provide coordinated cross-portlet response to the End-User's interaction while providing the equivalent user experience as WSRP v1 for those Consumers not supporting event distribution [C409].

The three non-overlapping steps of the protocol are:

1. (optional) The Consumer invokes **performBlockingInteraction** on the Portlet whose markup the End-User interacted with. The Consumer MUST NOT invoke operations on any Portlets within the context of the initiating request from the client of the Consumer until either the receipt of a response or the invocation of **performBlockingInteraction** fails (e.g. times out).
2. (optional) The Consumer has the option of distributing events among the Portlets using **handleEvents**. Note that the Consumer could distribute multiple events to a Portlet in this step (see **[Section 6.4.2.1]**), including via multiple concurrent invocations [C408]. The Consumer MUST NOT begin to gather markup until it considers all Portlets to have finished the event distribution step.
3. The Consumer invokes **getMarkup** on the Portlets being aggregated.

Examples of when one of the optional steps (**performBlockingInteraction** and **handleEvents**) might not be used include:

- The End-User interacting with a URL that simply looks to render the Portlet's markup with a different navigational state (e.g. with the next set of results from a search). In this case, both of the first two steps could be skipped.
- The initial page construction for the End-User. This may involve the Consumer distributing events (step two) in order to initialize a Portlet's state, but should not involve the first step as the End-User has not interacted with the Portlet's markup (Note that a bookmarked page could be an exception to this, but that a bookmarked interaction involving step one of the protocol is unlikely to have the same impact the interaction had during the session when the bookmark was generated).

Interaction semantics are well-defined across the spectrum of interaction styles supported in the protocol. In other words, the results of the Consumer invoking **performBlockingInteraction** on a Portlet and distributing events among the Portlets using **handleEvents**, regardless of whether those interactions have side effects on other Portlets at the Producer, is well-defined independent of the order of **getMarkup** invocations on the Portlets.

3.13 Transport Issues

Since the transport layer is often used to store various pieces of information (e.g. J2EE load balancing depends on a session cookie and HTTP transport), and these pieces of information often will pertain to a client session with the Consumer rather than the Consumer itself, Consumers that manage transport layer issues, such as cookies, MUST return them to the Producer only for subsequent invocations within the Markup Interface during the same client session. In addition, any supplying of cookies to resources the Portlet references needs to be in conformance with the rules established by RFC2109[4]. Not scoping their return in this manner will likely result in a loss of privacy for the End-User and unexpected behavior in general. Failure to return them for this full duration will often result in a loss of state at the Producer and unexpected behavior for the End-User. We also note that failure to properly do this management will eliminate the ability to use Producers that set `requiresInitCookie` to a value other than "`none`".

3.14 Load Balancing

Load balancing is a part of the Producer environment that cannot easily be managed from within the protocol. Load balancing is highly dependent on mechanisms in the transport, for example the use of cookies in HTTP. In order to permit load balancing to function, regardless of the transport binding in use, the Consumer needs to manage transport level issues itself. Using HTTP as an example, if the Producer requires such support of Consumers, it MUST indicate so by setting the `requiresInitCookie` metadata to a value other than "`none`". If the Producer set `requiresInitCookie` to a value other than "`none`", the Consumer MUST ensure that cookies are properly supplied in subsequent requests for the End-User.

---

4 Interface Overview

This specification defines four interfaces whose operations have the following signatures:

4.1 Service Description Operations

The Service Description interface, a required interface, defines an operation for acquiring the Producer's metadata.

```
ServiceDescription = getServiceDescription (registrationContext, desiredLocales,
portletHandles, userContext );
```

4.2 Markup Operations

The Markup interface, a required interface, defines operations for getting the markup from a Portlet as well as processing user interactions with that markup. This interface also contains the operation for Consumer assistance in pre-initializing HTTP cookies. Having this operation in this interface avoids the problems associated with moving cookies between bindings.

```
MarkupResponse = getMarkup (registrationContext, portletContext, runtimeContext, userContext,
markupParams);
```

```
ResourceResponse = getResource (registrationContext, portletContext, runtimeContext,
userContext, resourceParams );
```

```
BlockingInteractionResponse = performBlockingInteraction (registrationContext, portletContext,
runtimeContext, userContext, markupParams, interactionParams);
```

```
HandleEventsResponse = handleEvents (registrationContext, portletContext, runtimeContext,
userContext, markupParams, eventParams);
```

```
ReturnAny = initCookie (registrationContext,
userContext);
```

```
ReturnAny = releaseSessions (registrationContext, sessionIDs[],
userContext);
```

4.3 Registration Operations

The Registration interface, an optional interface, defines operations for establishing, updating and destroying a registration. Each registration reflects a particular relationship between a Consumer and a Producer.

```
RegistrationContext = register (registrationData, lifetime,
userContext);
```

```
RegistrationState = modifyRegistration (registrationContext, registrationData,
userContext);
```

```
ReturnAny = deregister (registrationContext,
userContext);
```

```
Lifetime = getRegistrationLifetime (registrationContext,
userContext);
```

```
Lifetime = setRegistrationLifetime (registrationContext, userContext,
lifetime);
```

4.4 Portlet Management Operations

The Portlet Management interface, an optional interface, defines operations for getting Portlet metadata, cloning Portlets for further customization and interacting with the property interface.

```
PortletDescriptionResponse = getPortletDescription (registrationContext, portletContext,
userContext, desiredLocales);
```

```
PortletContext = clonePortlet (registrationContext, portletContext, userContext,
lifetime);
```

```
DestroyPortletsResponse = destroyPortlets (registrationContext, portletHandles[],
userContext);
```

```
GetPortletsLifetimeResponse = getPortletsLifetime (registrationContext, portletContext[],
userContext);
```

```
SetPortletsLifetimeResponse = setPortletsLifetime (registrationContext, portletContext[],
userContext, lifetime);
```

```
CopyPortletsResponse = copyPortlets (toRegistrationContext, toUserContext,
fromRegistrationContext, fromUserContext, fromPortletContexts[], lifetime);
```

```
ExportPortletsResponse = exportPortlets (registrationContext, portletContext[], userContext,
lifetime, exportByValueRequired);
```

```
ImportPortletsResponse = importPortlets (registrationContext, importContext, importPortlet[],
userContext, lifetime);
```

```
ReturnAny = releaseExport (exportContext,
userContext);
```

```
Lifetime = setExportLifetime (registrationContext, exportContext, userContext,
lifetime);
```

```
PortletContext = setPortletProperties (registrationContext, portletContext, userContext,
propertyList);
```

```
PropertyList = getPortletProperties (registrationContext, portletContext, userContext,
names);
```

```
PortletPropertiesDescriptionResponse = getPortletPropertyDescription (registrationContext,
portletContext, userContext, desiredLocales);
```

---

5 Service Description Interface

A Producer may be discovered through mechanisms such as **[UDDI]** or **[ebXML Registry]**, which also provide information concerning the capabilities of the service. Other discovery mechanisms (e.g. emailed URL to a properly enabled user-agent) do not expose these capabilities. The **getServiceDescription** operation provides a discovery mechanism-agnostic means for a Consumer to ascertain a Producer's or Portlet's capabilities [A110]. This interface is required of all Producers to provide a well-defined means for Consumers to ascertain the requirements to register or use the Producer.

5.1 Data Structures

The following data structures are needed by this interface:

5.1.1 Extension Type

The `Extension` structure contains the payload extension mechanism for vendor and application extensions. These arbitrary elements are required to be from namespaces other than the WSRP "types" namespace (urn:oasis:names:tc:wsrp:v2:types) and extend their containing data structure. They are designed to communicate extended information between the Consumer and Producer. Consumers and Producers SHOULD NOT rely on receiving back any extensions passed to or returned from an invocation. Each such extension element carries a single child element which MUST declare its type using the schema-defined "type" attribute [5]. It is RECOMMENDED extensions be of type `xsd:string` (where xsd stands for http://www.w3c.org/2001/XMLSchema) , or be of a type from the WSRP-defined "wsrp-extra" namespace (urn:oasis:names:tc:wsrp:extra:v1) or be of a type defined in the Producer's WSDL as this enables Consumers to prepare an appropriate serializer/deserializer. We expect many extensions will be of the type `QNamedString` or `QNamedStringArray` (from the "wsrp-extra" namespace) and encourage their use in this manner. The other option is for each message to connect the extension to a type declared in a schema using the "schemaLocation" attribute as used by schema. Consumers and Producers are not required to process information supplied using these extension elements.

```
Extension
 [0] Object any
```

Members:

- `any`: This field has a schema declaration that allows any elements from namespaces other than WSRP. While the element definitions for these extensions are required to be in a namespace other than the WSRP types namespace, the use of the types defined within the XML Schema and WSRP "wsrp-extra" namespace is encouraged as this increases the likelihood of the receiving party being able to deserialize the extension in a highly typed manner. Overlap with the fields defined in the containing structure SHOULD be avoided.

5.1.2 Handle Type

Handles are opaque references that are passed between the Consumer and Producer.

Handles are represented as restricted strings in the protocol. Although a string is principally unlimited in length, the length of the handle is restricted for the following reasons:

- Handles may be stored in databases and may be used for indexing.
- The Consumer will likely embed handles in client URLs.
- Comparison of handles should be efficient.

The maximum length of a handle is restricted to 255 characters. It is strongly RECOMMENDED these characters be chosen from the first 127 characters of the Unicode character set so that it is feasible to represent the value in no more than 255 bytes of storage. Not following this recommendation will likely cause information to be lost as the Consumer stores and retrieves the value. The Consumer MAY truncate longer handles to 255 characters.

```
Handle restricts string (maximum length =
255)
```

5.1.3 Key Type

Keys are similar to Handles except that they are not opaque references. They are used for keying data and therefore need to support efficient comparisons. As a result their length is restricted to 255 characters. We STRONGLY RECOMMEND these characters be chosen from the first 127 characters of the Unicode character set so that it is feasible to represent the value in no more than 255 bytes of storage. Not following this recommendation will likely cause information to be lost as the value is stored and retrieved.

```
Key restricts string (maximum length =
255)
```

5.1.4 ID Type

IDs are used to refer to something, but are unlikely to be used as keys. As a result the length restriction is relaxed to 4096 characters. We STRONGLY RECOMMEND these characters be chosen from the first 127 characters of the Unicode character set so that it is feasible to represent the value in no more than 4096 bytes of storage. Not following this recommendation will likely cause information to be lost as the value is stored and retrieved. Those originating an ID are encouraged to keep them as small as possible relative to impacts on the other party's performance when storing large numbers of these (e.g. a sessionID is per user per Portlet and therefore a Consumer is likely to store a very large number of them).

```
ID restricts string (maximum length =
4096)
```

5.1.5 LocalizedString Type

The `LocalizedString` structure describes both the value for a particular locale and the name that can be used to extract the value for other locales from a `ResourceList`.

```
LocalizedString
```

```
 [R] string xmlLang
 [R] string value
 [O] string resourceName
```

Members:

- `xmlLang`: The locale for this supplied localized value. This is carried in the WSDL using the `xml:lang` attribute.
- `value`: The value for this localized string in the declared locale.
- `resourceName`: The name assigned to this localized string for dereferencing into a `ResourceList` for values from other locales. When the `resourceName` is not supplied, there are no values for additional locales available in the `ResourceList`.

5.1.6 ResourceValue Type

This structure provides the value of a resource for a locale.

```
ResourceValue
 [R] string    xmlLang
 [R] string    value
 [O] Extension extensions[]
```

Members:

- `xmlLang`: The locale for this localized value. This is carried in the WSDL using the `xml:lang` attribute.
- `value`: The value for this localized string in the declared locale.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.7 Resource Type

The `Resource` structure carries the values for a resource in a set of locales.

```
Resource
 [R] string        resourceName
 [R] ResourceValue values[]
 [O] Extension     extensions
```

Members:

- `resourceName`: The name of the resource for which this is a list of localized values.
- `values`: Each member of this array provides the value for the resource in a locale.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.8 ResourceList Type

This is an array of `Resource` structure, each of which carries the values for a localized resource in various locales.

ResourceList		
[R]	Resource	resources[]
[O]	Extension	extensions[]

Members:

- `resources:` Each member of this array provides the localized values for a resource.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.9 ItemDescription Type

This structure is used to describe custom items a Consumer is allowed to use when interacting with the Portlets at the Producer.

ItemDescription		
[R]	string	itemName
[R]	LocalizedString	description
[O]	Extension	extensions[]

Members:

- `itemName:` The name for this item. The preferred form is a URI such that it is definitively namespaced.
- `description:` A localized, free form description of the item. Expected use of this field is for display at the Consumer to someone who will provide a mapping to Consumer information.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.10 MarkupType Type

The `MarkupType` data structure is used to carry Portlet metadata that is `mimeType` specific.

MarkupType		
[R]	string	mimeType
[R]	string	modes[]
[R]	string	windowStates[]
[O]	string	locales[]
[O]	Extension	extensions

Members:

- `mimeType:` A mime type supported by the Portlet (e.g. *text/html, application/xhtml+xml, text/vnd.wap.wml*) for which the remainder of this structure applies. In addition to these fully specified mime types, use of "*" (indicates all mime types are supported) and type/* (where type includes things such as "text") from the

HTTP definition [6] MAY be specified. The Consumer does not have to process any optional parameters that can be included on mime type declarations.

- `modes`: The `modes` (defined in **[Section 6.9]**) that are supported by the Portlet for this `mimeType`. Possible values are:
  - Those defined by this specification in **[Section 6.9]**;
  - Modes supplied by the Consumer in `consumerModes`;
  - Any custom modes the Producer wishes to advertise as valid (see **[Section 6.9.5]**.

  Localized descriptions are only needed for the last category since the first two do not require manual interpretation.
- `windowStates`: The `windowStates` (defined in **[Section 6.10]**) that are supported by the Portlet for this `mimeType`. Possible values are:
  - Those defined by this specification in **[Section 6.10]**;
  - Window states supplied by the Consumer in `consumerWindowStates`;
  - Any custom window states the Producer wishes to advertise as valid (see **[Section 6.10.5]**.

  Localized descriptions are only needed for the last category since the first two do not require manual interpretation.
- `locales`: An optional array of locales or which this `mimeType` is available (e.g. "en-US"). If this array is not supplied, the Consumer can assume the Portlet will attempt to generate markup for any requested locale. Note that current practice on the Internet uses the format [2 char language code][7] "-" [2 char country code][8] as per the provided example with the full definition being found in RFC3066 and its successors. Values only using a two character language code mean that the Portlet is willing to generate markup of this type for any locale starting with the specified language code.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.11 EventDescription Type

The `EventDescription` structure provides the information needed to describe a Portlet's events.

EventDescription		
[R]	QName	name
[O]	QName	aliases[]
[O]	QName	type
[O]	anyURI	schemaLocation
[O]	ModelTypes	schemaType
[O]	LocalizedString	description
[O]	LocalizedString	hint
[O]	LocalizedString	label
[O]	Extension	extensions

Members:

- `name`: A namespaced name for the event being described. If this event's payload is not carried within the `NamedStringArray` alternative of the `EventPayload` structure, this name also becomes the XML element name carrying the payload with the type field defining the type of the element.
- `aliases`: An array of the QNames of events known to be semantically equivalent to this event. While this can provide a hint to the Consumer concerning potential items that could be correlated with this event, it

remains the Consumer's responsibility do any transformations required to produce the described event.

- `type`: A reference to a schema-defined payload the event will carry at runtime. If the `type` is not supplied, the Consumer SHOULD treat this as an opaque event payload. Note that this includes those events which are signals and therefore carry no payload.
- `schemaLocation`: This optional field carries a URI which resolves into a schema containing the `type` for the described `Event`. If a URI can not be provided for resolving the type definition, the `schemaType` field in the will need to contain the definition for the `type`.
- `schemaType`: This field can carry a schema defining the type for the described `Event` for those cases where a URI which resolves to a schema defining the type is not available.
- `description`: A localized, free form description of the event. Expected use of this field is for display at the Consumer to someone who will choose how this event is mapped to other events.
- `hint`: A relatively short description of the event. Intended for display (for example, as a tooltip) in any Consumer-generated user interface for processing/distributing the event.
- `label`: A short, human-readable name for the event. Intended purpose is for display in any Consumer-generated user interface for processing/distributing the event.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

Since an event's name can be referred to in a wildcard fashion (see **[Section 5.1.16]**), Portlet developers are encouraged to organize their event's local names in a hierarchical manner and MUST use the '.' character to delimit levels within the hierarchy. An example of such an organization would be events carrying changed address information on an application being organized as "applicant.address.streetChanged", "applicant.address.cityChanged", etc. Such an organization would allow another Portlet's metadata to simply say that it is interested in all events with names in the "applicant.address" branch of the hierarchy by specifying an interest in "applicant.address.". The trailing '.' tells the Consumer that this is not the end of the hierarchy and the the Portlet is interested in all events with names in this branch of the hierarchy.

5.1.12 PropertyDescription Type

Each property of a Portlet is described using the following structure.

PropertyDescription	
[R] QName	name
[R] QName	type
[O] anyURI	schemaLocation
[O] LocalizedString	description
[O] LocalizedString	label
[O] LocalizedString	hint
[O] string	usage[]
[O] QName	aliases[]
[O] Extension	extensions

Members:

- `name`: Name of the property being described. If this property's value is not carried within the `stringValue` alternative of the `Property` structure, this name also becomes the name of the XML element carrying the property's value with the type field defining the type of this element.
- `type`: Type of the property, using a namespace qualified name. We would encourage these to either be from

the set of schema-defined types or be explicitly typed in the schema element of an enclosing `ModelDescription`. This allows the Consumers to prepare the appropriate serializer/deserializer. The namespace for the schema-defined types used by this specification is http://www.w3c.org/2001/XMLSchema. Producers can assume that all Consumers support the basic types defined by http://www.w3c.org/TR/xmlschema-2/.

- `schemaLocation:` This optional field carries a URI which resolves into a schema containing the `type` for the described `Property`. If a URI can not be provided for resolving the type definition, the field of type `ModelTypes` in the enclosing structure will need to contain the definition for the `type`.
- `description:` A localized, free form description of the property. Expected use of this field is for display at the Consumer to someone who will choose how this property is mapped to other properties/data.
- `label:` A short, human-readable name for the property. Intended purpose is for display in any Consumer-generated user interface for administering the Portlet.
- `hint:` A relatively short description of the property. Intended for display, for example, as a tooltip in any Consumer-generated user interface for editing the property.
- `usage:` An array of values where the following definitions providing an initial set of values. We encourage those adding values to this extensible set to use qualified names in order to reduce instances of name clashes:
  - `wsrp:nonmodifiable:` This capability means the party using the property is not allowed to change the property [A605]. If this term is not specified, the party using the property may presume the right to change the property's value.
  - `wsrp:required:` This capability means that the party using the property has to supply a value for this property. If `wsrp:required` is not specified, the party using the property may choose whether or not to supply a value.
- `aliases:` An array of the QNames of properties known to be semantically equivalent to this property. While this can provide a hint to the recipient concerning potential items that could be correlated with this property, it remains the recipient's responsibility to do any transformations required to produce the described property.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.13 ModelTypes Type

The `ModelTypes` structure contains the payload mechanism for declaring the types referenced by the description types.

```
ModelTypes
 [R] Object any[]
```

Members:

- `any:` This field has a schema declaration that allows any elements from the schema namespace.

5.1.14 ModelDescription Type

The set of properties of a Portlet are described in its metadata using the following structure.

```
ModelDescription
 [O] PropertyDescription propertyDescriptions[]
```

[O]	ModelTypes	modelTypes
[O]	Extension	extensions

Members:

- `propertyDescriptions`: An array of property descriptions.
- `modelTypes`: A container for type definitions for the properties of this model. It is expected that XML schema will commonly be used to define Portlet-specific datatypes referenced in the `propertyDescriptions`.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.15 ParameterDescription Type

Portlet parameters are modeled as an array of strings, such that type information does not need to be declared, and are described using the following structure.

ParameterDescription		
[R]	string	identifier
[O]	QName	names[]
[O]	LocalizedString	description
[O]	LocalizedString	label
[O]	LocalizedString	hint
[O]	Extension	extensions

Members:

- `identifier`: A Producer defined identifier for this parameter which is REQUIRED to be unique for the `portletHandle` providing this `ParameterDescription`. The purpose of this identifier is enabling concise references to the parameter on URLs (see **[Section 10.2.1.3]**).
- `names`: Qualified names specifying the semantics supported by the parameter being described. The purpose of these names is assisting Consumers doing state-based coordination to properly associate items based on their semantics.
- `description`: A localized, free form description of the parameter. Expected use of this field is for display at the Consumer to someone who will choose how this parameter is mapped to other parameters.
- `label`: A short, human-readable name for the parameter. Intended purpose is for display in any Consumer-generated user interface for administering the Portlet.
- `hint`: A relatively short description of the parameter. Intended for display, for example, as a tooltip in any Consumer-generated user interface for editing the parameter.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.16 PortletDescription Type

The `PortletDescription` structure contains a set of fields that provide the metadata to describe the Portlet as well as any clones of the Portlet.

PortletDescription

[R]	Handle	portletHandle
[R]	MarkupType	markupTypes[]
[O]	ID	groupID
[O]	LocalizedString	description
[O]	LocalizedString	shortTitle
[O]	LocalizedString	title
[O]	LocalizedString	displayName
[O]	LocalizedString	keywords[]
[O]	ID	portletID
[O]	QName	publishedEvents[]
[O]	QName	handledEvents[]
[O]	ParameterDescription	navigationalParameterDescriptions[]
[O]	string	userCategories[]
[O]	string	userProfileItems[]
[O]	boolean	usesMethodGet
[O]	boolean	defaultMarkupSecure
[O]	boolean	onlySecure
[O]	boolean	userContextStoredInSession
[O]	boolean	templatesStoredInSession
[O]	boolean	hasUserSpecificState
[O]	boolean	doesUrlTemplateProcessing
[O]	boolean	mayReturnPortletState
[O]	Extension	extensions

Members:

- `portletHandle`: The handle by which Consumers can refer to this Portlet. Note that Handles are restricted to a maximum length of 255 characters.
- `markupTypes`: Each member of this array specifies metadata for a single `mimeType`.
- `groupID`: Identifier for the group within which the Producer places this Portlet or any Portlets derived from it via the cloning processes.
- `description`: Localized descriptions of the Portlet. This is intended for display in selection dialogs, etc.
- `shortTitle`: Localized short title for the Portlet.
- `title`: Localized title for the Portlet. This value is intended for display in a titlebar decoration for the Portlet's markup.
- `displayName`: Localized value intended for display in a Consumer's tooling for building aggregated page. In general this value is shorter than either title, though the available title values can be used as a default if this value is missing.
- `keywords`: Array of localized keywords describing the Portlet which can be used for search, etc.
- `portletID`: A Portlet assigned identifier which is invariant across deployments of compatible versions of the Portlet. Note that this breadth of invariance can only be achieved via an identifier assigned during the packaging of the Portlet for distribution to those who would deploy it.
- `publishedEvents`: This array provides the event names for the events the Portlet could generate. Consumers doing event distribution are REQUIRED to match 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. An example of where this is useful is when a set of events have been declared in an external source and this Portlet could generate a particular subset of those events (e.g. "applicant.address."). Note

that this wildcarding only applies to the local portion of the event name. While Portlets are allowed to generate events that are not described in this array, portlet developers choosing to not declare what events could be generated should carefully consider scenarios involving Consumers where events are only distributed if the page designer (i.e. the Consumer application developer) explicitly says to distribute an event generated by one Portlet to another set of Portlets. The information in non-declared events will not be distributed by such Consumers since the page designer was unaware of them.

- `handledEvents`: This array provides the event names the Portlet is willing to process. Each name specified is allowed to end with a "." character to indicate the Portlet is willing to process any event whose name starts with the characters before the "." character. Consumer responsibilities are the same as the `publishedEvents` field. An example of where this is useful is when a set of events have been declared in an external source and this Portlet could handle a particular subset of those events (e.g. "applicant.address.").
- `navigationalParameterDescriptions`: This array describes the Portlet's public or exposed navigational state [A504] [A505] [A506] [C407]. For a more complete description of navigational state see **[Section 6.1.12]**. Each state value is uniquely defined by the identifier field within the description. The `navigationalParameterDescriptions` array MUST NOT contain two descriptions with the same `identifier`.
- `userCategories`: An array of category names for the Producer's user categories which the Portlet supports. Each of these user categories has to have a `ItemDescription` available to the Consumer through the Producer's `ServiceDescription`. [R416]
- `userProfileItems`: An array of strings that enumerate what portions of the `UserContext` structure the Portlet needs to provide full functionality. For the fields this specification defines, the named profile items a Portlet uses MUST all come from the "Profile Name" column of the table found in **[Section 11]**. Any use of additional userProfile items specified as available when the Consumer registered SHOULD use the names the Consumer supplied. Any additional items specified SHOULD be interpreted by the Consumer as additional items the Portlet could use if the Consumer is able to supply the data.
- `usesMethodGet`: A flag indicating the Portlet generates markup that uses method=get in an HTML form. If the Consumer uses a Portlet which specifies `usesMethodGet` as "true", the Consumer MUST format its URLs in a manner that keeps user-agents from throwing away information (see **[Section 10.2.4]** for a description of the difficulties in using forms with method=get). The default value of this flag is "`false`".
- `defaultMarkupSecure`: Flag that indicates whether this Portlet requires secure communication on its default markup. This flag applies to all markup not generated as a direct result of an End-User interaction. The default value for this flag is "false".
- `onlySecure`: Flag that indicates whether this Portlet requires secure communication on all its markup [R403]. The intent of this flag is to allow Consumers to treat the Portlet specially because of this characteristic. The default value for this flag is "false".
- `userContextStoredInSession`: A flag indicating the Portlet will store any supplied `UserContext` in the current session. Setting this flag to "`true`" allows the Consumer to optimize when the `UserContext` is included on operation invocations. Since some data in the `UserContext` is sensitive, many Consumers will require that secure communication be used when the information is passed [A610]. Not requiring this of all invocations can result in a significant performance difference. Note that the Consumer MAY send `UserContext` information on any invocations as a replacement for information the Portlet MAY be storing in a session. The default value of this flag is "`false`".
- `templatesStoredInSession`: A flag indicating the Portlet will store any supplied `templates` in the current session. Setting this flag to "`true`" allows the Consumer to optimize when the `templates` structure is set in `MarkupParams`. Since the content of the `templates` structure can get quite large, not requiring it to be passed can result in a significant performance difference. Note that the Consumer MAY send templates on any invocations as a replacement for information the Portlet MAY be storing in a session. The default value of this flag is "`false`".
- `hasUserSpecificState`: A flag indicating the Portlet will store enduring state specific to each End-User. Setting this flag to "`true`" suggests to the Consumer to clone the Portlet when placing it on an aggregated

page rather than waiting for the processing described in **[Section 6.4.3]**. The default value of this flag is "`false`".

- `doesUrlTemplateProcessing`: A flag indicating the Portlet will process any `templates` supplied so as to correctly write URLs in its markup. For Portlets setting doesUrlTemplateProcessing to "`true`", Consumers MUST provide the URL writing templates and `namespacePrefix` field. The default value of this flag is "`false`".
- `mayReturnPortletState`: A flag indicating the Portlet might return at least a portion of its enduring state to the Consumer. The default value of this flag is "`false`".
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.17 Property Type

The `Property` data structure is used to carry typed information between the Consumer and the Producer. Each property includes a name and type (carried using the `xsi:type` attribute) [A505] [A507]. The primary reason for including type information is situations when the receiving party does not have a relevant `PropertyDescription`.

```
Property
 [R] QName  name
 [O] QName  type
 [O] string xmlLang
 [O] Object value[]
```

Members:

- `name`: Name of the property.
- `type`: A reference to a schema-defined type for the property's payload. If a `PropertyDescription` was supplied with a matching name, it is an error for this type to not match the type supplied in that `PropertyDescription`.
- `xmlLang`: The optional locale for the supplied localized value. This is carried in the WSDL using the `xml:lang` attribute and is optional as some, though not all, values are likely to be localized.
- `value`: The property's value. The type information needed to properly serialize / deserialize this value is carried in the relevant `PropertyDescription`. Note that the WSDL from **[Section 15]** defines two means by which this field may be sent, either as a generic array of elements or as a single element with a type of string. This second choice was added as many properties are likely to be of this type and it allows the web stack to automatically do the (de)serializing to the wire format.

5.1.18 ResetProperty Type

The `ResetProperty` data structure carries the name of a Property for which the Consumer wants the value reset to the default.

```
ResetProperty
 [R] QName name
```

Members:

- `name`: Name of the property whose value is to be reset; MUST have a non-zero length.

5.1.19 PropertyList Type

A `PropertyList` gathers a set of Property structures together for transmitting between the Consumer and Producer.

PropertyList		
[O]	Property	properties[]
[O]	ResetProperty	resetProperties[]
[O]	Extension	extensions[]

Members:

- `properties`: Each member in this array is a `Property` structure carrying information concerning one property.
- `resetProperties`: Each member in this array is a `ResetProperty` structure carrying a property to reset to its default value.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

It is an error for a `property` to be referenced by both the `properties` and `resetProperties` arrays. The Producer MUST return an `InvalidParameters` fault message if the Consumer supplies a `property` in both the `properties` array and the `resetProperties` array of a `PropertyList`.

5.1.20 CookieProtocol Type

This type is a restriction on the string type that is constrained to the values "`none`", "`perUser`" or "`perGroup`". These values carry the following semantics:

- "`none`": The Producer does not need the Consumer to ever invoke **initCookie**.
- "`perUser`": The Consumer MUST invoke **initCookie** once per user of the Consumer, and associate any returned cookies with subsequent invocations on behalf of that user.
- "`perGroup`": The Consumer MUST invoke **initCookie** once per unique `groupID` from the `PortletDescriptions` for the Portlets it is aggregating on a page for each user of the Consumer, and associate any returned cookies with subsequent invocations on behalf of that user targeting Portlets with identical `groupIDs`.

5.1.21 ExtensionPart Type

The `ExtensionPart` structure contains a set of fields for describing one part of a protocol extension.

ExtensionPart		
[R]	QName	name
[R]	QName	type
[O]	anyURI	schemaLocation
[O]	ModelTypes	schemaType

```
 [O] string     extendedTypes[]
 [O] Extension  extensions[]
```

Members:

- `name`: This field contains the full QName for the extension element used to carry this part of the extension.
- `type`: This field contains the QName of the type for the extension element.
- `schemaLocation`: This optional field carries a URI which resolves into a schema containing the `type` for the described `ExtensionPart`. If a URI can not be provided for resolving the type definition, the `schemaType` field in the will need to contain the definition for the `type`.
- `schemaType`: This field can carry a schema defining the type for the `ExtensionPart` for those cases where a URI which resolves to a schema defining the type is not available.
- `extendedTypes`: This array names the WSRP types where this extension is used. The appearance of a type within this list does not mean that the extension will always be sent when that type appears in a message, but is intended as to assist in determining when and how to handle the extension. If no WSRP types are named, then the extension could appear on any WSRP-defined type. As an example; an extension providing additional user contact information might list "Contact" as an `extendedType`.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.22 ExtensionDescription Type

The `ExtensionDescription` structure contains a set of fields for describing a protocol extension.

```
ExtensionDescription
 [R] QName           name
 [R] ExtensionPart   parts[]
 [O] QName           aliases[]
 [O] LocalizedString description
 [O] LocalizedString label
 [O] LocalizedString hint
 [O] Extension       extensions[]
```

Members:

- `name`: This QName references the semantic definition of the extension.
- `parts`: This field contains an array of information regarding the various types involved in this extension and where each might appear within the protocol structures.
- `aliases`: An array of the QNames of extensions known to be semantically equivalent to this extension. While this can provide a hint to the recipient concerning potential items that could be correlated with this extension, it remains the recipient's responsibility do any transformations required to produce the described extension.
- `description`: This localized string provides a description of the extension. To be useful, this needs to include the semantics, or a pointer to the semantics, of the extension.
- `label`: A short, human-readable name for the extension. Intended purpose is for display in any user interface for handling the extension.
- `hint`: A relatively short description of the extension. Intended for display (for example, as a tooltip) in any user interface for handling the extension.

- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.23 ExportDescription Type

The `ExportDescription` structure contains a set of fields for describing export capabilities and restrictions.

```
ExportDescription
 [O] integer   recommendedExportSize
 [O] Extension extensions[]
```

Members:

- `recommendedExportSize:` A optional integer which indicates the normal limit the Producer places on the number of Portlets it is willing to export on a single invocation. When this field is missing, the Producer is providing no guidance to the Consumer concerning limiting the quantity of Portlets requested for export in any one invocation.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.24 ServiceDescription Type

The `ServiceDescription` structure contains a set of fields that describe the offered services of the Producer.

```
ServiceDescription
 [R] boolean             requiresRegistration
 [O] PortletDescription  offeredPortlets[]
 [O] ItemDescription     userCategoryDescriptions[]
 [O] ExtensionDescription extensionDescriptions[]
 [O] ItemDescription     customWindowStateDescriptions[]
 [O] ItemDescription     customModeDescriptions[]
 [O] CookieProtocol      requiresInitCookie
 [O] ModelDescription    registrationPropertyDescription
 [O] string              locales[]
 [O] ResourceList        resourceList
 [O] EventDescription    eventDescriptions[]
 [O] string              supportedOptions[]
 [O] ExportDescription   exportDescription
 [O] boolean             mayReturnRegistrationState
 [O] Extension           extensions[]
```

Members:

- `requiresRegistration:` A boolean indicating whether or not the Producer requires Consumer registration. If `requiresRegistration` is set to "false" then it MUST be valid to not pass a `RegistrationContext`

parameter to all operations with this parameter. If `requiresRegistration` is set to "`true`" then the Producer MUST return a fault message when no `RegistrationContext` is supplied to an operation, other than **getServiceDescription,** which takes this field.

- `offeredPortlets`: An array of structures (defined in **[Section 5.1.16]**) containing the metadata for the Producer Offered Portlets.
- `userCategoryDescriptions`: An array of `ItemDescription` structures as defined in **[Section 5.1.9]**. This array includes entries for every user category the Producer is willing to have the Consumer assert for an End-User, including the user category names defined in **[Section B.1]** of this specification. Note that user categories are Producer-wide and therefore are inherently shared by the Producer's Portlets.
- `extensionDescriptions`: An array of `ExtensionDescription` structures defining extensions the Producer supports. This includes both incoming and outgoing extensions.
- `customWindowStateDescriptions`: An array of `ItemDescription` structures as defined in **[Section 5.1.9]**. This array MUST include an entry for any custom window state the Producer supports.
- `customModeDescriptions`: An array of `ItemDescription` structures as defined in **[Section 5.1.9]**. This array MUST include an entry for any custom mode the Producer supports.
- `requiresInitCookie`: A string (default value = "`none`") indicating whether or not the Producer requires the Consumer to assist with cookie support of the HTTP protocol.
- `registrationPropertyDescription`: Property descriptions for information the Consumer needs to supply during registration.
- `locales`: This array is a superset of locales for which the localized strings of this `serviceDescription` can be requested. The existence of a locale in this array does not imply that all fields of the `ServiceDescription` are available in the locale. Note that this is independent of the locales for which any of the Portlets of this Producer might supply markup.
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `eventDescriptions`: This array of `EventDescription` structures, with unique event names, describes the events the Portlets hosted at the Producer could either generate or handle. These descriptions match events at runtime via the event name, with it being an error for an event's type to not match the type from the `EventDescriptions` with the matching event name. This information is provided to assist the Consumer in pre-determining the flow of events between its constituents. While this list is not required to be exhaustive (i. e. the Portlet may generate an event it has not described), not describing an event greatly reduces its ability to be distributed by some Consumers' event distribution systems.
- `schemaTypes`: This field can carry schemas with types that are referenced by `extensionDescriptions` and `eventDescriptions`. While the descriptions can also reference schemas using a URI, this provides a convenient mechanism to deliver schemas which do not have a publicly accessible URI.
- `supportedOptions`: This field provides a means to indicate which optional features the Producer supports. This specification defines the following list of optional features:
  - wsrp:events
  - wsrp:leasing
  - wsrp:copyPortlets
  - wsrp:import
  - wsrp:export
- `exportDescription`: This optional field carries any guidance/restrictions the Producer is publishing concerning the export of Portlets.
- `mayReturnRegistrationState`: A flag indicating the Producer might return at least a portion of its enduring registration state to the Consumer. The default value of this flag is "`false`".
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.25 Lifetime Type

The `Lifetime` structure provides information regarding when a particular item is scheduled to be expunged. This introduces the ability of Producers and Consumers to cleanup related artifacts in scenarios such as the other party no longer being available.

Lifetime		
[R]	dateTime	currentTime
[R]	dateTime	terminationTime
[O]	duration	refreshDuration
[O]	Extension	extensions[]

Members:

- `currentTime`: This field holds the date and time at which the structure was created. This allows for sensible use of the `terminationTime` field without requiring a protocol to synchronize clocks.
- `terminationTime`: This field holds the date and time when the party sourcing the structure which contains this `Lifetime` information intends to expunge any underlying artifacts.
- `refreshDuration`: The appearance of this optional field tells the receiver that the originator of this information will potentially adjust the `terminationTime` to a value of the supplied value plus the current time on each use of the underlying artifact, should this calculation result in a `terminationTime` which is further in the future. In particular, should adding the `refreshDuration` to the current time result in a dateTime which occurs before the supplied `terminationTime`, the `terminationTime` will not be changed to this computed value.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.26 RegistrationState Type

The `RegistrationState` structure contains fields related to a particular registration of a Consumer with a Producer. It is returned by the **modifyRegistration** operation and contains the fields of a `RegistrationContext` that allow a Producer to return enduring state at registration scope to the Consumer and indicate any change in the scheduled destruction of the registration.

[O]	base64Binary	registrationState
[O]	Lifetime	scheduledDestruction
[O]	Extension	extensions[]

Members:

- `registrationState`: This field is used only when the Producer wants the Consumer to provide enduring storage for the state resulting from processing the registration. If the `RegistrationState` field has a value, the Consumer MUST return this value on any subsequent calls in the context of this registration [R362].
- `scheduledDestruction`: This optional field informs the Consumer that the Producer has changed the scheduled destruction of the underlying registration to the specified date and time. When this field is missing from a response message, scheduled destruction is not in use and the explicit destruction operations MUST be used instead.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.27 RegistrationContext Type

The `RegistrationContext` structure contains fields related to a particular registration of a Consumer with a Producer. It is returned by the **register** operation and is a required parameter on most other operations.

```
RegistrationContext
 [R] Handle       registrationHandle
 [O] base64Binary registrationState
 [O] Lifetime     scheduledDestruction
 [O] Extension    extensions[]
```

Members:

- `registrationHandle`: An unique, invariant and opaque reference to the Consumer-Producer relationship. This reference is generated by either the **register** operation [R355] or a process outside the scope of this specification. Note that Handles are restricted to a maximum length of 255 characters.
- `registrationState`: This field is used only when the Producer wants the Consumer to provide enduring storage for the state resulting from processing the registration. If the `RegistrationState` field has a value, the Consumer MUST return this value on any subsequent calls in the context of this registration [R362].
- `scheduledDestruction`: This optional field informs the Consumer that the Producer has scheduled the underlying registration to be destroyed on a certain date and time unless requested to change that date and time. When this field is missing from a response message, scheduled destruction is not in use and the explicit destruction operations MUST be used instead. Consumers do not need to send this field to the Producer on any operation that takes the `RegistrationContext` structure as this is an output field only and will be ignored when sent to the Producer.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

5.1.28 desiredLocales

This parameter is used to control the locales for which localized strings are returned. The `desiredLocales` parameter is an array of strings, each of which specifies a single locale, whose order indicates the preference of the Consumer as to the locales for which values are returned. Since localized strings use an indirection through resources to carry the set of values for different locales, the first member of this array SHOULD be used as the locale for the values returned directly in the structure. When no `desiredLocales` array is supplied, the Consumer is requesting values for all returned localized strings in all locales where they are available. Since excessive amounts of data can impact both network transmission times and processing time at the Consumer, Producers are encouraged to only send the localized data the Consumer actually requests.

5.2 getServiceDescription Operation

This operation allows a Producer to provide information about its capabilities in a context-sensitive manner (e.g. registration may be required to discover the full capabilities of a Producer) [R303].

```
ServiceDescription = getServiceDescription (registrationContext, desiredLocales,
portletHandles, userContext );
```

Faults: InvalidRegistration, ModifyRegistrationRequired, OperationFailed, ResourceSuspended

Producers may choose to restrict the information returned in `serviceDescription` based on the supplied `RegistrationContext`. The minimum information a Producer MUST return from **getServiceDescription** is that which declares what is required for a Consumer to register (i.e. the `requiresRegistration` flag and whenever additional data is required, the `registrationPropertyDescription` field) with the Producer [R300] [R301] [R303]. Note that the `RegistrationContext` parameter is not likely to be supplied when an unregistered Consumer invokes **getServiceDescription**. This allows the Consumer to gain access to the information required to successfully register. It is recommended that Consumers invoke **getServiceDescription** after registering in order to receive a full description of the capabilities the Producer offers within the context of that registration. Producers MUST return a complete enough `ServiceDescription` to registered Consumers for them to properly interact with both the Producer and Portlets it exposes.

When generating the `ServiceDescription` response the Producer SHOULD use the `desiredLocales` (an array of strings) to control what locales are returned for localized strings.

While it is possible a `ServiceDescription` will change with time (e.g. Producer deploys additional Portlets), Producers are encouraged to return as complete a `ServiceDescription` as possible.

The optional `portletHandles` parameter provides a means for a Consumer to restrict the set of Portlets for which it is requesting **getServiceDescription** return information. When the Consumer does not supply this parameter, the Producer MUST return a `portletDescription` for each of the "Producer Offered Portlets" the Consumer has access to through the supplied `registrationContext` If the Consumer supplies `portletHandles` that do not refer to "Producer Offered Portlets", the Producer is free to ignore those `portletHandles`.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

---

6 Markup Interface

As interactive presentation-oriented web services, all WSRP compliant services implement the markup interface. This interface has operations to request the generation of markup and the processing of interactions with that markup [A300]. This section explains both the signatures for these operations and how the concepts of mode and window state impact the generation of the markup.

6.1 Data Structures

The following additional data structures are needed by this interface:

6.1.1 SessionContext Type

The `SessionContext` structure contains the `ID` and `expires` information the Consumer needs to refer to the session in subsequent invocations.

SessionContext		
[O]	ID	sessionID
[O]	int	expires
[O]	Extension	extensions[]

Members:

- `sessionID`: An opaque string the Portlet defines for referencing state that is stored locally on the Producer. If the Consumer fails to return this reference on future invocations, the Portlet will be unable to reference this state and therefore likely not generate a markup fragment meeting the End-User's expectations. The maximum length of a `sessionID` is 4096 characters, though Producers SHOULD keep it as short as possible as this can have a significant impact on Consumer performance. Producers SHOULD also keep the `sessionID` as stable as possible since changes in value can also have a significant impact on Consumer performance.
- `expires`: Maximum number of seconds between invocations referencing the `sessionID` before the Producer will schedule releasing the related items. A value of -1 indicates that the `sessionID` will never expire. Returning a `sessionID` without an accompanying `expires` carries the semantics of the Producer providing no information to the Consumer regarding the likely lifetime of the referenced session.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

If the Producer returns an `InvalidSession` fault message after returning a `sessionID`, the Consumer MUST NOT resupply that sessionID on a subsequent invocation and SHOULD reinvoke the operation that caused the fault message without any `sessionID` and supply any data that may have been stored in the session.

6.1.2 SessionParams Type

The `SessionParams` structure contains the session information the Consumer is supplying to the Portlet for connection to a Portlet session.

SessionParams		
[O]	ID	sessionID
[O]	Extension	extensions[]

Members:

- `sessionID`: An opaque string the Producer defines for referencing state stored locally on the Producer. If the Producer has returned such a reference and the Consumer fails to return it in this field on future invocations, the Portlet will be unable to reference this state and therefore likely not generate a markup fragment meeting the End-User's expectations.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.3 RuntimeContext Type

The `RuntimeContext` structure defines a collection of fields used only in transient interactions between the Producer and Consumer.

RuntimeContext	
[R] string	userAuthentication
[R] Key	portletInstanceKey
[R] string	namespacePrefix
[O] Templates	templates
[O] SessionParams	sessionParams
[O] Extension	extensions[]

Members:

- `userAuthentication`: String indicating how the End-User was authenticated. Common values include:
  - `wsrp:none`: No authentication was done, user information is asserted for informational purposes only.
  - `wsrp:password`: The End-User identified themselves using the common userid/password scenario.
  - `wsrp:certificate`: The End-User presented a security certificate to validate their identity.
  - Other strings: Some authentication was done outside this limited set of possibilities.
- `portletInstanceKey`: An opaque string, unique within the `RegistrationContext`, which the Consumer MUST supply as a reference to its use of the Portlet. The value which the Consumer supplies in the `portletInstanceKey` field MUST remain constant for any one particular use of the Portlet. The intent of this reference is to allow the Portlet, whenever needed, to use this key to namespace multiple instances of itself within Producer supplied mechanisms. Examples include namespacing within a Producer-defined data sharing mechanism. Since this reference is a `Key`, its length is restricted to 255 characters. Consumers SHOULD keep their `portletInstanceKey` values as short as possible.
- `namespacePrefix`: This field provides a useful string for the Portlet prefixing of tokens that need to be unique within the markup of the aggregated page (e.g. JavaScript variables [A303], HTML id attributes, etc.). In order to support items that could become part of a URL activation, this token MUST remain constant for the lifetime of the `portletInstanceKey` and be the value used for both Consumer and Producer namespacing (see **[Section 10.3]**).
- `templates`: If this Portlet declared `doesUrlTemplateProcessing` as "`true`" in its `PortletDescription`, then this field contains the templates the Consumer is supplying for that processing. If the `PortletDescription` also has `templatesStoredInSession` set to "`true`", then the Consumer MAY elect to only send these once for a `sessionID`.
- `sessionParams`: The Consumer uses this field to supply Portlet session information.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.4 PortletContext Type

The `PortletContext` structure is used as a parameter on many operations to supply the Portlet information that was returned to the Consumer.

PortletContext	
[R] Handle	portletHandle
[O] base64Binary	portletState
[O] Lifetime	scheduledDestruction
[O] Extension	extensions[]

Members:

- `portletHandle`: An opaque and invariant handle, unique within the context of the Consumer's registration (unique within the Producer for Producers not supporting registration). Note that `Handles` are restricted to a maximum length of 255 characters.
- `portletState`: An opaque field the Portlet uses when it depends on the Consumer to store its enduring state [A205] [A502]. If the `portletState` field has a value, the Consumer MUST return this value on subsequent calls using the same `portletHandle` [A512]. Note that such uses can span multiple starting and stopping cycles of the Consumer and therefore this state MUST be persisted by the Consumer until the Portlet's lifecycle ends via the **destroyPortlets** or **deregister** operations or expunging of the Portlet after the expiration of a lease. Producers choosing to return `portletState` to the Consumer still need to persist for themselves any information needed to clean up items related to the portletHandle.
- `scheduledDestruction`: This optional field informs the Consumer that the Producer has scheduled the Portlet to be destroyed on a certain date and time unless requested to change that date and time. When this field is missing, scheduled destruction is not in use and the explicit destruction operations MUST be used instead. Consumers do not need to send this field to the Producer on any operation that takes the `PortletContext` structure as this is an output field only and will be ignored when sent to the Producer.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.5 Standard UserScopes

This specification defines initial values for `UserScopes`. `UserScope` is an open set of values where the Producer SHOULD restrict the values supplied to those specified in this specification or custom values the Consumer has indicated it supports. If another value is specified and the Consumer does not understand it, the Consumer should ignore the cache control and treat the content as non-cacheable. The following values are defined by this specification:

- `wsrp:perUser`: The markup is specific to the userContext for which it was generated. Changes to the data of the `UserContext` MUST invalidate the cached markup.
- `wsrp:forAll`: The markup is not specific to the `UserContext` and therefore may be supplied to all users of the Consumer.

6.1.6 CacheControl Type

The `CacheControl` structure contains a set of fields needed for the Portlet to manage cached markup fragments. Note that any key used by the caching system to locate this markup MUST include the `MarkupParams` structure that was current when the content was originally cached.

```
CacheControl
 [R] int       expires
 [R] string    userScope
 [O] string    validateTag
 [O] Extension extensions[]
```

Members:

- `expires`: Number of seconds the markup fragment referenced by this cache control entry remains valid. A value of -1 indicates that the markup fragment will never expire.

- `userScope`: A string indicating when the markup may be used by various users. If the Consumer does not know how to process the specified `userScope`, it MUST NOT cache the markup.
- `validateTag`: A string the Consumer MAY use to attempt to revalidate markup once the `expires` duration elapses. This potentially eliminates the need for the Portlet to regenerate the markup and thereby can significantly improve the performance for the End-User.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.7 Templates Type

The `Templates` structure contains a set of fields that enable Producer URL writing. The template style format of these fields is defined in **[Section 10.2.2]**.

Templates		
[O]	string	defaultTemplate
[O]	string	blockingActionTemplate
[O]	string	renderTemplate
[O]	string	resourceTemplate
[O]	string	secureDefaultTemplate
[O]	string	secureBlockingActionTemplate
[O]	string	secureRenderTemplate
[O]	string	secureResourceTemplate
[O]	Extension	extensions[]

Members:

- `defaultTemplate`: This template provides the default value for all of the other template fields that do not begin with the string "secure".
- `blockingActionTemplate`: This template provides the template for URLs that will be directed to the Consumer and processed as a **performBlockingInteraction** on the Portlet.
- `renderTemplate`: This template provides the template for URLs that will be directed to the Consumer and processed as a **getMarkup** on the Portlet.
- `resourceTemplate`: This template provides the template for URLs that will be directed to the Consumer and processed either as a **getResource** invocation or a HTTP request on the named resource. When using the HTTP mechanism, the Consumer should use the same verb as was used by the End-User's agent.
- `secureDefaultTemplate`: This template provides the default value for all the secure template fields.
- `secureBlockingActionTemplate`: This template provides the template for secure URLs that will be directed to the Consumer and processed as a **performBlockingInteraction** on the Portlet using a secure protocol.
- `secureRenderTemplate`: This template provides the template for secure URLs that will be directed to the Consumer and processed as a **getMarkup** on the Portlet using a secure protocol.
- `secureResourceTemplate`: This template provides the template for secure URLs that will be directed to the Consumer and processed either as a **getResource** invocation or a HTTP request over SSL/TLS on the named resource. When using the HTTP mechanism, the Consumer should use the same verb as was used by the End-User's agent.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.8 CCPPProfileDiff Type

The `CCPPProfileDiff` structure holds the information defined by the CC/PP standard for declaring differences from the referenced profiles.

CCPPProfileDiff		
[R]	string	diffName
[R]	string	description
[O]	Extension	extensions[]

Members:

- `diffName`: This field provides the header name consisting of the prefix "Profile-Diff-" and a number as defined in the CC/PP exchange.
- `description`: This field carries the XML/RDF describing the difference.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.9 CCPPHeaders Type

The `CCPPHeaders` structure holds the information defined by the CC/PP standard.

CCPPHeaders		
[R]	string	profile
[O]	CCPPProfileDiff	profileDiffs[]
[O]	Extension	extensions[]

Members:

- `profile`: This field carries the list of profiles as defined in the CC/PP standard.
- `profileDiffs`: This array carries the profile-diff headers from the CC/PP exchange.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.10 ClientData Type

The `ClientData` structure contains information the client supplied to Consumer about itself, including user-agent identification and capabilities.

ClientData		
[O]	string	userAgent
[O]	CCPPHeaders	ccppHeaders
[O]	string	requestVerb
[O]	Extension	extensions[]

Members:

- `userAgent`: String identifying the user-agent of the End-User.
- `ccppHeaders`: Contains the WSRP representation of the CCPP Headers which were received or produced by the Consumer.
- `requestVerb`: This field provides the means for the Producer to be notified of the transport level verb (e.g. in HTTP these are: "GET", "POST", "PUT" and "DELETE") used for the End-User interaction. The primary purpose of this field is to allow the Producer to communicate with other systems with the same semantics as the End-User interaction.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.11 NamedString Type

The `NamedString` type provides a standardized way of carrying a simple name/value pair.

```
NamedString
 [R] string name
 [O] string value
```

Members:

- `name`: The name to be associated with this value.
- `value`: The associated value. Uses of this structure will need to define the semantics for when a `value` is not supplied.

6.1.12 NavigationalContext Type

The `NavigationalContext` type provides a means to carry both the opaque and public portions of the Portlet's navigational state. Comments regarding this type of state from **[Section 3.7]**, **[Section 3.8]** and **[Section 6.8]** apply equally to all components of this structure.

```
NavigationalContext
 [O] string      opaqueValue
 [O] NamedString publicValues[]
 [O] Extension   extensions[]
```

Members:

- `opaqueValue`: The portion of the Portlet's navigational state which is opaque to the Consumer.
- `publicValues`: The Portlet can declare a portion of its navigational state to the Consumer for the purpose of coordination with other components the Consumer is aggregating. This declared portion of the Portlet's navigational state is stored and managed by the Consumer in a manner equivalent to the opaque portion and carried to the Portlet in this field on all relevant invocations [A502] [A503] [C404]. The Portlet also returns the current values for the publicly exposed portion of its navigational state using this field. [C405].
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

Both the Producer and Consumer supply values to each other for the `opaqueValue` and `publicValues` fields. The

Consumer needs to return the current value for the `opaqueValue` field, but can apply updates to the `publicValues` field as part of state-based coordination between Portlets and other Consumer constituents. These statements are true for all operations receiving or returning the `NavigationalContext` structure [C401]. While Consumer policy will govern when values from sources other than the Portlet update the `publicValues`, Consumers MUST apply Portlet supplied values (i.e. supplied on a portlet url parameter or on a response from **performBlockingInteraction** or **handleEvents**) to the `publicValues` and SHOULD supply the same value to Portlets which provide a `navigationalParameterDescription` referencing the same QName in the `names` array. An example of when this might not be appropriate is when the Portlets are related to the same Producer offered portlet handle. The context in which the Consumer is using Portlets related to the same Producer offered portlet handle will determine whether or not such Portlets will share the same `publicValues`. For example, a Consumer may choose to share a `publicValue` related to the End User's name between such Portlets and not share a `publicValue` related to a stock symbol.

6.1.13 MimeRequest Type

The `MimeRequest` structure contains information frequently used to control generation of items with varying mime types.

MimeRequest	
[R] boolean	secureClientCommuncation
[R] string	locales[]
[R] string	mimeTypes[]
[R] string	mode
[R] string	windowState
[O] ClientData	clientData
[O] NavigationalContext	navigationalContext
[O] string	markupCharacterSets[]
[O] string	validateTag
[O] string	validNewModes[]
[O] string	validNewWindowStates[]
[O] Extension	extensions[]

Members:

- `secureClientCommuncation`: A flag indicating whether or not the Consumer knows the delivery channel between the client and Consumer to be secure [A609] [R401] [R408]. The Consumer MUST set the `secureClientCommunication` flag as the Portlet MAY render different content when it knows the delivery channel is secure.
- `locales`: An array of locales where the order in the array is the Consumer's order of preference for the Portlet to generate the markup (e.g. "en-US"). Note that current practice on the Internet uses the format [2 char language code][9]"-" [2 char country code][10] as per the provided example. The Consumer can supply this information based on the setting the End-User has requested, but is encouraged to also take into account the locales the PortletDescription declared were supported for the mime types being requested.
- `mimeTypes`: An array of Mime types[11] (e.g. "text/html", "application/xhtml+xml", etc.) where the order in the array is the order in which the Consumer would prefer the Portlet generate the markup (i.e. first is most preferred, second is next preferred, etc.). In addition to these fully specified Mime types, use of "*" (indicates all Mime types are acceptable) and type/* (where type includes things such as "text") from the HTTP definition

[12] MAY be specified. Portlets SHOULD generate markup in one of the specified Mime types. The Producer/ Portlet does not have to process any optional parameters that can be included on mime type declarations.

- `mode`: The mode for which the Portlet should render its output. A set of modes is defined in this specification (see **[Section 6.9]**). In addition, the Portlet's metadata indicates which of these modes the Portlet supports as well as any Producer-defined modes. The Consumer MUST specify either one of the modes from the Portlet's metadata or "`wsrp:view`" (all Portlets are required to support this mode).
- `windowState`: The state of this Portlet's virtual window relative to other Portlets on the aggregated page. Constants and definitions for the specification-defined states are found in **[Section 6.10]**. The Consumer MUST specify either one of the windowStates from the Portlet's metadata or "`wsrp:normal`" (all Portlets are required to support this windowState).
- `clientData`: A structure that provides information about the client device which will render the markup.
- `navigationalContext`: This field contains the navigational state for this Portlet. To exist, the opaque portion of navigational state must be set explicitly on each URL activation or by setting its value upon return from the **performBlockingInteraction** or **handleEvents** operations [C403]. The public portion of navigational state could be set by the Consumer even if not supplied by the Portlet.
- `markupCharacterSets`: An array of `characterSets`[13] (e.g. "UTF-8", "ISO-10646-Unicode-Latin1", etc.) the Consumer is willing to have the Portlet use for encoding the markup (i.e. the character set for the aggregated page). The order of this array indicates the preferred ordering of the Consumer with the first element in the array being the most preferred. When the SOAP binding is in use, the Producer MUST either use one of the markupCharacterSets, UTF-8 or UTF-16 for the response message as the nature of XML requires the character set used for the markup to be the same as the response message.
- `validateTag`: This field MAY contain a `validateTag` previously supplied to the Consumer in a `CacheControl` structure. When this field has a value, the Consumer is indicating it has cached a previous response for this item for the Portlet, but the `CacheControl` structure governing the use of that cached item no longer indicates it is valid. The Consumer is supplying the `validateTag` as a means for the Portlet to avoid generating the item if the cached item can be validated. The Portlet sets the `useCachedItem` field in the returned `MimeResponse` to "`true`" to indicate the item referenced by the `validateTag` is still valid.
- `validNewModes`: Current set of modes the Producer MAY request changing to. These can be used to specify a mode change either via an `UpdateResponse` or within an URL written into the returned markup. It should be noted that this is no guarantee that a requested transition will be honored, as factors not easily represented may cause the Consumer to reject a requested transition. The primary reason for supplying this information is to assist the Portlet in preparing a user interface that does not contain links the Consumer will not honor. If no values are supplied, the Portlet can assume that all transitions are valid. Consumers can indicate they prohibit all transitions by supplying just the current `mode` in this array.
- `validNewWindowStates`: An array of `windowStates` which the Consumer is indicating as available to be requested as a `newWindowState` in `UpdateResponse`. It should be noted that this is no guarantee that a requested transition will be honored, as factors not easily represented may cause the Consumer to reject a requested transition. The primary reason for supplying this information is to assist the Portlet in preparing a user interface that does not contain links the Consumer will not honor. If no values are supplied, the Portlet can assume that all transitions are valid. Consumers can indicate they prohibit all transitions by supplying just the current `windowState` in this array.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

class="v2">Custom `modes`, `windowStates`, `userScopes` and `userAuthentication` values MUST be URI's in order to reduce name clashes with any values that may be defined by future versions of this specification.

6.1.14 MarkupParams Type

The schema definition of the `MarkupParams` structure extends the common `MimeRequest` definition (see **[Sections 6.1.13]**) without any additional fields and provides the data needed for the Portlet to generate markup that will enable the End-User to visualize the state of the Portlet. These are also supplied to the interaction processing operations as they may impact that processing (e.g. validNewModes) and those operations are allowed to return markup and thereby avoid an additional invocation.

6.1.15 ResourceParams Type

The schema definition of the `ResourceParams` structure extends the common `MimeRequest` definition (see **[Sections 6.1.13]**) adding fields specific to invoking the **getResource** operatione.

```
ResourceParams (also see the fields defined in [Sections 6.1.13])
 [R] ID            resourceID
 [O] string        resourceState
 [O] NamedString   formParameters[]
 [O] UploadContext uploadContexts[]
```

Members:

- `resourceID`: This field provides the identifier the Portlet had placed as the value of the `wsrp-resourceID` portlet URL parameter.
- `resourceState`: This field provides the state encoded on the resource URL using the `wsrp-resourceState` portlet URL parameter.
- `formParameters`: Name/value pairs reflected, for example, in the case of HTML either from the query string of a form submitted with method=get or in a request with mime type = "application/x-www-form-urlencoded" submitted with method=post. For the case of query string parameters, Consumers should take care with regard to how user-agents encode this data. In particular, common user-agents (e.g. web browsers) encode posted data in the character set of the page containing the form. As the Producer is ignorant of this encoding and the Consumer is required to consistently encode parameters passed to the Producer in the SOAP message, Consumers MUST ensure that form data is properly decoded before it is passed to the Producer.
- `uploadContexts`: An optional field where mime types not parsed into `formParameters` are placed for transfer to the Producer.

6.1.16 MimeResponse Type

The `MimeResponse` structure contains common fields relative to returning an item described by a mime type.

```
MimeResponse
 [O] boolean      useCachedItem
 [O] string       mimeType
 [O] string       itemString
 [O] base64Binary itemBinary
 [O] string       locale
 [O] boolean      requiresRewriting
 [O] CacheControl cacheControl
```

```
 [O] string       ccppProfileWarning
 [O] Extension    extensions[]
```

Members:

- `useCachedItem`: A boolean used to indicate whether the item the Consumer indicated it has cached is still valid. The default value of this field is "false" (i.e. a new item is being returned for the Consumer's use). If the value for `useCachedItem` is "`true`" the `itemString` and `itemBinary` fields MUST NOT be returned. If the field's value is "`true`", any supplied `cacheControl` field MUST be processed as a replacement for the `cacheControl` originally supplied with the cached item.
- `mimeType`: The mime type of the returned item. The `mimeType` field MUST be specified whenever an item is returned, and if the `itemBinary` field is used to return the item, the mime type MUST include the character set for textual mime types using the syntax specified in RFC1522[14] (e.g. "text/html; charset=UTF-8"). In this particular case this character set MAY be different than the response message.
- `itemString`: This is a string version of the item. If this is encoded in a SOAP message (i.e. XML), various characters will likely need to be escaped using XML entities (e.g. "<" becomes "<"), either by the Portlet or the Producer's runtime. The character set of the markup a Portlet returns MUST either match that requested in `MimeRequest`, be UTF-8 or UTF-16. When a SOAP binding is used, the XML specification requires the character set of the markup match the character set of the response message's document. This field is only missing when the `useCachedItem` flag is "`true`". This field is mutually exclusive with returning the markup in the `itemBinary` field.
- `itemBinary`: The item represented as a binary stream. This is useful if the item is not easily mapped to the string type or an attachment scheme is in use that moves binary types into separate message parts (e.g. DIME). This field is mutually exclusive with returning the item in the `itemString` field.
- `locale`: The locale, if any, for the returned item.
- `requiresRewriting`: A flag by which the Portlet/Producer indicates whether or not Consumer-side rewriting (see **[Sections 10.2.1]** and **[Section 10.3.1]**) is required. The Consumer MUST parse the item for rewriting if the value of `requiresRewriting` is "`true`". The default value for this flag is "`false`".
- `cacheControl`: Defines the caching policies for the returned item. If the `cacheControl` field is not supplied, the Portlet is indicating it does not consider the item cacheable. This is without prejudice to Consumer specific caching policies.
- `ccppProfileWarning`: This field can carry a list of warning values as defined in the CC/PP standard.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.17 ResourceContext Type

The schema definition of the `ResourceContext` structure extends the common `MimeResponse` definition (see **[Sections 6.1.16]**) without any additional fields.

6.1.18 ResourceResponse Type

The `ResourceResponse` structure contains fields for returning various items in response to a **getResource** invocation.

```
ResourceResponse
 [R] ResourceContext resourceContext
```

```
 [O] SessionContext  sessionContext
 [O] Extension       extensions[]
```

Members:

- `resourceContext`: A structure carrying the returned resource and fields related to it.
- `sessionContext`: This structure contains session-oriented fields that may be returned from various operations, including a new `sessionID` and the duration before it `expires`.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.19 MarkupContext Type

The schema definition of the `MarkupContext` structure extends the common `MimeResponse` definition (see **[Sections 6.1.16]**), adding fields relative to returning a Portlet's markup.

```
MarkupContext (also see the fields defined in [Sections 6.1.16])
 [O] string                preferredTitle
```

Members:

- `preferredTitle`: The title the Portlet would prefer to be used in any decoration of the markup. The locale and markup type, for textual markup types only, of the preferred title has to be identical to that of the markup.

6.1.20 MarkupResponse Type

The `MarkupResponse` structure contains fields for returning various items in response to a **getMarkup** invocation.

```
MarkupResponse
 [R] MarkupContext  markupContext
 [O] SessionContext sessionContext
 [O] Extension      extensions[]
```

Members:

- `markupContext`: A structure carrying the returned markup and fields related to the markup.
- `sessionContext`: This structure contains session-oriented fields that may be returned from various operations, including a new `sessionID` and the duration before it `expires`.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.21 EventPayload Type

The `EventPayload` structure provides a wrapper for the data carried by an event.

EventPayload		
[R]	Object	any

Members:

- `any`: The wrapper element for the event's data. This is required to come from a namespace other than the WSRP "types" namespace (urn:oasis:names:tc:wsrp:v2:types) unless it can be carried within the `NamedStringArray` type (defined in the WSDL as an alternative for convenient serialization/deserialization of this common type).

6.1.22 Event Type

The `Event` structure provides the references back to the QName and payload datatype provided by the `EventDescription`.

Event		
[R]	QName	name
[O]	QName	type
[O]	EventPayload	payload
[O]	Extension	extensions[]

Members:

- `name`: The namespaced name of the event.
- `type`: A reference to a schema-defined type for the event's payload. If an `EventDescription` was supplied with a matching event name, it is an error for this type to not match the type supplied in that `EventDescription`.
- `payload`: This optional field contains the data for the event and becomes a REQUIRED field when either the relevant `EventDescription` or the `Event` structures specify a value for the `type` field. One example of when this field might not be included is when the event is simply a signal and the event's name thereby carries all of the semantics of the event. Its contents MUST conform to the schema referenced by the `type` field.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.23 UpdateResponse Type

The `UpdateResponse` structure contains the items normally returned by **performBlockingInteraction**.

UpdateResponse		
[O]	SessionContext	sessionContext
[O]	PortletContext	portletContext
[O]	MarkupContext	markupContext
[O]	Event	events[]
[O]	NavigationalContext	navigationalContext
[O]	string	newWindowState
[O]	string	newMode

Members:

- `sessionContext`: This structure contains session-oriented fields that may be returned from various operations, including a new `sessionID`, the duration before it `expires`.
- `portletContext`: This structure is where a Portlet using Consumer-side enduring storage may return a change in its enduring state, provided the `portletStateChange` flag in `InteractionParams` had been set to "`readWrite`" or "`cloneBeforeWrite`". When the `portletStateChange` flag had been set to "`cloneBeforeWrite`", this may also include a new `portletHandle`. The sequence by which a Portlet can otherwise request changing this state is described in **[Section 6.4.3]**.
- `markupContext`: Markup may be returned at the end of interaction processing as an optimization that avoids an additional remote invocation. To ensure End-Users receive expected behavior from bookmarked pages, it is important that Portlets taking advantage of this optimization use the navigational state that the Consumer would have had for invoking **getMarkup**.
- `events`: An optional array of Event structures specifying the events generated during the processing of the operation returning this structure.
- `navigationalContext`: The navigational state which the Portlet is returning to the Consumer to indicate the navigational state to be supplied on future invocations of the Portlet, including for page refreshes and page bookmarks [C402] [C403]. The semantics of returning a `publicValue` item without a `value` are to clear any stored `value` for that item. This state is for the purpose of generating markup. The Consumer SHOULD supply the returned values, along with any additional or updated `publicValues`, as the navigational state on the subsequent invocations for this use of the Portlet for at least the duration of the End-User's interactions with this aggregated page. The Consumer is not required to persist the `navigationalContext` for longer than this set of interactions, but can provide such a persistence if desired.
- `newWindowState`: A request from the Portlet to change the window state. See [Section 6.10] relative to the processing of such requests.
- `newMode`: A request from the Portlet to change the mode. See [Section 6.9] relative to the processing of such requests.

For optimization purposes this structure allows markup to be returned even while state changes, events and mode/windowState change requests are also returned. The semantics of this are that the markup reflects the current Portlet state and assumes any requested changes to the mode and/or windowState are honored by the Consumer.

6.1.24 BlockingInteractionResponse Type

The `BlockingInteractionResponse` structure contains the various items **performBlockingInteraction** can return.

BlockingInteractionResponse		
[O]	UpdateResponse	updateResponse
[O]	string	redirectURL
[O]	Extension	extensions[]

Members:

- `updateResponse`: This field captures the items returned when the Portlet is not directing the user to a different URL. It is mutually exclusive with the `redirectURL` field.
- `redirectURL`: As a result of processing this interaction, the Portlet may indicate to the Consumer that it would like the End-User to view a different URL. It is mutually exclusive with the `updateResponse` field. Note that for this version of the specification, these URLs are required to be absolute URLs, however, they are

allowed to include values for parameters which utilize the URL rewriting mechanism defined in **[Section 10.2.1]**. Consumers MUST rewrite these URLs in the same manner as those contained within markup the Portlet might return. An example of when this might be useful is passing a parameter containing a url for returning to the current page when redirecting the End-User to a general page which handles a user sign-in process.

- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.25 ErrorCodes

The following is an enumerated set of QNames (the wsrp: prefix refers to the URI "`urn:oasis:names:tc:wsrp:v2:types`") defined as `ErrorCodes` by this specification for use in `HandleEventsFailed, FailedPortlets` and `ImportPortletsFailed` types. These `ErrorCodes` have the following meanings:

- `wsrp:AccessDenied`: Required access to the specified item was denied.
- `wsrp:ExportNoLongerValid`: An **importPortlets** operation referenced an exported item that is no longer available.
- `wsrp:InconsistentParameters`: The supplied parameters do not provide a consistent set of references.
- `wsrp:InvalidRegistration`: The supplied `registrationHandle` is invalid.
- `wsrp:InvalidCookie`: A supplied cookie was invalid for the referenced items. The Consumer can try a recovery process as would be used if the equivalent fault had been returned.
- `wsrp:InvalidHandle`: The supplied `portletHandle` is not valid for the operation.
- `wsrp:InvalidSession`: The supplied session is no longer valid. The Consumer can try a recovery process as would be used if the equivalent fault had been returned.
- `wsrp:InvalidUserCategory`: The supplied `userCategory` is not allowed to perform the requested action on the referenced items.
- `wsrp:ModifyRegistrationRequired`: The supplied `registrationHandle` has been suspended pending appropriate changes from a **modifyRegistration** request.
- `wsrp:MissingParameters`: The request did not supply the full set of parameters required to perform the processing on the referenced items.
- `wsrp:OperationFailed`: An attempt to process the request resulted in a failure. Whether or not a retry would likely succeed is unknown.
- `wsrp:OperationNotSupported`: The requested operation can not be performed on the referenced items.
- `wsrp:ResourceSuspended`: The requested operation can not be performed as the referenced resources have been suspended.
- `wsrp:TooBusy`: The Producer has too many other processing needs to attend to the request at this time.
- `wsrp:TooManyRequested`: The request has taken all of the processing time the Producer is willing to allow.

Additional reasons may be constructed by Producers, though automatic recovery becomes less likely with such additional definitions. As a result, the reason fields are defined as `LocalizedString` so that they are reasonable to display to administrative users.

6.1.26 HandleEventsFailed Type

The `HandleEventsFailed` structure contains the 0-based index of an event in the incoming events array that could not be processed fully by the Producer, and the reason for failure.

`HandleEventsFailed`

[R]	int	index[]
[R]	ErrorCodes	errorCode
[O]	LocalizedString	reason
[O]	Extension	extensions[]

Members:

- `index`: The 0-based index of the events, each of which had their processing fail for the stated `errorCode`.
- `errorCode`: One of the enumerated `ErrorCodes`, describing why the handling of the referenced events failed.
- `reason`: An explanation of the failure encountered which is intended for display to an End-User.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.27 HandleEventsResponse Type

The `HandleEventsResponse` structure contains the various items **handleEvents** can return.

HandleEventsResponse		
[O]	UpdateResponse	updateResponse
[O]	HandleEventsFailed	failedEvents[]
[O]	Extension	extensions[]

Members:

- `updateResponse`: This field captures the items normally returned from the Portlet having processed the set of events. Note that Consumer policy will control how requested changes in `windowState` and `mode` are handled, but that the difficulties in reconciling multiple requests to change `windowState` (impacts on overall page layout could make it unusable) will cause many Consumers to not honor such requests.
- `failedEvents`: This optional array carries notifications to the Consumer of events the Portlet failed to process. Since the Producer/Portlet is capable of appropriate retries for the processing of any given event, the Consumer MUST NOT retry distributing failed events to the Portlet.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.28 StateChange Type

This type is a restriction on the string type that is constrained to the values "`readWrite`", "`cloneBeforeWrite`" or "`readOnly`", the meanings of which are explained in **[Section 6.4.3]**.

6.1.29 UploadContext Type

The `UploadContext` structure contains fields specific to uploading data to the Portlet.

UploadContext		
[R]	string	mimeType

```
[R] base64Binary uploadData
[O] NamedString  mimeAttributes[]
[O] Extension    extensions[]
```

Members:

- `mimeType`: Mime type of what is in the `uploadData` field. The syntax for this value is defined in RFC1522[15] (e.g. "text/html; charset=UTF-8").
- `uploadData`: A binary data blob that is being uploaded.
- `mimeAttributes`: Attributes received from the client (e.g. http headers, other attributes relative to the upload file, etc.) that are not represented elsewhere within the protocol.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.30 InteractionParams Type

The `InteractionParams` structure contains fields specific to invoking the **performBlockingInteraction** operation.

```
InteractionParams
 [R] StateChange   portletStateChange
 [O] string        interactionState
 [O] NamedString   formParameters[]
 [O] UploadContext uploadContexts[]
 [O] Extension     extensions[]
```

Members:

- `portletStateChange`: A flag by which a Consumer indicates whether or not the processing of the interaction is allowed to return a modified `portletState`. This flag is needed as only the Consumer knows whether or not such a state change would be acceptable. In many cases where the Consumer does not authorize the End-User to modify the enduring state of the Portlet in use, it may permit the Producer to clone the Portlet (i. e. set `portletStateChange` to "`cloneBeforeWrite`") and return a clone of the Portlet in addition to any other return parameters. The full use of this flag is described in **[Section 6.4.3]**.
- `interactionState`: Opaque representation of transient information for use in processing this invocation of **performBlockingInteraction**. The value for this field is supplied through the portlet URL parameter `wsrp-interactionState` (see **[Section 10.2.1.4]**).
- `formParameters`: Name/value pairs reflected, for example, in the case of HTML either from the query string of a form submitted with method=get or in a request with mime type = "application/x-www-form-urlencoded" submitted with method=post. For the case of query string parameters, Consumers should take care with regard to how user-agents encode this data. In particular, common user-agents (e.g. web browsers) encode posted data in the character set of the page containing the form. As the Producer is ignorant of this encoding and the Consumer is required to consistently encode parameters passed to the Producer in the SOAP message, Consumers MUST ensure that form data is properly decoded before it is passed to the Producer.
- `uploadContexts`: An optional field where mime types not parsed into `formParameters` are placed for transfer to the Producer.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.31 EventParams Type

The `EventParams` structure contains fields specific to invoking the **handleEvents** operation.

EventParams		
[R]	StateChange	portletStateChange
[R]	Event	events[]
[O]	Extension	extensions[]

Members:

- `portletStateChange`: A flag by which a Consumer indicates whether or not the processing of events is allowed to return a modified `portletState`. This flag is needed as only the Consumer knows whether or not such a state change would be acceptable. In many cases where the Consumer does not authorize the End-User to modify the enduring state of the Portlet in use, it may permit the Producer to clone the Portlet (i.e. set `portletStateChange` to "`cloneBeforeWrite`") and return a clone of the Portlet in addition to any other return parameters. The full use of this flag is described in **[Section 6.4.3]**.
- `events`: An array of events for the Portlet to process.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32 User Profile Types

The `UserProfile` structure is used to carry information about the End-User. The Portlet uses the `userProfileItems` in its metadata to describe the fields it uses to generate markup from this set and any others the Consumer indicated were available when it registered. See **[Section 11]** for a complete description of this portion of the protocol. We expect that most extensions of the types referenced by `UserProfile` will be of the type `QNamedString` (from the "wsrp-extra" namespace) and encourage its use in this manner.

UserProfile		
[O]	PersonName	name
[O]	dateTime	bdate
[O]	string	gender
[O]	EmployerInfo	employerInfo
[O]	Contact	homeInfo
[O]	Contact	businessInfo
[O]	Extension	extensions[]

Members:

- `name`: A structure containing the various fields for the End-User's name.
- `bdate`: The End-User's birthdate. This uses the schema-defined datatype for DateTime rather than Date as not all web stacks serialize / deserialize Date properly.
- `gender`: The End-User's gender ("M" = male, "F" = female).
- `employerInfo`: A structure containing various fields for the End-User employer's information.
- `homeInfo`: The End-User's home location information.

- `businessInfo:` The End-User's work location information.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.1 PersonName Type

The `PersonName` structure carries the detailed fields for the parts of an End-User's name.

PersonName	
[O] string	prefix
[O] string	given
[O] string	family
[O] string	middle
[O] string	suffix
[O] string	nickname
[O] Extension	extensions[]

Members:

- `prefix:` Examples include Mr, Mrs, Ms, Dr, etc.
- `given:` The End-User's first or given name.
- `family:` The End-User's last or family name.
- `middle:` The End-User's middle name(s) or initial(s).
- `suffix:` Examples include Sr, Jr, III, etc.
- `nickname:` The End-User's preferred nick name.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.2 EmployerInfo Type

The `EmployerInfo` structure contains the detailed fields concerning the End-User's employer.

Employerinfo	
[O] string	employer
[O] string	department
[O] string	jobtitle
[O] Extension	extensions[]

Members:

- `employer:` The name of the employer.
- `department:` The name of the department the End-User works within.
- `jobtitle:` The title of the End-User's job.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.3 TelephoneNum Type

The `TelephoneNum` structure is used to describe the subfields of a phone number.

TelephoneNum		
[O]	string	intcode
[O]	string	loccode
[O]	string	number
[O]	string	ext
[O]	string	comment
[O]	Extension	extensions[]

Members:

- `intcode`: The international telephone code.
- `loccode`: Local telephone area code.
- `number`: The telephone number.
- `ext`: Any telephone number extension.
- `comment`: Comment about this phone number.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.4 Telecom Type

The `Telecom` structure is used to describe the various phone contact information.

Telecom		
[O]	TelephoneNum	telephone
[O]	TelephoneNum	fax
[O]	TelephoneNum	mobile
[O]	TelephoneNum	pager
[O]	Extension	extensions[]

Members:

- `telephone`: Standard phone number.
- `fax`: Phone number for faxes.
- `mobile`: Phone number for mobile contact.
- `pager`: Phone number for activating a pager.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.5 Online Type

The `Online` structure is used to describe various types of web-oriented contact information.

```
Online
 [O] string    email
 [O] string    uri
 [O] Extension extensions[]
```

Members:

- `email`: Email address.
- `uri`: Relevant web page.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.6 Postal Type

The `Postal` structure carries the detailed fields describing a particular address.

```
Postal
 [O] string    name
 [O] string    street
 [O] string    city
 [O] string    stateprov
 [O] string    postalcode
 [O] string    country
 [O] string    organization
 [O] Extension extensions[]
```

Members:

- `name`: The name to which items should be addressed.
- `street`: The street portion of the address.
- `city`: The city portion of the address.
- `stateprov`: The state or province portion of the address.
- `postalcode`: The postal code portion of the address.
- `country`: The country portion of the address.
- `organization`: Any organization needing to be specified in the address.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.32.7 Contact Type

The `Contact` structure is used to describe a location for the End-User.

```
Contact
 [O] Postal    postal
 [O] Telecom   telecom
 [O] Online    online
```

[O] Extension	extensions[]

Members:

- `postal`: Postal oriented contact information.
- `telecom`: Telephone oriented contact information.
- `online`: Web oriented contact information.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.1.33 UserContext Type

The `UserContext` structure supplies End-User specific data to operations. Note that this does not carry user credentials (e.g. userID / password) as quite flexible mechanisms for communicating this information are being defined elsewhere (e.g. WS-Security defines how to carry User Information in a SOAP header). There are several use cases for the application-level information carried in this structure, particularly in the absence of authenticated security-level information, namely; logging of remote invocations, interoperable assertions of unauthenticated information, personalization of the user experience, etc.

UserContext	
[R] Key	userContextKey
[O] string	userCategories[]
[O] UserProfile	profile
[O] Extension	extensions[]

Members:

- `userContextKey`: This key is a token that the Consumer supplies to uniquely identify the UserContext. The `userContextKey` MUST remain invariant for the duration of a Consumer's registration. The Producer can use this key as a reference to the user. One anticipated such usage relates to Producer logging regarding remote invocations.
- `userCategories`: An array of strings, each of which specifies an Producer-defined user category in which the Consumer places the End-User relative to the current operation [R418]. The Consumer MUST NOT assert a user category for which no `ItemDescription` was part of the Producer's `ServiceDescription`. See the discussion of user categories in **[Section 6.12]**. One anticipated usage of this field is application-level decisions in the absence of security-level guidance.
- `profile`: End-User profile data structure as defined in **[Section 6.1.28]** [R409]. Note that while the `UserContext` structure is passed to many operations, only the interaction oriented operations need this optional field to be supplied.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

6.2 getMarkup Operation

The Consumer requests the markup for rendering the current state of a Portlet by invoking:

```
MarkupResponse = getMarkup (registrationContext, portletContext, runtimeContext, userContext,
markupParams);

Faults: AccessDenied, InconsistentParameters, InvalidCookie, InvalidHandle,
InvalidRegistration, InvalidSession, InvalidUserCategory, MissingParameters,
ModifyRegistrationRequired, OperationFailed, ResourceSuspended, UnsupportedLocale,
UnsupportedMimeType, UnsupportedMode, UnsupportedWindowState
```

This operation's semantics are that the Consumer is aggregating a page which includes the Portlet's markup.

6.2.1 Caching of markup fragments

For performance reasons the Consumer might prefer to cache markup across a series of requests. The Producer passes information about the cacheability of the markup fragment in the `cacheControl` structure returned in a `MarkupContext` structure. The Consumer can infer from this information when it may cache markup and when the cached markup needs to be invalidated and updated by a new call to **getMarkup**.

6.2.1.1 Cacheability

Whenever the `cacheControl` field of a `MarkupResponse` structure is filled in the Consumer MAY cache the markup fragment. The Consumer MUST follow the defined invalidation policies from **[Section 6.2.1.2]** in order to keep the cache up-to-date. If the `cacheControl` field is empty, the Portlet has provided no guidance and the Consumer MAY apply whatever cache policy it chooses. For a Portlet to indicate the markup is not cacheable, it will need to return a `cacheControl` structure with a value of zero in the `expires` field.

6.2.1.2 Cache Invalidation

The `expires` field of the `cacheControl` structure provides a time duration during which it is valid to supply the markup from a cache. Once this time has elapsed, counting from the point in time when the `MarkupContext` structure was returned, the Consumer can use the `validateTag` field of the `MarkupParams` structure to inquire whether the markup is still valid, as this potentially avoids having the Portlet regenerate the same markup. Portlets indicating the cached markup can be used SHOULD also supply a new `CacheControl` structure with a new expiry for the markup.

Consumers should be aware that invoking **performBlockingInteraction** and/or **handleEvents** may cause cached markup to become invalid. This version of the specification does not address how a Portlet can indicate that cached markup is invalid, but it is anticipated that future versions will address this issue.

6.3 getResource Operation

The Consumer requests a resource by invoking:

```
ResourceResponse = getResource (registrationContext, portletContext, runtimeContext,
userContext, resourceParams );
```

Faults:	AccessDenied, InconsistentParameters, InvalidCookie, InvalidHandle, InvalidRegistration, InvalidSession, InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed, ResourceSuspended, UnsupportedLocale, UnsupportedMimeType, UnsupportedMode, UnsupportedWindowState

This operation's semantics are that the client/client-agent has requested additional information in a manner that utilized the Consumer as a proxy for supplying that information. As the Consumer is only being used as a proxy for accessing the resource, a number of techniques for storing the Portlet's `navigationalContext"` are not available to it. As a result, while the Portlet's `navigationalContext"` is supplied to this operation, neither the URL nor the response are permitted to change this `navigationalContext"`. If a logical side effect of the invocation is changing the Portlet's `navigationalContext"`, either the Portlet or the Producer will need to manage this change until the next opportunity to return the `navigationalContext"` to the Consumer.

This operation can be used to fetch a resource whenever the activated URL has specified a value for the `wsrp-resourceID` portlet URL parameter and is the preferred mechanism whenever a value of `true` is specified for the `wsrp-preferOperation` (see **[Section 10.2.1.1.3.3]**. It provides all the contextual information needed for a Portlet to re-establish its state relevant to the End-User so that the generated response is able to take that state into account.

Resources which are inserted into the Consmer's aggregated page MUST follow the guidelines in **[Section 10.4]** for the mime types described there and the relevant fragment rules, if any, for other mime types.

6.3.1 Caching of resources

For performance reasons the Consumer might prefer to cache resources across a series of requests. The Producer passes information about the cacheability of the resource representation in the `cacheControl` structure returned in a `MarkupContext` structure. The Consumer can infer from this information when it may cache the resource and when the cached resource needs to be invalidated and updated by a new call to **getResource**. All of the cache semantics described in **[Section 6.2.1]** applies to the caching of resources as well.

6.4 Interaction Operations

End-User interactions with the generated markup may result in invocations for the Portlet to respond to the interactions [A400]. In the case where the invocations may change some data the Portlet is storing in a shared data area (including a database), an operation is needed to carry the semantics of this type of update. The Consumer MUST always propagate these End-User interactions to the Producer.

6.4.1 performBlockingInteraction Operation

This operation requires that both the Consumer beginning the generation of the aggregated page (because the invocation can return a `redirectURL`), invoking other operations on Portlets and the gathering of markup from other Portlets on the page (often because shared state, including state shared via a database, impacts the markup of other Portlets) are blocked until **performBlockingInteraction** either returns or communication errors occur. The Portlet will receive only one invocation of **performBlockingInteraction** per client interaction, excepting for retries.

```
BlockingInteractionResponse = performBlockingInteraction (registrationContext, portletContext,
runtimeContext, userContext, markupParams, interactionParams);
```

```
        AccessDenied, InconsistentParameters, InvalidCookie, InvalidHandle,
        InvalidRegistration, InvalidSession, InvalidUserCategory, MissingParameters,
Faults: ModifyRegistrationRequired, OperationFailed, PortletStateChangeRequired,
        ResourceSuspended, UnsupportedLocale, UnsupportedMimeType, UnsupportedMode,
        UnsupportedWindowState
```

The Consumer has to wait for the response from **performBlockingInteraction** before invoking **getMarkup** on the Portlets it is aggregating. This permits any Producer-mediated sharing to proceed safely (provided it happens in a synchronous manner). Since this operation potentially returns state to the Consumer for storage, this operation also allows Consumers who wish to store this by propagating it to their client to do so before opening the stream for the aggregated page. Consumers doing this will enable End-User bookmarking of the aggregated page for later use. In order to support such bookmarking and reduce issues related to potentially reinvoking a transaction for the End-User, Consumers are encouraged to redirect the client in a manner that keeps a bookmarked page from reissuing a request to invoke **performBlockingInteraction**. Producer still need to be prepared for such repeated invocations as the End-User may activate the link that caused the invocation more than once.

Note, if the Producer chooses to use the optimized form of this operation and return markup directly, care must be taken to ensure the markup is generated with the `navigationalContext` that will be returned from the operation and not the `navigationalContext` that was passed to it. Also, the markup should be generated presuming that any requested `mode` or `windowState` changes are honored. This ensures consistency when the optimization is not used and the Consumer invokes **getMarkup** after **performBlockingInteraction** returns.

6.4.2 handleEvents Operation

A useful way of describing the distinction between an interaction and an event is that an interaction is an encodable event (i.e. can be referenced by presentation markup) with an opaque payload which the Consumer will always attempt to deliver to the Portlet that generated the markup. This differences result in the need for a different signature that the Consumer uses to distribute events to a Portlet; namely:

```
HandleEventsResponse = handleEvents (registrationContext, portletContext, runtimeContext,
userContext, markupParams, eventParams);


        AccessDenied, InconsistentParameters, InvalidCookie, InvalidHandle,
        InvalidRegistration, InvalidSession, InvalidUserCategory, MissingParameters,
Faults: ModifyRegistrationRequired, OperationFailed, OperationNotSupported,
        PortletStateChangeRequired, ResourceSuspended, UnsupportedLocale, UnsupportedMimeType,
        UnsupportedMode, UnsupportedWindowState
```

The events (carried within `eventParams`) provide a means by which a Portlet can be notified about changes in the Consumer application incorporating the Portlet's markup. These notifications could have originated from another Portlet or directly from the Consumer. It is the Consumer which determines which events to distribute to which Portlets, usually based on the needs of the Consumer application and the metadata concerning the events Portlets can publish and/or handle. Since events are independent notifications and the Consumer manages event distribution, Portlets should not expect a particular event distribution behavior (either on event ordering or whether a particular event is distributed to any of the Portlets that can process it).

6.4.2.1 Event handling

In order for the inherent overhead of remote invocations to not degrade End-User performance to unacceptable levels, WSRP collapses invocations that are equivalent outside of the event being processed into a single invocation that deals with an array of events.

Since events are independent notifications, both the Consumer and Producer/Portlet are encouraged to deal with events in a simple time-ordered manner. The protocol defines no semantic meaning to the order of the events nor to the invocation of **handleEvents** that carried a particular set of events to the Portlet.

As many technologies used to implement Portlets do not protect against concurrent updates to runtime state, Producers hosting such Portlets will need to consider each, potentially concurrent, **handleEvents** invocation as transferring a portion of the overall queue of events for the Portlet to process. The Producer could also need to manage the serialization of the event processing with connections back to the response messages at appropriate times.

6.4.2.2 State handling

The processing of an event can potentially impact any portion of a Portlet's state (namely; `portletState`, session state and navigational state). Relevant to the enduring `portletState`, we would particularly note that cloning may occur and that the handling of the `portletStateChange` flag, which provides the Consumer control over cloning, is detailed in **[Section 6.4.3]**.

6.4.2.3 Mode and WindowState handling

Events can return requests to the Consumer to change the mode or windowState of the Portlet. The Consumer SHOULD honor a **handleEvents** invocation returning requests to change `mode` or `windowState` provided they do not conflict with other such requests. Since the policy for handling conflicting requests is up to the Consumer implementation, Portlet developers SHOULD be aware that the impact on the overall layout MAY cause the Consumer to not honor a request to change `windowState`.

6.4.3 Updating Enduring Portlet State

In designing how a Portlet and Consumer interact in order to update the enduring state of the Portlet, the following items were considered:

- Only the Portlet knows when such a state change is desired. While it is expected that changes to enduring state will be relatively rare, they could occur on any interaction the Portlet has with an End-User.
- Only the Consumer knows whether or not a enduring state change would be safe. Reasons for this include whether the enduring state is shared among a group of users, the authorization level of the End-User to impact any shared enduring state and Consumer policies regarding whether the enduring state is modifiable.

This combination requires that all enduring Portlet state changes happen in a manner that has Consumer approval for the change to occur, while the Portlet decides both when the change is required and its exact character. The Consumer indicates whether or not it is safe for the Portlet to modify its enduring state by setting the `portletStateChange` field in the `InteractionParams` or `EventParams` structures. If the Consumer has set the `portletStateChange` flag to "`readWrite`", the Portlet MAY modify its enduring state regardless of whether it is persisted on the Producer or Consumer. Only if the Portlet's enduring state is modified and it is persisted on the Consumer should a `PortletContext` be returned to the Consumer and such a `PortletContext` MUST reference the portletHandle which was supplied by the Consumer.

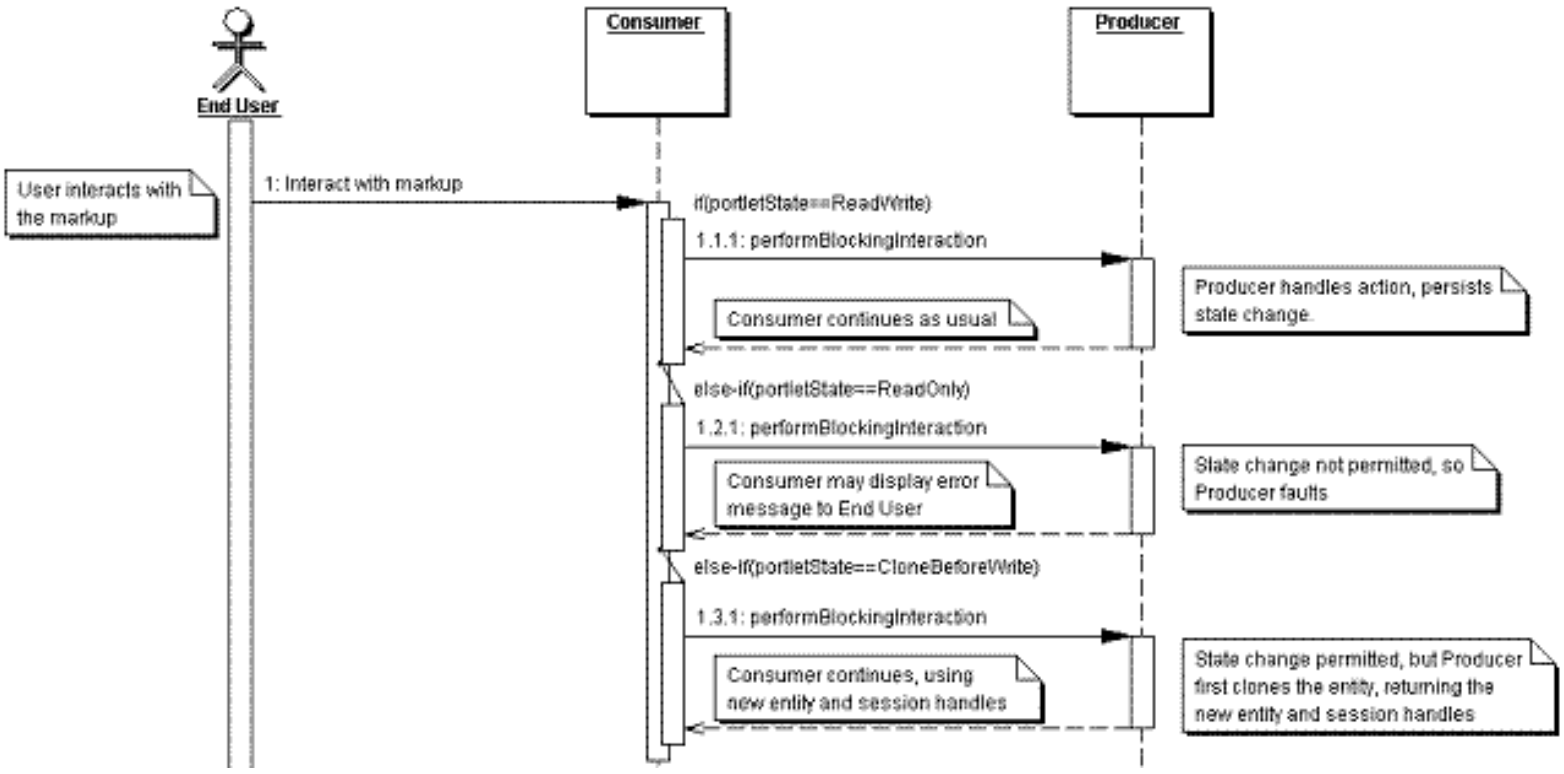
If the Consumer has set the `portletStateChange` field to "`cloneBeforeWrite`", enduring state changes are allowed only if the Producer first clones the Portlet. If the Producer does not clone the Portlet, processing attempts to modify

enduring state MUST proceed as if the Consumer had specified "`readOnly`" for `portletStateChange`. If the Producer clones the Portlet, processing attempts to modify enduring state on the new Portlet SHOULD proceed as if the Consumer had specified "`readWrite`" for `portletStateChange`. The Producer returns the impact of any cloning to the Consumer, regardless of whether the `portletState` is persisted on the Producer or Consumer. If the Producer returns a new `portletHandle` without returning a new `sessionID`, the Consumer MUST associate the current `sessionID` with the new `portletHandle` rather than the previous `portletHandle`. The metadata associated with the original Portlet applies to any cloned Portlet as well.

If the Consumer has set the `portletStateChange` flag to "`readOnly`", the Portlet MUST NOT modify its enduring state regardless of whether it is persisted on the Producer or Consumer and MUST throw a fault message if processing the interaction requires changing its enduring state. Commonly Consumer's will only set the `portletStateChange` flag to "`readOnly`" for End-Users that are not authorized to clone or customize the Portlet (e. g. an End-User using a guest account).

If the Producer implements access control that prevents Portlets from updating enduring state and a Portlet is unable to process the interaction without such an update, then the fault "`PortletStateChangeRequired`" MAY be thrown indicating the interaction processing failed.

This set of possibilities is depicted in the following figure:



6.5 initCookie Operation

In general, the Producer completely manages its own environment, including items such as the initialization of cookies when using the HTTP transport. There are cases, however, when assistance from the Consumer in initializing these cookies is useful. Cookies needed to manage distribution of requests within a load balanced environment are an example of such. This operation is how the Consumer provides such assistance:

```
ReturnAny = initCookie (registrationContext userContext);

Faults: AccessDenied, InvalidRegistration, ModifyRegistrationRequired, OperationFailed,
OperationNotSupported, ResourceSuspended
```

If the Producer's metadata has set the `requiresInitCookie` field to any value other than "`none`", then the Consumer MUST invoke **initCookie** and supply any returned cookies according to the semantics of the value of `requiresInitCookie` as defined in **[Section 5.1.20]**. If at any time the Producer throws a fault message ("`InvalidCookie`") indicating the supplied cookies have been invalidated at the Producer, then the Consumer MUST again invoke **initCookie** and SHOULD reprocess the invocation that caused the fault message to be thrown and include any data that may have been stored in a session related to a cookie.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

6.6 releaseSessions Operation

The Consumer MAY inform the Producer that it will no longer be using a set of sessions by invoking **releaseSessions** (e.g. the Consumer is releasing items related to the `sessionIDs`):

```
ReturnAny = releaseSessions (registrationContext, sessionIDs[], userContext);

Faults: AccessDenied, InvalidRegistration, MissingParameters, ModifyRegistrationRequired,
OperationFailed, OperationNotSupported, ResourceSuspended
```

After invoking **releaseSessions** the Consumer MUST NOT include any of the supplied `sessionIDs` on subsequent invocations.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

6.7 Consumer Transitions across Bindings

Consumers need to be careful about the support supplied by the web stack with regards to multiple bindings that will be offered by many Producers. If a Producer indicates that it uses cookies, the Consumer MUST ensure that any cookies the Producer sets are available on all invocations within the `Markup` interface. Another implication of the Producer indicating it uses cookies is that the Consumer should be aware of the issues involved in protocol transitions (e.g. from HTTP to HTTPS). Current technologies do not always manage cookies in a manner that allows cookies to be shared across such a transition. In addition, moving cookies from an HTTPS to an HTTP connection opens security issues that MUST be handled in the manner prescribed in RFC2109[16]. Consumers MUST respect the security setting on each cookie.

6.8 Stateful Portlet Scenarios

There are several common scenarios for Portlets with varying needs regarding statefulness [A202] [A203]. This section explains how they map into the operational signatures above.

6.8.1 No State

This type of Portlet maintains no state, but encodes everything required to generate the markup on the URL causing the invocation of **getMarkup** [A201]. Often these Portlets involve only a single page, but could provide links on that page that cause the generation of a completely different markup due to the parameters passed when the link is activated.

Note: Invocations of **performBlockingInteraction** can happen in this scenario if the Portlet impacts some backend system as a result of the invocation as this impact could change the markup some other Portlet will generate.

The following table outlines the values for certain key parameters that support this scenario.

Method	Parameter/Field	Value	Comments
performBlockingInteraction	RuntimeContext / sessionParams	Producer provided session state	No value as this Portlet uses no session.
	InteractionParams / interactionState	Consumer extracts value from link.	Interaction state encoded on the URLs in the markup only.
	MarkupParams / navigationalContext	Consumer extracts values from link.	Navigational state encoded on the URLs in the markup only.
	InteractionResponse / navigationalContext	This type of Portlet does not return navigational state.	
handleEvents	RuntimeContext / sessionParams	Producer provided session state	No value as this Portlet uses no session state.
	MarkupParams / navigationalContext	Consumer supplies current value for the Portlet.	Navigational state encoded on the URLs in the markup only.
	HandleEvents Response / navigationalContext	This type of Portlet does not return navigational state.	
getMarkup	RuntimeContext / sessionParams	Producer provided session state	No value as this Portlet uses no session state.
	MarkupParams / navigationalContext	Consumer extracts value from link.	Navigational state from the URL.

6.8.2 Navigational State Only

This type of Portlet does not maintain state at the Producer, but does return navigational state to the Consumer. Both to support these Portlets and to assist Consumers in properly supporting End-User page refreshes and bookmarks, Portlets are allowed to return their navigational state (i.e. the `navigationalContext` field) back to the Consumer. It is then the responsibility of the Consumer to retransmit the navigational state to the Producer with each request [A206].

A stateless Consumer can store the navigational state for all of its aggregated Portlets by returning them to the client, for example by encoding them in the URL. Since this implementation option requires the URL to be generated before the output stream is opened, the navigational state of all Portlets must be known before the Consumer begins generating the output stream. In order to allow the Consumer to open the output stream before it has collected markup from all Portlets aggregated on the page, a **getMarkup** invocation is not allowed to modify the navigational state. Only invocations of **performBlockingInteraction** and **handleEvents** are allowed to modify the navigational state of a Portlet.

The following table outlines the values for certain key parameters that support this scenario.

Method	Parameter/Field	Value	Comments
performBlockingInteraction	RuntimeContext / sessionParams	Producer provided session state	No value as this Portlet uses no session state.
	InteractionParams / interactionState	Consumer extracts value from link.	Interaction state encoded on the URLs in the markup only.
	MarkupParams / navigationalContext	Consumer extracts values from link or previous value.	
	InteractionResponse / navigationalContext	Portlet may compute a changed navigational state.	
handleEvents	RuntimeContext / sessionParams	Producer provided session state	No value as this Portlet uses no session state.
	MarkupParams / navigationalContext	Consumer extracts values from link or previous value.	
	HandleEventsResponse / navigationalContext	Portlet may compute a changed navigational state.	

getMarkup	RuntimeContext / sessionParams	Producer provided session state	No value as this Portlet uses no session state.
	MarkupParams / navigationalContext	From link or from **performBlockingInteraction/ handleEvents** operations.	

6.8.3 Local state

Portlets storing state locally on the Producer establish a session and return an opaque reference (i.e. a `sessionID`) which the Consumer then returns on all subsequent invocations of this Portlet instance on the aggregated page for this End-User in order to reconnect the Portlet to the state stored in the session. These Portlets can also return navigational state to the Consumer such that an End-User may bookmark some portion of the state for use in later conversations. The means by which the Consumer enables this functionality for the End-User is a Consumer implementation choice [A304].

The following table outlines the values for certain key parameters that support this scenario.

Method	Parameter/Field	Value	Comments
performBlockingInteraction	RuntimeContext / sessionParams	Producer provided session state	With session state, the session handle offers ability to store information without increasing the message size to the Consumer.
	InteractionParams / interactionState	Consumer extracts value from link.	Interaction state encoded on the URLs in the markup only.
	MarkupParams / navigationalContext	Consumer extracts values from link or previous value.	
	InteractionResponse / navigationalContext	Portlet may compute a changed navigational state.	

handleEvents	RuntimeContext / sessionParams	Producer provided session state	With session state, the session handle offers ability to store information without increasing the message size to the Consumer.
	MarkupParams / navigationalContext	Consumer extracts values from link or previous value.	
	HandleEventsResponse / navigationalContext	Portlet may compute a changed navigational state.	
getMarkup	RuntimeContext / sessionParams	Producer provided session state	With session state, the session handle offers ability to store information without increasing the message size to the Consumer.
	MarkupParams / navigationalContext	From link or from **performBlockingInteraction/ handleEvents** operations.	

6.9 Modes

A Portlet should render different content and perform different activities depending on its current state, the operation (with parameters) currently being processed, and the functionality requested by the End-User. A base set of functions is defined which reflects those common for portal-portlet interactions. They are referred to as modes and should be thought of as how the Consumer is managing the interaction with the End-User. Portlets may request mode changes either through parameters on a link that an End-User activates or by returning a `newMode` in a `BlockingInteractionResponse` or a `HandleEventsResponse`. The Consumer MUST respect requests to change the mode outside of exceptional circumstances (e.g. access control restrictions), but the Portlet must not depend on such a request being respected.

During **getMarkup, getResource, handleEvents** and **performBlockingInteraction** invocations the Consumer indicates to the Portlet its current mode via the `MarkupParams` data structure.

Because modes are an extensible set of values, the following semantics apply relative to determining what modes are valid for the interactions of a Consumer with a Portlet:

- Portlets specify what modes are supported through their `PortletDescription`. The Producer determines whether or not this information is available to the Consumer prior to registration.
- During registration the Consumer informs the Producer about modes it uses on aggregated pages.
- After registration, the `PortletDescription` can dynamically modify the set of modes supported to incorporate those specified by the Consumer during registration.
- The Consumer is required to use one of the modes specified by the `PortletDescription` (or the required "`wsrp:view`" mode).

6.9.1 "wsrp:view" Mode

The expected functionality for a Portlet in `wsrp:view` mode is to render markup reflecting the current state of the Portlet. The `wsrp:view` mode of a Portlet will include one or more screens that the End-User can navigate and interact with or it may consist of static content devoid of user interactions.

The behavior and the generated content of a Portlet in the `wsrp:view` mode may depend on configuration, personalization and all forms of state.

Conformant Portlets MUST support the `wsrp:view` mode.

6.9.2 "wsrp:edit" Mode

Within the `wsrp:edit` mode, a Portlet should provide content and logic that let a user customize the behavior of the Portlet, though such customizations are not limited to markup generated while in this mode. The `wsrp:edit` mode can include one or more screens which users can navigate to enter their customization data.

Typically, Portlets in `wsrp:edit` mode will set or update Portlet enduring state. How such changes impact Consumer management of Portlet usage by End-Users is discussed in **[Section 6.4.3]**.

6.9.3 "wsrp:help" Mode

When in `wsrp:help` mode, a Portlet may provide help screens that explains the Portlet and its expected usage. Some Portlets will provide context-sensitive help based on the markup the End-User was viewing when entering this mode.

6.9.4 "wsrp:preview" Mode

In `wsrp:preview` mode, a Portlet should provide a rendering of its standard `wsrp:view` mode content, as a visual sample of how this Portlet will appear on the End-User's page with the current configuration. This could be useful for a Consumer that offers an advanced layout capability.

6.9.5 Custom Modes

The extensible `RegistrationData` structure provides a field for Consumers to declare additional custom modes. In

addition, the extensible `PortletDescription` structure provides a field for Portlets to declare what modes they understand. The Portlet could receive a mode it does not currently support as it may have existed in a previous `PortletDescription`. A Portlet MUST map any mode it does not understand to the `wsrp:view` mode. Custom mode values are required to be URI's in order to reduce name clashes with any values that may be defined by other parties, including future versions of this specification.

6.10 Window States

Window state is an indicator of the amount of page space that will be assigned to the content generated by a Portlet. This hint is provided by the Consumer for the Portlet to use when deciding how much information to render in the generated markup. Portlets may request window state changes either through parameters on a link that an End-User activates or by returning a `newWindowState` from **handleEvents** or **performBlockingInteraction**. The Consumer SHOULD choose to respect this request to change the window state, but since the Portlet cannot depend on that choice it MUST NOT encode this new window state into any of its stateful settings. Rather, the Portlet MUST compute any such impact on stateful settings after the Consumer has actually changed the window state.

Because window states are an extensible set of values, the following semantics apply relative to determining what window states are valid for the interactions of a Consumer with a Portlet:

- Portlets specify what window states are supported through their `PortletDescription.` The Producer determines whether or not this information is available to the Consumer prior to registration.
- During registration the Consumer informs the Producer about window states it uses on aggregated pages.
- After registration, the `PortletDescription` can dynamically modify the set of window states supported to incorporate those specified by the Consumer during registration.
- The Consumer is required to use one of the window states specified by the `PortletDescription` (or the required "`wsrp:normal`" window state).

6.10.1 "wsrp:normal" Window State

The `wsrp:normal` window state indicates the Portlet is likely sharing the aggregated page with other Portlets. The `wsrp:normal` window state 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.

Conformant Portlets MUST support the `wsrp:normal` window state.

6.10.2 "wsrp:minimized" Window State

When the window state is `wsrp:minimized`, the Portlet SHOULD NOT render visible markup, but is free to include non-visible data such as JavaScript [A303] or hidden forms. The **getMarkup** operation can be invoked for the `wsrp:minimized` state just as for all other window states.

6.10.3 "wsrp:maximized" Window State

The `wsrp:maximized` window state is an indication the Portlet is likely the only Portlet being rendered in the aggregated page, or that the Portlet has more space compared to other Portlets in the aggregated page. A Portlet SHOULD generate richer content when its window state is `wsrp:maximized`.

6.10.4 "wsrp:solo" Window State

The `wsrp:solo` window state is an indication the Portlet is the only Portlet being rendered in the aggregated page. A Portlet SHOULD generate richer content when its window state is `wsrp:solo`.

6.10.5 Custom Window States

The extensible `RegistrationData` structure provides a field for Consumers to declare additional custom window states. In addition, the extensible `PortletDescription` structure contains a field for Portlets to declare what window states they understand. The Portlet could receive a window state it does not currently support as it may have existed in a previous `PortletDescription`. A Portlet MUST map any window state it does not understand to `wsrp:normal`. Custom `windowState` values are required to be URI's in order to reduce name clashes with any values that may be defined by other parties, including future versions of this specification.

6.11 Defined Events

This specification defines the following events in the interest of promoting interoperability on a set of commonly encountered scenarios. All of these are defined within the WSRP types namespace (i.e. `urn:oasis:names:tc:wsrp:v2:types`).

6.11.1 wsrp:eventHandlingFailed

This is a Consumer generated event which signals to the Portlet that the Consumer detected that errors occurred while distributing events. As a simple notification, this event carries no predefined payload, but does use an open content definition.

6.11.2 wsrp:newNavigationalContextScope

This is a Consumer generated event which notifies the Producer/Portlet that Consumer policy has determined that the scope of the `navigationalContext` has changed. While the value of the `navigationalContext` can change within any one such scope. the changing of scope will reset both the value and managing of the `navigationalContext`. The purpose for this notification is enabling those Producers/Portlets which choose an implementation style precluding the Consumer from fully managing the Portlet's `navigationalContext` to manage an extension to this state in a manner consistent with the Consumer's management for other Producers/Portlets (i.e. write an initial key into the Portlet's `navigationalContext` for later use in determining the correct extended state). While Consumer policy governs when this notification is sent, that policy MUST include providing those Portlets whose `PortletDescription` indicate they handle the `wsrp:newNavigationalContextScope` event an opportunity to write such a key into their `navigationalContext` on at least the initial pass through the three protocol steps for the current set of End-User interactions with the Consumer's pages which incorporate the Portlet (i.e. if the **performBlockingInteraction** operation is not invoked, the `wsrp:newNavigationalContextScope` event MUST be sent to the Portlet prior to the first invocation of the **getMarkup** operation for each End-User's set of interactions with Consumer pages). This guarantees the Producer/Portlet at least has an opportunity to write a key into the `navigationalContext` prior to the initial invocation of **getMarkup** for an End-User. Portlets indicating they handle this event SHOULD be aware this might limit the set of Consumers willing to incorporate them on the Consumer's pages. As a simple notification, this event carries no payload, but does use an open content definition.

6.12 User Categories

A Producer's `ServiceDescription` MAY declare support for user categories. A Consumer MAY map End-Users to the user categories a Producer declares in any manner it chooses, including ignoring them. Producers that use user categories SHOULD implement appropriate default behavior in the event a Consumer does not assert any user category for the End-User.

A Portlet optionally declares the user categories for which it personalizes markup in the `PortletDescription` structure described in **[Section 5.1.16]**. The Consumer may assert any of these categories on behalf of a user. Since the Producer has no means of authenticating that the End-User belongs to one of these categories, this assertion should not be used for any security related purposes. If such an authentication is desired, standard security protocols should be employed to provide the authentication.

6.12.1 User Category Assertions

Since user categories are an optional means for the Producer and Consumer to cooperatively apply personalization that are relevant to the user, the following examines the various combinations of Producer and Consumer choices:

- Neither Producer nor Consumer support user categories. In this case the `PortletDescription` structures from the Producer will not declare any user categories and the Consumer will never assert any user categories in the `UserContext` structure.
- Both the Producer and Consumer support user categories. In this case the `PortletDescription` structures from the Producer will declare user categories. The Consumer will need to map its information about the user to this set from the Producer when asserting user categories in the `UserContext` structure in order to satisfy the requirement that the asserted user categories come only from the Producer published user categories [R417]. The Consumer controls the mechanism by which this mapping occurs.
- Producer supports user categories, but the Consumer does not. In this case the `PortletDescription` structures from the Producer declare user categories, but the Consumer will never assert any user categories in the `UserContext` structure. The Producer will need to default the user category it uses to process invocations.
- The Producer does not support user categories, but the Consumer does. In this case the `PortletDescription` structures from the Producer will not declare any user categories.

---

7 Registration Interface

A Producer that supports in-band registration of Consumers exposes the optional registration interface. Regardless of whether or not the registration portType is exposed, Producers can offer out-of-band processes to register a Consumer [R354]. All Producer registration processes MUST result in a unique, opaque token that may be used to refer to the registration. This specification calls this token a `registrationHandle` (defined in **[Section 5.1.25]**).

7.1 Data Structures

The following additional data structures are needed by this interface:

7.1.1 RegistrationData Type

The `RegistrationData` structure provides the means for the Consumer to supply the data required for registration with a Producer as well as protocol extensions that it supports [R355] [R356].

RegistrationData		
[R]	string	consumerName
[R]	string	consumerAgent
[R]	boolean	methodGetSupported
[O]	string	consumerModes[]
[O]	string	consumerWindowStates[]
[O]	string	consumerUserScopes[]
[O]	ExtensionDescription	extensionDescriptions[]
[O]	Property	registrationProperties[]
[O]	Extension	extensions[]

Members:

- `consumerName`: A name (preferably unique) that identifies the Consumer [R355] An example of such a name would be the Consumer's URL.
- `consumerAgent`: Name and version of the Consumer's vendor [R356]. The `consumerAgent` value MUST start with "productName.majorVersion.minorVersion" where "productName" identifies the product the Consumer installed for its deployment, and majorVersion and minorVersion are vendor-defined indications of the version of its product. This string can then contain any additional characters/words the product or Consumer wish to supply.
- `methodGetSupported`: A flag that tells the Producer whether the Consumer has implemented portlet URLs (regardless of whether they are written through Consumer URL rewriting or Producer URL writing, see **[Section 10.2]**) in a manner that supports HTML markup containing forms with method="get".
- `consumerModes`: An array of modes, beyond the required `wsrp:view` mode, the Consumer is willing to manage. This specification defines a set of constants for a base set of modes (see **[Section 13]**). This array may reference both those constants and additional custom modes of the Consumer.
- `consumerWindowStates`: An array of window states, beyond the required `wsrp:normal` window state, the Consumer is willing to manage. This specification defines a set of constants for a base set of window states (see **[Section 13]**). This array may reference both those constants and additional custom window states of the Consumer.
- `consumerUserScopes`: This field specifies the all the values for `UserScope` the Consumer is willing to process, including those defined by this specification. If the Consumer fails to supplies any values for this field, the Producer is free to specify any of the values defined by this specification.
- `extensionDescriptions`: An array of `ExtensionDescription` structures defining extensions the Consumer supports. This includes both incoming and outgoing extensions [R304] [R357] [R360].
- `registrationProperties`: List of registration properties. The names of these properties SHOULD be from the set declared in the `registrationPropertyDescription` from the Producer's `ServiceDescription` and are not part of this specification.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

7.2 register Operation

Registration describes how a Consumer establishes a relationship with a Producer that will be referenced via an

opaque handle in subsequent invocations the Consumer makes of the Producer [R350] [R352]. Both the Consumer and the Producer are free to end this relationship at any time [R500]. When the Consumer chooses to end the relationship, it MUST attempt an invocation of the **deregister** operation [R400], unless the registration is using a scheduled destruction, so that the Producer may release related items. When the Producer chooses to invalidate the registration identifier, it MUST inform the Consumer of this through a fault message on the next invocation specifying this `registrationHandle` so that the Consumer may release related items.

```
RegistrationContext = register (registrationData, lifetime,
userContext);

Faults: MissingParameters, OperationFailed,
```

If a Lifetime parameter is supplied, the Consumer is requesting the registration to be "leased" (i.e. use scheduled destruction) with the supplied value providing what the Consumer would otherwise provide on a subsequent **setRegistrationLifetime** invocation. If a Lifetime parameter is not supplied, the Consumer is indicating to not use scheduled destruction. The returned `RegistrationContext` is used in all subsequent invocations to reference this registration [R362]. If the Producer's metadata declares registration is not supported (i.e. `requiresRegistration` flag was set to "`false`"), then it MUST be valid to not supply a `RegistrationContext` to operations with this parameter. Whenever the registration attempt fails a fault message MUST be thrown indicating this to the Consumer [R363].

A Producer supporting registration MUST allow a Consumer to register itself multiple times with potentially different settings (e.g. billing settings) resulting in multiple `registrationHandles` [R351].

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

7.3 modifyRegistration Operation

This operation provides means for the Consumer to modify a relationship with a Producer [R353].

```
RegistrationState = modifyRegistration (registrationContext, registrationData, userContext);

Faults: InvalidRegistration, MissingParameters, OperationFailed, OperationNotSupported,
ResourceSuspended
```

The supplied parameters reference a pre-existing registration and the modifications desired. If the Producer chooses to have the Consumer provide enduring storage, the entire resulting registration state is carried in the `registrationState` field of the returned `RegistrationState` structure.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

7.4 deregister Operation

The Consumer MUST NOT consider a relationship with a Producer ended until either a successful invocation of **deregister** , elapsing of the scheduled destruction time or receipt of an `InvalidRegistration` fault message from the Producer on an invocation supplying the `registrationHandle`.

```
ReturnAny = deregister (registrationContext, userContext);

Faults: InvalidRegistration, OperationFailed,
        OperationNotSupported
```

After this operation is invoked, all handles created within the context of the `RegistrationContext` become invalid [R500] [R501] [R503]. It is a Producer implementation choice whether this immediately aborts in-progress operations or waits until all transient items time out. The Consumer MUST NOT use an invalidated `registrationHandle`, where the invalidation occurs either by passing the handle to **deregister** or by receiving a `InvalidRegistration` fault message from the Producer on an invocation supplying the handle. The Producer MUST return a `InvalidRegistration` fault message whenever a Consumer supplies an invalid `registrationHandle`. If the **deregister** operation fails, the Producer MUST return a fault message specifying the reason for the failure.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

7.5 getRegistrationLifetime Operation

This operation allows a Consumer to refresh its understanding of the scheduled destruction for a registration.

```
Lifetime = getRegistrationLifetime (registrationContext, userContext);

Faults: AccessDenied, InvalidHandle, InvalidRegistration, ModifyRegistrationRequired,
        OperationFailed, OperationNotSupported, ResourceSuspended,
```

If the nillable response from **getRegistrationLifetime** is nil, then scheduled destruction is not in use for this registration and the Consumer MUST use the **deregister** operation to destroy the registration.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

7.6 setRegistrationLifetime Operation

This operation allows a Consumer to request a change to the scheduled destruction of a registration. The Producer returns the actual change that was made.

```
Lifetime = setRegistrationLifetime (registrationContext, userContext, lifetime);

Faults: AccessDenied, InvalidHandle, InvalidRegistration, ModifyRegistrationRequired,
        OperationFailed, OperationNotSupported, ResourceSuspended,
```

If the nillable `Lifetime` parameter is nil, then this is a request to not use scheduled destruction as Consumer will use the **deregister** operation to destroy the registration. A nil response indicates that the Producer is not using scheduled destruction for this registration.

---

8 Portlet Management Interface

Producers MUST expose one or more logically distinct ways of generating markup and handling interactions with that markup [A205], which this specification refers to as Portlets. The Producer declares the Portlets it exposes through its description [A104]. This declaration contains a number of descriptive parameters; in particular it includes a `portletHandle` that Consumers use to refer to the so-called "Producer Offered Portlet". These Portlets are pre-configured and non-modifiable by Consumers.

In addition to the Producer Offered Portlets, a Producer can expose the `PortletManagement` portType and thereby allow Consumers to clone and customize the Portlets the Producer offers. A Consumer MAY request a unique configuration of one of these Portlets, either in an opaque manner (e.g. the "edit" button common on aggregated pages which invokes a Portlet-generated page for setting the configuration) or by using the property definitions found in the Portlet's metadata to configure it in an explicit manner [R600]. Such a configured Portlet is called a "Consumer Configured Portlet".

Any Producer that supports cloning Portlets on **performBlockingInteraction** or **handleEvents** invocations MUST support the **destroyPortlets** operation.

8.1 Data Structures

The following additional data structures are needed by this interface:

8.1.1 FailedPortlets Type

The `FailedPortlets` structure contains a set of `portletHandles` which failed to be processed for the same reason and the reason for the failure.

FailedPortlets		
[R]	Handle	portletHandles[]
[R]	ErrorCodes	errorCode
[O]	LocalizedString	reason
[O]	Extension	extensions[]

Members:

- `portletHandles`: An array of `portletHandle` which all failed to be processed for the supplied reason.
- `errorCode`: One of the enumerated `ErrorCodes`, describing why processing failed.
- `reason`: An explanation of the failure encountered, which is intended for display to an End-User.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.2 DestroyPortletsResponse Type

The `DestroyPortletsResponse` structure carries an array of failed destroys.

DestroyPortletsResponse		
[O]	FailedPortlets	failedPortlets[]

[O] Extension	extensions[]

Members:

- `failedPortlets`: An array of failures returned by **destroyPortlets**. This is carried as a return message since not all web stacks properly handle typed information in fault messages.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.3 PortletDescriptionResponse Type

The `PortletDescriptionResponse` structure contains the fields that **getPortletDescription** can return.

PortletDescriptionResponse	
[R] PortletDescription	portletDescription
[O] ResourceList	resourceList
[O] Extension	extensions[]

Members:

- `portletDescription`: The metadata for the Portlet.
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.4 PortletPropertyDescriptionResponse Type

The `PortletPropertyDescriptionResponse` structure contains the fields that **getPortletPropertyDescription** can return.

PortletPropertyDescriptionResponse	
[O] ModelDescription	modelDescription
[O] ResourceList	resourceList
[O] Extension	extensions[]

Members:

- `modelDescription`: The description of the Portlet's properties [R602].
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.5 CopiedPortlet Type

The `CopiedPortlet` structure provides the Consumer with the details for a Portlet the **copyPortlets** operation has generated.

CopiedPortlet		
[R]	Handle	fromPortletHandle
[R]	PortletContext	newPortletContext
[O]	Extension	extensions[]

Members:

- `fromPortletHandle`: The `portletHandle` used as the source for the copy operation.
- `newPortletContext`: The newly generated `PortletContext`.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.6 CopyPortletsResponse Type

The `CopyPortletsResponse` structure contains the fields that **copyPortlets** can return.

CopyPortletsResponse		
[O]	CopiedPortlet	copiedPortlets[]
[O]	FailedPortlets	failedPortlets[]
[O]	ResourceList	resourceList
[O]	Extension	extensions[]

Members:

- `copiedPortlets`: An array returning information for the successfully copied Portlets.
- `failedPortlets`: An array returning information about the Portlets where the copy failed.
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.7 ExportedPortlet Type

The `ExportedPortlet` structure represents a single exported Portlet.

ExportedPortlet		
[R]	Handle	portletHandle
[R]	base64Binary	exportData
[O]	Extension	extensions[]

Members:

- `portletHandle`: The `portletHandle` of the Portlet whose exported data is represented by this structure.

- `exportData:` The portlet-specific data which will be supplied to the **importPortlets** operation. As such, it needs to contain all data and references needed for the **importPortlets** operation to reconstitute the Portlet.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.8 ExportPortletsResponse Type

The `ExportPortletsResponse` structure contains the fields that **exportPortlets** can return. The **exportPortlets** operation can export many Portlets in a single invocation and this structure represents the bulk response.

ExportPortletsResponse		
[O]	base64Binary	exportContext
[O]	ExportedPortlet	exportedPortlets[]
[O]	FailedPortlets	failedPortlets[]
[O]	Lifetime	lifetime
[O]	ResourceList	resourceList
[O]	Extension	extensions[]

Members:

- `exportContext:` An opaque data structure containing common data that will be needed to import any of the successfully exported Portlets. This field is only returned if there is such common data and at least one Portlet was successfully exported.
- `exportedPortlets:` An array of `ExportedPortlet` structures where each exported Portlet MUST be represented by a single entry in the array.
- `failedPortlets:` An array returning information about the Portlets where the export failed.
- `lifetime:` This field indicates that the exported data references data stored at the Producer and states how long the Producer intends to make this data available. If this field is not supplied, the export data contains all the information needed to import the Portlets and the Consumer is solely responsible for managing its lifetime.
- `resourceList:` This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.9 ImportPortlet Type

The `ImportPortlet` structure represents a single exported Portlet representation which the Consumer wishes to import.

ImportPortlet		
[R]	ID	importID
[R]	base64Binary	exportData
[O]	Extension	extensions[]

Members:

- `importID:` A Consumer defined identifier for the Portlet being imported. Unlike export, where the `portletHandle` can be used to identify both the Portlet to export and the resulting exported data, the import process will generate a new `portletHandle`. As Consumers still need to match requested imports with their respective reconstitued Portlets, a different mechanism is needed. In this case, a Consumer defined identifier is used. Consumers supply a `importID` with each import record. The Producer returns this `importID` with the resultant reconstituted `PortletContext` in the result. By matching these `ID`s, the Consumer can properly match each returned `PortletContext` with the Consumer's intended usage.
- `exportData:` The exported representation of the Portlet to be imported.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.10 ImportedPortlet Type

The `ImportedPortlet` structure represents a single response for the import of a Portlet representation.

ImportedPortlet		
[R]	ID	importID
[R]	PortletContext	newPortletContext
[O]	Extension	extensions[]

Members:

- `importID:` The `ID` which the Consumer supplied for identifying a particular Portlet representation.
- `newPortletContext:` The newly generated `PortletContext` representing the reconstituted Portlet.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.11 ImportPortletsFailed Type

The `ImportPortletsFailed` structure provides the Consumer with the details for a set of Portlets the **importPortlets** operation failed to process into reconstituted Portlets.

ImportPortletsFailed		
[R]	ID	importIDs[]
[R]	ErrorCodes	errorCode
[R]	LocalizedString	reason
[O]	Extension	extensions[]

Members:

- `importIDs:` An array of Consumer supplied `ID`s which all failed to be imported for the supplied `reason`.
- `errorCode:` One of the enumerated `ErrorCodes`, describing why the processing of the referenced Portlets failed.
- `reason:` An explanation of the failure encountered which is intended for display to an End-User. Note that while it is recommended Portlets with common failures be grouped into a single `ImportPortletsFailed`, it is not required. In other words, it is permissible to have duplicate `errorCode` and `reason` values in the failed

array of the `ImportPortletsResponse`.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.12 ImportPortletsResponse Type

The `ImportPortletsResponse` structure contains the fields that **importPortlets** can return. The **importPortlets** operation can import many Portlets in a single invocation and this structure represents the bulk response.

ImportPortletsResponse		
[O]	ImportedPortlet	importedPortlets[]
[O]	ImportPortletsFailed	importFailed[]
[O]	ResourceList	resourceList
[O]	Extension	extensions[]

Members:

- `importedPortlets:` An array of `ImportedPortlet` structures where each member in the array represents a single imported Portlet.
- `importFailed:` An array returning information about the Portlets where the import failed.
- `resourceList:` This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.13 PortletLifetime Type

The `PortletLifetime` structure represents the lifetime for a single Portlet.

PortletLifetime		
[R]	PortletContext	portletContext
[R]	Lifetime	scheduledDestruction
[O]	Extension	extensions[]

Members:

- `portletContext:` The `PortletContext` to which the `Lifetime` pertains.
- `scheduledDestruction:` This field informs the Consumer of the scheduled destruction of the Portlet.
- `extensions:` The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.14 GetPortletsLifetimeResponse Type

The `GetPortletsLifetimeResponse` structure contains the fields that **getPortletsLifetime** can return. The **getPortletsLifetime** operation can update many Portlets in a single invocation and this structure represents the bulk response.

GetPortletsLifetimeResponse	
[O] PortletLifetime	portletLifetimes[]
[O] FailedPortlets	failedPortlets[]
[O] ResourceList	resourceList
[O] Extension	extensions[]

Members:

- `portletLifetimes`: An array of `PortletLifetime` structures where each member in the array represents a single Portlet.
- `failedPortlets`: An array returning information about the Portlets where the operation failed.
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.1.15 SetPortletsLifetimeResponse Type

The `SetPortletsLifetimeResponse` structure contains the fields that **setPortletsLifetime** can return. The **setPortletsLifetime** operation can update many Portlets in a single invocation and this structure represents the bulk response.

SetPortletsLifetimeResponse	
[O] PortletLifetime	updatedPortlets[]
[O] FailedPortlets	failedPortlets[]
[O] ResourceList	resourceList
[O] Extension	extensions[]

Members:

- `updatedPortlets`: An array of `SetPortletsLifetime` structures where each member in the array represents a single updated Portlet.
- `failedPortlets`: An array returning information about the Portlets where the update failed.
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions`: The `extensions` field MAY be used to extend this structure. Extension elements MUST be from namespaces other than WSRP.

8.2 getPortletDescription Operation

This operation allows a Producer to provide information about the Portlets it offers in a context-sensitive manner.

```
PortletDescriptionResponse = getPortletDescription (registrationContext, portletContext,
userContext, desiredLocales);
```

```
        AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration,
Faults: InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed,
        OperationNotSupported, ResourceSuspended
```

Producers may choose to restrict access to the information returned in `PortletDescriptionResponse` based on the supplied registration and user contexts. Consumers may choose to alter how they interact with a Portlet based on the metadata contained in the returned `PortletDescriptionResponse`. For security reasons related to exposing the existence of something the caller is not allowed to access, it is RECOMMENDED that a AccessDenied fault be generated both for the case of the supplied `portletHandle` not being a valid reference in the scope of the supplied `registrationHandle` and for the case of the user not having access to a valid reference (i.e. by definition access is denied when the Portlet reference is invalid).

When generating the `PortletDescriptionResponse` the Producer SHOULD use the `desiredLocales` to control what locales are supplied for localized strings.

8.3 clonePortlet Operation

This operation allows the Consumer to request the creation of a new Portlet from an existing Portlet.

```
PortletContext = clonePortlet (registrationContext, portletContext, userContext, lifetime);


        AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration,
Faults: InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed,
        OperationNotSupported, ResourceSuspended
```

The supplied `PortletContext` MUST refer to either a Producer Offered Portlet or a previously cloned Consumer Configured Portlet. The initial state of the new Portlet MUST be equivalent to the state of the Portlet referenced by the supplied handle. The `PortletDescription` for the supplied `portletHandle` will apply to the newly cloned Portlet as well. In the case of a Consumer Configured Portlet that returns the Portlet's enduring state to the Consumer, the `portletState` field of the returned `PortletContext` structure will supply that state. The new `portletHandle` MUST be scoped by the `registrationHandle` in the supplied `RegistrationContext` and be unique within this registration.

If a Lifetime parameter is supplied, the Consumer is indicating a preference for the Portlet to be "leased" with the supplied value providing what the Consumer would otherwise provide on a subsequent **setPortletsLifetime** invocation. If a Lifetime parameter is not supplied, the Consumer is indicating a preference for the Portlet to not use scheduled destruction.

If a Producer chooses to return the enduring state of its Portlets to the Consumer, it is RECOMMENDED that the `portletHandle` encode the supplied `registrationHandle`. In this case, it is also RECOMMENDED that the `portletState` encode the `portletHandle` so that the Producer can do reasonable cross checks that it is receiving a consistent set of handles and state.

The returned `PortletContext` contains both the `portletHandle` and `portletState` fields for use in subsequent invocations on the configured Portlet. No relationship between the supplied Portlet and the new Portlet is defined by this specification. The Consumer attempts to release the new `portletHandle` by invoking **destroyPortlets** when it is no longer needed.

If the Consumer has not registered, then the Consumer MUST invoke **destroyPortlets** when it would have

deregistered, passing each `portletHandle` that would have been scoped by a registration and were not using scheduled destruction.

8.4 destroyPortlets Operation

The Consumer MUST inform the Producer that a Consumer Configured Portlet which does not use leasing will no longer be used by invoking either the **destroyPortlets** or the **deregister** operation and MUST NOT consider such a Portlet to have been destroyed until one of these operations has been successfully invoked for that Portlet.

```
DestroyPortletsResponse = destroyPortlets (registrationContext, portletHandles[],
userContext);

Faults: InconsistentParameters, InvalidRegistration, MissingParameters,
ModifyRegistrationRequired, OperationFailed, OperationNotSupported, ResourceSuspended
```

The supplied `portletHandles` is an array of type `portletHandle`, each of which the Consumer is informing the Producer it will no longer use. The Producer returns failures to destroy supplied `portletHandles` in the `DestroyPortletsResponse` structure. It is a choice of the Producer's implementation whether the items related to the `portletHandles` are immediately reclaimed or whether transient items are allowed to timeout first. A Consumer MUST NOT reference any of the supplied `portletHandles` after successfully invoking **destroyPortlets** and MAY reclaim items related to the supplied `portletHandles` (e.g. `portletState`).

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

8.5 getPortletsLifetime Operation

This operation allows a Consumer to refresh its understanding of the scheduled destruction for a set of Portlet.

```
GetPortletsLifetimeResponse = getPortletsLifetime (registrationContext, portletContexts[],
userContext);

Faults: AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration,
ModifyRegistrationRequired, OperationFailed, OperationNotSupported, ResourceSuspended
```

If the optional `Lifetime` field from **getPortletsLifetime** is not returned for a particular Portlet, then scheduled destruction is not in use for this Portlet and the Consumer MUST use the **destroyPortlets** operation to destroy the Portlet.

8.6 setPortletsLifetime Operation

This operation allows a Consumer to request a change to the scheduled destruction of a set of Portlets. The Producer returns the actual changes that were made.

```
SetPortletsLifetimeResponse = setPortletsLifetime (registrationContext, portletContext[],
userContext, lifetime);
```

Faults:	AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration, ModifyRegistrationRequired, OperationFailed, OperationNotSupported, ResourceSuspended

If the nillable `Lifetime` parameter is nil, then this is a request to not use scheduled destruction as Consumer will use the **destroyPortlets** operation to destroy the Portlets. A nil response indicates that the Producer is not using scheduled destruction for this Portlet.

8.7 copyPortlets Operation

This operation provides the means for the Consumer to make new copies of a set of Portlets, potentially specifying a different registration scope for the new Portlets.

Since Consumers will commonly invoke this operation as part of a migration effort, this operation is a bulk operation. The Consumer supplies a list of Portlets it wants to copy. The Producer's response contains exactly one element for each entry in the supplied list.

`CopyPortletsResponse = copyPortlets (toRegistrationContext, toUserContext, fromRegistrationContext, fromUserContext, fromPortletContexts[], lifetime);`	
Faults:	AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration, InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed, OperationNotSupported, ResourceSuspended

The supplied `toRegistrationContext` parameter provides the `RegistrationContext` that scopes the set of new Portlets being requested. If the `toRegistrationContext` parameter is not supplied, the new Portlets are to be scoped by the same `RegistrationContext` as the existing Portlets (i.e. the `fromRegistrationContext`). The supplied `toUserContext` provides information concerning the End-User requesting creation of the new Portlets in the `toRegistrationContext`. A null value means the Consumer is not indicating which End-User is making the request. The supplied `fromRegistration` parameter provides the `RegistrationContext` which scopes the set of Portlets supplied as the source for generating the new Portlets. The supplied `fromUserContext` provides information concerning the End-User requesting the new copy in the `fromRegistrationContext`. A null value means the Consumer is not indicating which End-User on whose behalf the request is being made. The `fromPortletContexts` parameter specifies the set of Portlets which are to serve as the basis for generating new Portlets. The returned `CopyPortletsResponse` contains information on both the set of new Portlets generated and a reason for each Portlet where the copy failed to generate a new Portlet.

If a Lifetime parameter is supplied, the Consumer is indicating a preference for the Portlets to be "leased" with the supplied value providing what the Consumer would otherwise provide on subsequent **setPortletsLifetime** invocations. If a Lifetime parameter is not supplied, the Consumer is indicating a preference for the Portlets to not use scheduled destruction.

The returned `CopyPortletsResponse` contains arrays for both the set of successfully copied Portlets and reasons for each Portlet where the attempt failed. When not returning a fault, the Producer MUST return exactly one entry in its response for each entry in the request to **copyPortlets**.

Consumers should be aware that processing invocations of **copyPortlets** may take significant time and should therefore break large bulk requests into chunks. However, even when the Consumer limits the request in this manner, the Producer is free to evaluate the cost of any given request and adjust the number of Portlets it copies in the response.

8.8 exportPortlets Operation

This operation allows the Consumer to get an opaque representation of a Portlet which can be supplied to the corresponding import operation to reconstitute the Portlet.

Since Consumers will commonly invoke this operation when taking a snapshot of all or a portion of itself (e.g. to package itself for deployment or migration), this operation is a bulk operation. The Consumer passes a list of Portlets for which it wants to obtain a representation. The Producer's response contains exactly one element for each entry in the supplied array of Portlets.

```
ExportPortletsResponse = exportPortlets (registrationContext, portletContext[], userContext,
lifetime, exportByValueRequired);


       AccessDenied, ExportByValueNotSupported, InconsistentParameters, InvalidHandle,
Faults: InvalidRegistration, InvalidUserCategory, MissingParameters,
       ModifyRegistrationRequired, OperationFailed, OperationNotSupported, ResourceSuspended
```

The supplied `RegistrationContext` sets the scope for the set of Portlets requested to be exported. The array of `PortletContext` elements specifies the set of Portlets where exported data is being requested. The `UserContext` parameter provides information about the End-User making the request. The `exportByValueRequired` parameter is a boolean indicating whether or not the usage intended by the Consumer requires that the export not contain references to data stored at the Producer. If the Consumer requires the Producer to export the Portlets by value and the Producer does not support such exports, the `ExportByValueNotSupported` fault MUST be returned.

If a Lifetime parameter is supplied, the Consumer is indicating a preference for any items referenced by the export to be "leased" with the supplied value providing what the Consumer would otherwise provide on subsequent **setExportLifetime** invocations. If a Lifetime parameter is not supplied, the Consumer is indicating a preference for any items referenced by the export to not use scheduled destruction. This parameter only applies when the Producer's export is not by value and is otherwise ignored.

The returned `ExportPortletsResponse` contains arrays for both the set of successfully exported Portlets and reasons for each Portlet where the export attempt failed. When not returning a fault, the Producer MUST return exactly one entry in its response for each entry in the request to **exportPortlets**.

Consumers should be aware that processing invocations of **exportPortlets** may take significant time and should therefore break large bulk requests into chunks. The Producer indicates its preferred request size in the `recommendedExportSize` field of its `ServiceDescription`. Consumers SHOULD limit their export requests to chunks no larger then this size. However, even when the Consumer limits the request to this size, the Producer is free to evaluate the cost of any given request and adjust the number of Portlets it exports in the response.

Note that the Producer is responsible for identifying the correct offered Portlet to clone from during an import. Producers should be aware that the importing Producer may be in a different place, time, and version from the exporting Producer. The mechanism for identifying the correct offered Portlet should be sufficiently tolerant to work in such situations. For the same reasons, the importing Producer should be capable of dealing with version differences between the representation of the Portlet in the exported data and the imported Portlet. Both of these imply a need to carry extra information in the `exportData` that allows the importing Producer to resolve such details.

8.9 importPortlets Operation

The **importPortlets** operation reconstitutes a set of previously exported Portlets.

Since Consumers will commonly invoke this operation when reconstituting a particular configuration, this operation is a bulk operation. The Consumer passes a list of Portlets which it wants to be reconstituted. The Producer response MUST contain exactly one element for each entry in the list supplied to **importPortlets**.

```
ImportPortletsResponse = importPortlets (registrationContext, importContext, importPortlet[],
userContext, lifetime);


       AccessDenied, ExportNoLongerValid, InconsistentParameters, InvalidRegistration,
Faults: InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed,
       OperationNotSupported, ResourceSuspended
```

The `importContext` carries the data from the `exportContext` field associated with the Portlet representations by a response to a single invocation of the **exportPortlets** operation. The `importPortlet` array identifies the Portlets to be reconstituted. The `userContext` parameter provides information about the End-User making the request. The reconstituted Portlets or failure messages are returned in a corresponding array in the response. The relationship between the Portlet requested to be imported and the resulting reconstituted Portlet passed back in the response is maintained by a Consumer supplied identifier (`importID`).

If a Lifetime parameter is supplied, the Consumer is indicating a preference for the Portlets to be "leased" with the supplied value providing what the Consumer would otherwise provide on subsequent **setPortletsLifetime** invocations. If a Lifetime parameter is not supplied, the Consumer is indicating a preference for the Portlets to not use scheduled destruction.

8.10 releaseExport Operation

The **releaseExport** operation provides the means for the Consumer to indicate to the Producer that it no longer needs to maintain any stored artifacts relative to a particular export.

```
ReturnAny = releaseExport (exportContext,
userContext);
```

All that is needed to release an export is the `exportContext` from the original **exportPortlets** invocation. As defined, the **releaseExport** operation returns nothing to the Consumer and throws no fault. As a result, the Producer has full responsibility to recover from any failures associated with the attempt to release artifacts.

The primary purpose the nillable `userContext` is provided to this operation is Producer logging.

8.11 setExportLifetime Operation

The **setExportLifetime** operation provides the means for the Consumer to request that the Lifetime of a particular export be changed.

```
Lifetime = setExportLifetime (registrationContext, exportContext, userContext, lifetime);
```

```
Faults: AccessDenied, InvalidHandle, InvalidRegistration, ModifyRegistrationRequired,
        OperationFailed, OperationNotSupported, ResourceSuspended
```

This operation is only valid when the Producer has indicated that it has stored artifacts relative to an export by having returned a `Lifetime` structure. The Consumer indicates the particular export it is requesting the lifetime change for using the `exportContext` parameter and the desired new lifetime using the `lifetime` parameter.

8.12 setPortletProperties Operation

The Portlet state in the previous operations was opaque to the Consumer (e.g. as `portletState`). In addition, means are defined by which a Producer may publish information about state in a Portlet-specific manner. This is enabled through **Properties** that are declared in the metadata specific to a Portlet. This operation enables the Consumer to interact with this published portion of a Portlet's state.

```
PortletContext = setPortletProperties (registrationContext, portletContext, userContext,
propertyList);

        AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration,
Faults: InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed,
        OperationNotSupported, ResourceSuspended
```

Since **setPortletProperties** is interacting only with the published portion of the Portlet's state, it is always safe for the Portlet to modify its state (i.e. equivalent to `portletStateChange` set to "`readWrite`" for a **performBlockingInteraction** invocation). The supplied set of property changes MUST be processed together. In particular, validation SHOULD only be done considering the entire set, as partial updates could easily create an internally inconsistent set of properties. The storage of the update caused by applying the set of property updates SHOULD only occur after the Producer/Portlet executes this validation. The Producer SHOULD serialize invocations of **setPortletProperties** for any one `portletHandle`. Note that changing a property's value could impact the value of any of the Portlet's properties.

8.13 getPortletProperties Operation

This operation provides the means for the Consumer to fetch the current values of the published Portlet's properties. The intention is to allow a Consumer-generated user interface to display these for administrative purposes.

```
PropertyList = getPortletProperties (registrationContext, portletContext, userContext, names);

        AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration,
Faults: InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed,
        OperationNotSupported, ResourceSuspended
```

The supplied `names` parameter is an array of strings each of which declares a property for which the Consumer is requesting its value. The returned `PropertyList` declares the current values for these properties. If the Consumer does not pass a `names` parameter, the Producer / Portlet MUST treat this as a request to enumerate the properties of the Portlet.

8.14 getPortletPropertyDescription Operation

This operation allows the Consumer to discover the published properties of a Portlet and information (e.g. type and description) that could be useful in generating a user interface for editing the Portlet's configuration.

```
PortletPropertiesDescriptionResponse = getPortletPropertyDescription (registrationContext,
portletContext, userContext, desiredLocales);


       AccessDenied, InconsistentParameters, InvalidHandle, InvalidRegistration,
Faults: InvalidUserCategory, MissingParameters, ModifyRegistrationRequired, OperationFailed,
       OperationNotSupported, ResourceSuspended
```

The `modelDescription` returned in the `PortletPropertyDescriptionResponse` structure is a typed `Property` view onto the portion of the Portlet's enduring state that the user (referenced through the `userContext`) is allowed to modify. While it is possible the set of properties can change with time (e.g. the Portlet dynamically creates or destroys properties), Producers and Portlets SHOULD make the returned `modelDescription` as complete as possible.

When generating the `PortletPropertyDescriptionResponse` the Producer SHOULD use the `desiredLocales` to control which locales are supplied for localized strings.

---

9 Security

Systems compliant with this specification will be exposed to the same security issues as other web service systems [A613]. For a representative summary of security concerns, refer to the Security and Privacy Considerations document produced by the XML-Based Security Services Oasis TC . One area to particularly note is that security information, including fields within the WSRP protocol that relate to a need for secure communication, is only as trustworthy as the means used to transmit the information.

It is a goal within this specification to leverage standards efforts that address web services security and to avoid defining mechanisms that will be redundant with those standards efforts. In particular, the Technical Committee is producing a **[Security Tech Note]** which will call out the security concerns of the WSRP use case and provide pointers at relevant security-oriented work. The standards in the security area generally fall into two main categories: document-level mechanisms and transport-level mechanisms.

Producers and Consumers wishing to apply document-level security techniques are encouraged to adhere to the mechanisms defined by WS-Security, SAML, XML-Signature, XML-Encryption, and related specifications. It is anticipated that as the web services security roadmap becomes more fully specified by standards, and support for those standards becomes widely available from infrastructure components, that these will play an increasingly important role in the use of this specification [R414]. At the writing of this specification, there does not exist a standard for expressing policy issues, though several have been proposed. Consumers and Producers will need to work out how to exchange policy information outside of this protocol.

Use of transport-level security standards (e.g. SSL/TLS) to address the security issues involved in Consumers invoking Producers on behalf of End-Users only require that a Producer's WSDL declare ports for an HTTPS service entry point. Consumer's can only determine that secure transport is supported by parsing the URL for the service entry point [R413].

9.1 Authentication of Consumer

Producer authentication of a Consumer may be achieved at the transport level through the use of client certificates in conjunction with SSL/TLS. Certificate provisioning by a Producer to a Consumer happens outside the scope of this protocol, typically as part of establishing a business relationship between the Producer and Consumer [R402].

The WS-Security standard also specifies the means by which tokens, including those asserting the identity of the Consumer, may be included in a SOAP message. A number of profiles for suppling tokens that can be used for this purpose have already been defined. Since additional profiles continue to be defined, implementors are encouraged to consult the WS-Security TC's web site for the latest information [R402].

9.2 Confidentiality & Message Integrity

The WS-Security standard also specifies the means by which security features can be used to ensure SOAP message confidentiality and integrity. When employing message level security, implementors are encouraged to leverage these means in order to achieve message confidentiality or integrity, as needed [A604].

SSL/TLS may be used to ensure the contents of messages are neither tampered with nor decipherable by an unauthorized third party [A604]. Consideration needs to be given to both the communication between Producer and Consumer and communication between the End-User client and the Consumer.

For Producer - Consumer communication, the Producer declares the use of SSL/TLS in the service's WSDL by declaring an HTTPS service endpoint.

For Consumer - End-User client communication, the Consumer indicates in the `MarkupParams` structure whether or not communication with the End-User is happening in a secure manner. The Portlet can choose to change behavior based on this value, for example it may generate markup that redirects the End-User to the equivalent secure page or throw a fault indicating secure client communication are required.

9.3 Access control

A Consumer can implement access control mechanisms that restrict which End-Users may access which Portlets and operations on those Portlets. Additionally, a Producer can implement access control programmatically through the use of facilities such as an authenticated user identity [A605].

It should be noted that the security concept of access control is generally controlled by subsystems/protocols operating at a lower layer than the WSRP implementation/protocol and that this should not be confused with the application level concept of user categories, even when applications use the user categories to determine whether or not to show particular portions of the overall user interface.

---

10 Markup

This section covers the issues related to Portlets generating markup that Consumers could safely aggregate into a page and then properly process End-User interactions with the resulting aggregated page [A301].

10.1 Encoding

The Consumer indicates to the Portlet the preferred character encoding, using the `markupCharacterSets` field of the `MarkupParams` structure. It is up to the Portlet to generate markup that complies with this encoding. The Portlet is allowed to generate its markup in either the UTF-8 or UTF-16 character set encoding if it is unable to generate the requested character set. If it is unable to generate markup in any of these character sets, the Portlet MUST return a fault message to the Consumer. When the SOAP binding is in use, the nature of XML requires that the markup use the same character set as the XML response message.

10.2 URL Considerations

As part of its markup, a Portlet will often need to create URLs that reference the Portlet itself. When an End-User activates such an URL, by clicking a link or submitting a form, the result should be a new invocation targeted to the Portlet. This section describes the different possibilities for how the Portlet can encode these "portlet URLs" in its markup.

The portlet URLs embedded in a markup fragment often can not (or should not) be direct links to the Producer for a number of reasons:

- URLs the Portlet writes in its markup will be invoked or accessed by the End-User operating on the client. In the general case however it is only guaranteed that the client has direct access to the Consumer; the Producer may be shielded from the client via a firewall. So the Consumer needs to intercept URLs and route them to the Producer [A103].
- The Consumer may want to intercept URLs to perform additional operations, enrich the request with context information or do some book keeping.
- The client might not be able to directly invoke the Producer, e.g. if the client is a user-agent that cannot issue SOAP requests to the Producer but can only talk HTTP to the Consumer. In this case the Consumer must translate the request into the correct protocol.

This implies that portlet URLs must be encoded so that the Consumer intercepts them and re-routes them to the correct Portlet at the Producer [A305], including the proper context. Because the same Portlet can be instantiated more than once in a single aggregated page, portlet URLs will have to allow the Consumer to track the Portlet to which the request is targeted. The problem is that the Producer requires Consumer-dependent information to write such a link. In principle there exist two options to make a portlet URL point back to the Consumer and consist of all information necessary for the Consumer to properly process an activation of the URL:

1. The Consumer can pass information on its context to the Portlet. The Portlet exploits this information during URL encoding so the resulting URL can be passed without further modification to the client. The advantages of this technique are efficiency and exploitation of these settings, even in client-side scripting. The disadvantages include that the Portlet will not be able to serve static content as the content depends on Consumer runtime settings and that the resulting markup might be less cacheable by the Consumer.
2. The Portlet can use a special syntax to encode portlet URLs. It is then the task of the Consumer to detect portlet URLs in the markup and modify them according to its requirements. The markup generated by the Portlet is now Consumer-independent, allowing the Portlet to exploit caching mechanisms or even to serve static content. However, the Consumer will be more complex, as it needs to parse the markup to locate and rewrite portlet URLs, and require additional processing time. Consumers can minimize this impact on performance by using efficient encoding and parsing mechanisms (for example, the Boyer-Moore algorithm [17]).

As there is no clear advantage to either technique, both styles of portlet URL encoding are supported (see **[Section 10.2.1]** and **[Section 10.2.2]**). This facilitates the capabilities both of the Producer and the Consumer with regards to the ability to adapt the generated markup and requires that the following semantics be followed:

- If a Portlet's metadata declares it is willing to process URL templates, then the Consumer supplies templates for the Portlet to use
- As the Portlet writes URLs into the markup it MUST encode them based on the mime type. For example, XML based markup requires that all &' characters have to be encoded as "&".
- If a Portlet is unable to completely write the portlet URLs for its markup, it MUST set the `requiresRewriting` flag in the returned `MarkupContext` structure to "`true`".
- If the `requiresRewriting` flag in the `MarkupContext` structure is "`true`", then the Consumer parses the returned markup and rewrites URLs conforming to the definitions in **[Section 10.2.1]** of this specification.

Note: In principle it would not be necessary to mark portlet URLs in a special way. The Consumer could always analyze the markup semantically and syntactically, detect portlet URLs and rewrite them. This approach however would be very difficult and time consuming to implement for the Consumer, for reasons including that such a rewriting algorithm would be dependent on the markup type and version. Therefore both the Consumer and the Producer URL writing scenarios are introduced for convenience.

Portlets MUST adopt the following convention for including non-ASCII characters within portlet URLs in order to comply with W3C recommendations.

- Represent each character in UTF-8 (see [RFC2279]) as one or more bytes.
- Escape these characters with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the character value).

This procedure results in a syntactically legal URI (as defined in RFC1738, section 2.2 or RFC2141, section 2) that is independent of the character encoding[18] to which the document carrying the URI may have been transcoded.

Since the values a Portlet provides will appear as either an URL parameter value or as part of the path of an URL, these values it MUST be strictly encoded (i.e. "&", "=", "/", and "?" need to be url-escaped) so that special URL characters do not invalidate the processing of the enclosing URL.

When URL activation occurs, the Consumer MUST process all mode and window state change requests and either honor and reject them prior to invoking the operation indicated by the URL. If the requested mode or window state is for a value that is either invalid or unavailable, the Consumer SHOULD leave the current value unchanged.

10.2.1 Consumer URL Rewriting

All portlet URLs (i.e. those the Consumer needs to rewrite) are demarcated in the markup by a token (wsrp_rewrite) both at the start (with a "?" appended to clearly delimit the start of the name/value pairs) and end (proceeded by a "/" to form the end token) of the URL declaration. The Consumer will have to locate the start and end token in the markup stream and use the information between the tokens to rewrite the portlet URL correctly. Details on this URL writing process are Consumer-specific and out of scope for this specification. The content between this pair of tokens follows the pattern of a query string (name/value pairs separated by "&" characters) with several well-known "portlet URL parameter" names specifying the information the Consumer needs in order to both correctly rewrite the URL and then process it when an End-User activates it. This results in an portlet URL declaration of the form:

```
wsrp_rewrite?wsrp-urlType=value&name1=value1&name2=value2 ... /
wsrp_rewrite
```

Consumers MUST accept both the "&" character and the corresponding entity reference (i.e. "&") as separators between the name/value pairs as this allows Portlets to produce markup fragments valid for a larger range of mime types. We encourage Portlets to use the entity reference form ("&") in static markup as this is likely to result in the ability to include that markup in a larger set of enclosing mime types. The Consumer is NOT REQUIRED to process URLs not conforming to this format and MAY choose to pass them unchanged, replace them with error text, do a best effort processing or invalidate the entire markup fragment. The Consumer is NOT REQUIRED to process escaped characters in parameter names or values, but rather MAY pass them unchanged to either the user-agent (during URL rewriting) or the Producer (during processing of an activated URL).

The following well-known portlet URL parameter names (e.g. replacing "*wsrp-urlType*=value", "name1" and "name2" above) are defined:

10.2.1.1 wsrp-urlType

This parameter MUST be specified first when using the Consumer URL rewriting template and the value selected from the following definitions. Well-known portlet URL parameter names that are valid for only one `wsrp-urlType` are described relative to that `wsrp-urlType` while the remainder are described later. The following values are defined for `wsrp-urlType`:

10.2.1.1.1 wsrp-urlType = blockingAction

Activation of the URL will result in an invocation of **performBlockingInteraction** on the Portlet that generated the markup. All form parameters, submitted as query string parameters using the HTTP GET method, that are not used to encode parameters defined by this specification MUST be passed to **performBlockingInteraction** as `formParameters`.

10.2.1.1.2 wsrp-urlType = render

Activation of the URL will result in an invocation of **getMarkup**. This mechanism permits a Portlet's markup to contain URLs, which do not involve changes to local state, to avoid the overhead of two-step processing by directly invoking **getMarkup**. The URL MAY specify the `wsrp-navigationalState` and/or `wsrp-navigationalValues` portlet URL parameters, whose values the Consumer MUST supply in the `opaqueValue` and `publicValues` fields of the `NavigationalContext` structure, respectively.

10.2.1.1.3 wsrp-urlType = resource

Activation of the URL will result in the Consumer acting as a gateway to the underlying resource, possibly in a cached manner, and returning it to the user-agent. The URL for the resource (including any query string parameters) is encoded as the value of the `wsrp-url` parameter. When a portlet URL specifies "resource" for the `wsrp-urlType` portlet URL parameter, either the `wsrp-resourceID` portlet URL parameter or a combination of the `wsrp-requiresRewrite` and the `wsrp-url` portlet URL parameters MUST also be specified. Since the Portlet is allowed to specify either both or just one of the means of getting a resource, Consumers will need to support both the transport proxy and SOAP operation mechanisms. This flexibility allows the Portlet's markup to control which resource serving mechanism is in use whenever it needs to and allows for the Portlet to provide additional guidance when both mechanisms are supported, since the fidelity of what is presented to the End-User might depend on which mechanism is selected by the Consumer.

10.2.1.1.3.1 wsrp-url

This parameter provides the actual URL to the resource. Note that this needs to be an absolute URL as the resource fetch will have no base for use in fetching a relative URL. Also note that since this resource URL will appear as a parameter value, it has to be strictly encoded (i.e. "&", "=", "/", and "?" need to be url-escaped) so that special URL characters do not invalidate the processing of the enclosing URL. Consumers are strongly encouraged to use the same communication style (e.g. HTTP Get or Post) for retrieving the resource as was used in requesting the resource by the user-agent. Producers can set cookies with any needed data which the Consumer will then process, using the cookie rules established by RFC2109[19] , relative to forwarding these when retrieving the resource using HTTP transport.

10.2.1.1.3.2 wsrp-resourceID

This parameter provides the resourceID parameter which the Consumer MUST supply when invoking the **getResource** operation. The presence of this parameter informs the Consumer that the **getResource** operation is a viable means of fetching the resource requested by the Portlet's markup.

10.2.1.1.3.3 wsrp-preferOperation

When this optional parameter (default value is "`false`") has a value of "`true`", the Portlet is indicating a preference for the Consumer to use the **getResource** operation to fetch the resource. If the resource URL specifies both the `wsrp-url` and the `wsrp-resourceID` parameters, the Consumer can use either the http proxy technique introduced in WSRP v1.0 or the **getResource** operation, but is encouraged to follow the guidance provided by this url parameter.

10.2.1.1.3.4 wsrp-resourceState

The value of this portlet URL parameter defines the state which the Consumer MUST send in the `resourceState` field of the `ResourceParams` structure when the URL is activated. If this parameter is missing, the Consumer MUST NOT supply a value in the `resourceState` field of the `ResourceParams` structure.

10.2.1.1.3.5 wsrp-requiresRewrite

This boolean informs the Consumer that the resource needs to be parsed for URL rewriting. Normally this means that there are names that will be cross-referenced between the markup and this resource (e.g. JavaScript references). Note that this means the Consumer needs to deal with rewriting unique "namespaced" names in a set of documents, rather than treating each document individually. Processing such resources in a manner that allows caching of the resulting resource by the End-User's user-agent can improve the performance of the aggregated page for the End-User. In particular, Consumers can process namespace rewriting by using a prefix that is unique to the user/Portlet pair provided any such prefix is held constant for the duration of use within the user's session with the Consumer of any one Portlet.

10.2.1.1.4 Other wsrp-urlType values

WSRP extensions can define additional values for the `wsrp-urlType` portlet URL parameter. Extensions doing so are encouraged to define extensions to the `Templates` structure to add templates for both normal and secure URI access using the semantics defined by the extension. Consumers supporting an extension MUST supply defined

extensions to the `Templates` structure to those Producers indicating support for the same extension in their `ServiceDescription`.

10.2.1.2 wsrp-navigationalState

The value of this portlet URL parameter defines the `opaqueValue` portion of the Portlet's navigational state the Consumer MUST send to the Producer when the URL is activated. If this parameter is missing, the Consumer MUST NOT supply the `opaqueValue` portion of the Portlet's navigational state.

10.2.1.3 wsrp-navigationalValues

The value of this portlet URL parameter defines updates to the `publicValues` portion of the Portlet's navigational state which the Portlet defined in its `PortletDescription`. As updates, a Portlet-defined `navigationalParameter` without a value on a particular URL maintains the current value the Consumer has for it while a `navigationalParameter` without a value defines the clearing of that parameter. Since multiple updates can be specified on a single URL, the following sequence of steps MUST be used when encoding the value for the `wsrp-navigationalValues` portlet url parameter:

1. A querystring-like [20] value is built from the parameter identifiers and values to be set. Parameters that are an array of strings can be specified by repeating the parameter. For example, if a Portlet has defined three parameters and is setting two of these parameters, one an array of strings (parameter name is "{http://www.example.com}param1" with an identifier of "p1"), and clearing another (parameter name is "{http://www.example.com}param2" with an identifier of "p2") while leaving the third parameter (parameter name is "{http://www.example.com}param3" with an identifier of "p3") unchanged, results in;
   p1=value1&p1=value2&p2=
2. The string built in step #2 is url-encoded so that it can appear as the value of the portlet url parameter. The example becomes;
   p1%3Dvalue1%26p1%3Dvalue2%26p2%3D
3. The url-encoded string is then used as the value of the `wsrp-navigationalValues` portlet url parameter.

If values are supplied for the `wsrp-navigationalValues` portlet URL parameter, the Consumer MUST supply those values in the `publicValues` field along with the values the Consumer has for the Portlet's `navigationalParameters` which were not referenced by the `wsrp-navigationalValues` portlet URL parameter.

10.2.1.4 wsrp-interactionState

The value of this portlet URL parameter defines the interaction state the Consumer MUST send to the Producer when the URL is activated. If this parameter is missing, the Consumer MUST NOT supply the `interactionState` field of the `InteractionParams` structure.

10.2.1.5 wsrp-mode

Activating this URL includes a request to change the `mode` parameter in `MarkupParams` into the mode specified as the value for this portlet URL parameter. The value for `wsrp-mode` MUST be one of the modes detailed in **[Section 6.9]** or a custom mode the Consumer specified as supported during registration. The `wsrp-mode` portlet URL parameter MAY be used whenever the `wsrp-urlType` portlet URL parameter has a value of "`blockingAction`" or "`render`".

10.2.1.6 wsrp-windowState

Activating this URL includes a request to change the `windowState` parameter in `MarkupParams` into the window state specified as the value for this portlet URL parameter. The value for `wsrp-windowState` MUST be one of the values detailed in **[Section 6.10]** or a custom window state the Consumer specified as supported during registration. The `wsrp-windowState` portlet URL parameter MAY be used whenever the `wsrp-urlType` portlet URL parameter has a value of "`blockingAction`" or "`render`".

10.2.1.7 wsrp-fragmentID

This portlet URL parameter specifies the portion of an URL that navigates to a place within a document.

10.2.1.8 wsrp-secureURL

The value for the `wsrp-secureURL` is a boolean indicating whether the resulting URL MUST involve secure communication between the client and Consumer, as well as between the Consumer and Producer. The default value of this boolean is "`false`". Note that the Consumer's aggregated page MUST be secure if any of the Portlets whose content is being displayed on the page have indicated the need for secure communication for their current markup [A612] [R415],

10.2.1.9 wsrp-extensions

Extensions to the data supplied on the URL which this protocol defines MUST use the following sequence to encode the additoinal information into the `wsrp-extensions` portlet URL parameter:

1. A querystring-like [20] value is built from the parameter identifiers and values to be set. Parameters that are an array of strings can be specified by repeating the parameter. For example, if a Portlet has two additional parameters, one is an array of strings (parameter name is "param1") and and the other a simple string (parameter name is "param2"), results in;
   "param1=value1&param1=value2&param2=value3
2. The string built in step #1 is url-encoded so that it can appear as a querystring value or part of the path portion of the URL. The example becomes;
   param1%3Dvalue1%26param1%3Dvalue2%26param2%3Dvalue3
3. The url-encoded string is then used as the value of the `wsrp-extensions` portlet url parameter or to replace it within a template.

10.2.1.10 URL examples

Here are some examples of portlet URLs following this format:

- Load a resource: http://resource.example.com/images/test.gif:

```
wsrp_rewrite?wsrp-urlType=resource&amp;wsrp-url=http%3A%2F%2Fresource.example.com%
2Fimages %2Ftest.gif&amp;wsrp-requiresRewrite=true&amp;wsrp-resourceID=images%2Ftest.gif/
wsrp_rewrite
```

- Declare a secure interaction back to the Portlet:

```
wsrp_rewrite?wsrp-urlType=blockingAction&amp;wsrp-secureURL=true&amp; wsrp-
navigationalState=a8h4K5JD9&amp;wsrp-interactionState=fg4h923mdk/wsrp_rewrite
```

- Request the Consumer render the Portlet in a different mode and window state:

```
wsrp_rewrite?wsrp-urlType=render&amp;wsrp-mode=help&amp;wsrp-windowState=maximized/
wsrp_rewrite
```

10.2.2 Producer URL Writing

To enable Producer URL writing, several templates are defined by which the Consumer indicates how it needs portlet URLs formatted in order to process them properly. These all take the form of a simple template, for example:

```
http://Consumer.example.com/path/{wsrp-urlType}?mode={wsrp-mode}
&amp;...
```

The definition of the content of this template is completely up to the Consumer. It may consist of zero or more replacement tokens. These tokens are enclosed in curly braces (i.e. "{" and "}") and contain the name of the portlet URL parameter which the Producer MUST replace (using "" for those parameters where the Producer has no value). This replacement includes the curly braces, namely; "{`wsrp-urlType`}" becomes "`render`" when that is the desired URL type. All content outside the {} pairs and all {} pairs containing tokens the Producer/Portlet does not recognize MUST be treated by the Producer/Portlet as constants the Consumer needs to receive when the portlet URL is activated. The list of defined portlet URL parameter names matches those in **[Section 10.2.1]** with the addition of those specified in **[Section 10.2.2.9]**.

Portlets dynamically computing portlet URLs in the user-agent environment (e.g. in JavaScript) need to take into account the distributed nature of preparing and processing portlet URLs. The Consumer's template might not directly produce a valid URL, but may be such that the Consumer further processes the portlet URL for its own purposes (e. g. adds data it needs when the portlet URL is activated which has not been supplied to the Producer). As a result, the only safe means to accommodate this computation of dynamic URLs is to store the template(s) whole in the markup and then use this template directly when computing the dynamic portlet URL.

This specification defines a `Templates` structure that supplies values for the set of defined templates. The Consumer supplies these templates for Portlets specifying `doesUrlTemplateProcessing` as "`true`". Portlets also specifying `templatesStoredInSession` as "`true`" enable the Consumer to only send these until the Producer returns a `sessionID`. The following describe in more detail the usage of the fields from the `Templates` structure.

10.2.2.1 blockingActionTemplate

Activation of the URL will result in an invocation of **performBlockingInteraction**. The Consumer MUST integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState, wsrp-navigationalValues, wsrp-interactionState, wsrp-mode` and `wsrp-windowState` in its template and SHOULD integrate placeholders for the other portlet URL parameters defined in this specification.

10.2.2.2 secureBlockingActionTemplate

secureBlockingActionTemplate is equivalent to blockingActionTemplate, but using secure communication.

10.2.2.3 renderTemplate

Activation of the URL will result in an invocation of **getMarkup**. The Consumer MUST integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState, wsrp-navigationalValues, wsrp-mode` and `wsrp-windowState` in its template.

10.2.2.4 secureRenderTemplate

`secureRenderTemplate` is equivalent to `renderTemplate,` but using secure communication.

10.2.2.5 resourceTemplate

Activation of the URL will result in the Consumer fetching the underlying resource, possibly in a cached manner, and returning it to the End-User. The Consumer MUST integrate placeholders for at least the portlet URL parameters `wsrp-url, wsrp-resourceID, wsrp-preferOperation, wsrp-requiresRewrite` and `wsrp-resourceState` to allow the Portlet to place all the pieces of information it could use for accessing a resource. If the Portlet needs to share data with a resource when the `wsrp-preferOperation` has a value of "`false`", it can either supply the data on the URL to the resource or exploit the cookie support when the transport layer is http.

10.2.2.6 secureResourceTemplate

`ResourceTemplate` is equivalent to `ResourceTemplate,` but using secure communication.

10.2.2.7 defaultTemplate

This is the template whose value is to be used as the default value for any non-secure template whose value is not supplied. Consumers not supplying all the other non-secure templates MUST set a value for this template. Since this may become the value for action and resource oriented templates, the Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState, wsrp-navigationalValues, wsrp-interactionState, wsrp-resourceID, wsrp-resourceState, wsrp-mode` and `wsrp-windowState` in this template.

10.2.2.8 secureDefaultTemplate

This is the template whose value is to be used as the default value for any secure template (i.e. those with names beginning with "secure") whose value is not supplied. Consumers not supplying all the other secure templates MUST set a value for this template. Since this may become the value for action and resource oriented templates, the Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState, wsrp-navigationalValues, wsrp-interactionState, wsrp-resourceID, wsrp-resourceState,` and `wsrp-windowState` in this template.

10.2.2.9 Portlet URL parameters

The following portlet URL parameters are defined for the purpose of enabling a Consumer's templates to be generic to a Producer. If the Consumer includes `wsrp-portletHandle, wsrp-userContextKey, wsrp-portletInstanceKey` or `wsrp-sessionID` in a template, Producer written URLs based on that template MUST replace the specified portlet URL parameter with the value the Consumer separately supplied in a data field.

Portlet URL parameter	Structure name	Field name
wsrp-portletHandle	PortletContext	portletHandle
wsrp-userContextKey	UserContext	userContextKey
wsrp-portletInstanceKey	RuntimeContext	portletInstanceKey
wsrp-sessionID	SessionParams	sessionID

10.2.3 Extended BNF Description of URL formats

These definitions utilize the syntax defined in ISO/IEC 14977 [21]

ConsumerURL = BeginToken (RenderURL | ActionURL | ResourceURL | ExtensionURL) EndToken

BeginToken = "wsrp_rewrite?"

EndToken = "/wsrp_rewrite"

RenderURL = "wsrp-urlType=render" {("&" | "&") (CommonPair | RenderPair)}

ActionURL = "wsrp-urlType=blockingAction" {("&" | "&") (CommonPair | ActionPair)}

ResourceURL = "wsrp-urlType=resource" {("&" | "&") (CommonPair | ResourcePair)}

WSRPURLTypes = "render" | "blockingAction" | "resource"

ExtensionURL = "wsrp-urlType=" (Text - WSRPURLTypes) {("&" | "&") (ExtensionPair)}

CommonPair = CommonTextPair | CommonBooleanPair

CommonTextPair = CommonTextName "=" Text

CommonTextName = "wsrp-fragmentID" | "wsrp-extensions"

CommonBooleanPair = CommonBooleanName "=" BooleanValue

CommonBooleanName = "wsrp-secureURL"

RenderPair = RenderTextName "=" Text

RenderTextName = "wsrp-mode" | "wsrp-windowState" | "wsrp-navigationalState" | "wsrp-navigationalValues"

ActionPair = ActionTextName "=" Text

ActionTextName = "wsrp-interactionState" | RenderTextName

ResourcePair = (ResourceTextName "=" Text) | (ResourceBooleanName "=" BooleanValue)

ResourceTextName = "wsrp-url" | "wsrp-resourceID" | "wsrp-resourceState"

ResourceBooleanName = "wsrp-requiresRewrite" | "wsrp-preferOperation"

ExtensionPair = Text "=" Text

Text = { "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")" | "%" }

BooleanValue = (("true" | "1") | ("false" | "0"))

ProducerURLTemplate = {{Text} {ReplacementToken}}

ReplacementToken = "{" CommonTextName | RenderTextName | ActionTextName | ResourceTextName | CommonBooleanName | ResourceBooleanName | ParameterName "}"

ParameterName = "wsrp-urlType" | "wsrp-portletHandle" | "wsrp-userContextKey" | "wsrp-portletInstanceKey" | "wsrp-sessionID"

10.2.4 Method=get in HTML forms

User-Agents often throw away any query string from the URL indicated with the form's action attribute when generating the URL they will activate when the form's method is "get". This is the simplest means for them to generate a valid query string. The difficulty this causes is that Consumer's often prefer to store the information they will use when a portlet URL is activated as query string parameters. As a result, Portlets that include HTML forms with method=get in their markup MUST specify `usesMethodGet` as "`true`" in their `PortletDescription`. Consumers choosing to use such Portlets need to format their portlet URLs such that portlet URL activations are processed correctly, regardless of whether Consumer or Producer URL writing is in use. Note that the discussion in **[Section 12.2]** will apply to some uses of method=get in HTML forms.

10.3 Namespace Encoding

Aggregating multiple Portlets from different sources can potentially result in naming conflicts for various types of elements: named attributes, JavaScript functions and variables [A303], etc. Tokens needing uniqueness on the aggregated page MUST be encoded to a Portlet-instance specific namespace [A301]. The Portlet MAY do this by prefixing the name of the item with the `namespacePrefix` from the `RuntimeContext` structure. We note that the JavaScript examples exclude the possibility of starting such prefixes with a numerical character.

In order for the semantics of namespace encoding, whether by Consumer rewriting or Producer writing, to serve the purposes of making the encoded items unique on the aggregated page and processable by the Producer/Portlet, should they become part of a user interaction, the `namespacePrefix` from the `RuntimeContext` structure MUST be the value used for both Consumer rewriting and Producer writing and MUST be constant for the duration of the `portletHandle`. Portlets choosing to apply namespace encoding to items that become part of a user interaction MUST process these items with their namespace prefix attached as Consumers MUST NOT remove such prefixes,

regardless of whether Consumer rewriting or Producer writing was used to attach the prefix. The Portlet does not need to store the value of the namespace prefix as the required semantics result in the `namespacePrefix` from the `RuntimeContext` structure supplying it to the Portlet again. Note that many things Producers and Portlets could namespace, such as CSS style names, are not required to be unique and therefore are preferably not namespaced as this adds processing burden to generate the namespacing. This likely also has an impact on client side code that may use such items to locate items on the page.

Similar to the case of URL rewriting, two options exist to obtain a namespace prefix.

10.3.1 Consumer Rewriting

The Portlet can prefix the token with "wsrp_rewrite_". The Consumer will locate such markers and MUST replace them with the same value that was supplied to the Portlet in the `namespacePrefix` field of the `RuntimeContext` structure. This prefix has been chosen such that the Consumer is able to do a single parse of the markup to both locate such markers and the URL rewrite expressions described in **[Section 10.2.1]**. In addition, this prefix is legal for at least the JavaScript and VBScript scripting languages and CSS class names. This permits the independent testing of most generated markup fragments.

10.3.2 Producer Writing

The Portlet uses the `namespacePrefix` provided by the Consumer in the `RuntimeContext` structure to prefix these tokens in its markup.

10.4 Markup Fragment Rules

Because the Consumer aggregates the markup fragments produced by Portlets into a single page, some rules and limitations are needed to ensure the coherence of the resulting page to be displayed to the End-User. For efficiency reasons, Consumers are not required to validate the markup fragments returned by the Portlet. So in order to be aggregated, the Portlet's markup needs to conforms to the following general guidelines [A300] [A302].

The disallowed tags listed below are those tags that impact other Portlets or may even break the entire aggregated page. Inclusion of such a tag invalidates the whole markup fragment, which the Consumer MAY replace with an error message.

10.4.1 HTML

10.4.1.1 Disallowed Tags

Since the Consumer may implement its aggregation in many ways, including using frames, some Consumers may actually support these disallowed tags. However, in order to be a conforming Portlet, a Portlet MUST NOT use the tags <body>, <frame>, <frameset>, <head>, <html>, and <title>.

10.4.1.2 Other Tags

There are some tags that are specifically prohibited by the HTML specification from occurring outside the <head> of the document. However, user-agent implementations offer varying levels of support. For example, current versions of Internet Explorer and Firefox both support the style tag anywhere within the document. It is up to the Portlet

developer to decide when using such tags is appropriate. Tags fitting this description include <base>, <link>, <meta>, and <style>.

Since HTML excludes the nesting for <form> tags and Consumers could be embedding a Portlet's markup with a form, special care needs to be taken by both the Portlet and Consumer.

10.4.2 XHTML

10.4.2.1 Disallowed Tags

Since the Consumer may implement its aggregation in many ways, including using frames, some Consumers may actually support these disallowed tags. However, in order to be a conforming Portlet, a Portlet MUST NOT use the tags <body>, <head>, <html>, and <title>.

10.4.2.2 Other Tags

There are some tags that are specifically prohibited by the XHTML specification from occurring outside the <head> of the document. However, user-agent implementations offer varying levels of support. For example, current versions of Internet Explorer and Firefox both support the style tag anywhere within the document. It is up to the Portlet developer to decide when using such tags is appropriate. Tags fitting this description include <base>, <link>, <meta>, and <style>.

Since XHTML excludes the nesting for <form> tags and Consumers could be embedding a Portlet's markup with a form, special care needs to be taken by both the Portlet and Consumer.

10.4.3 XHTML Basic

10.4.3.1 Disallowed Tags

Since the Consumer may implement its aggregation in many ways, including using frames, some Consumers may actually support these disallowed tags. However, in order to be a conforming Portlet, a Portlet MUST NOT use the tags <body>, <head>, <html>, and <title>.

10.4.3.2 Other Tags

There are some tags that are specifically prohibited by the XHTML Basic specification from occurring outside the <head> of the document. However, user-agent implementations offer varying levels of support. For example, current versions of Internet Explorer and Firefox both support the style tag anywhere within the document. It is up to the Portlet developer to decide when using such tags is appropriate. Tags fitting this description include <base>, <link>, <meta>, and <style>.

Since XHTML Basic excludes the nesting for <form> tags and Consumers could be embedding a Portlet's markup with a form, special care needs to be taken by both the Portlet and Consumer.

10.5 CSS Style Definitions

One of the goals of an aggregated page is a common look-and-feel across the Portlets contained on that page

[A500]. This not only affects the decorations around the Portlets, but also their content. Using a common CSS style sheet for all Portlets, and defining a set of standard styles, provides this common look-and-feel without requiring the Portlets to generate Consumer-specific markup. Portlets SHOULD use the CSS style definitions from this specification in order to participate in a uniform display of their content by various Consumers. For markup types that support CSS stylesheets, Consumers MUST supply a CSS stylesheet to the End-User's agent with definitions for the classes defined in **[Section 10.5]** of this specification.

This section defines styles for a variety of logical units in the markup.

10.5.1 Links (Anchor)

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

10.5.2 Fonts

The font style definitions affect the font attributes only (i.e. 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 a Portlet author wants a certain font type to be larger or smaller, they should indicate this using a relative size.

Example1: Important information

Example2: Small and dim

10.5.3 Messages

Message style definitions affect the rendering of a paragraph (i.e. 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

10.5.4 Sections

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

Style	Description

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

10.5.5 Tables

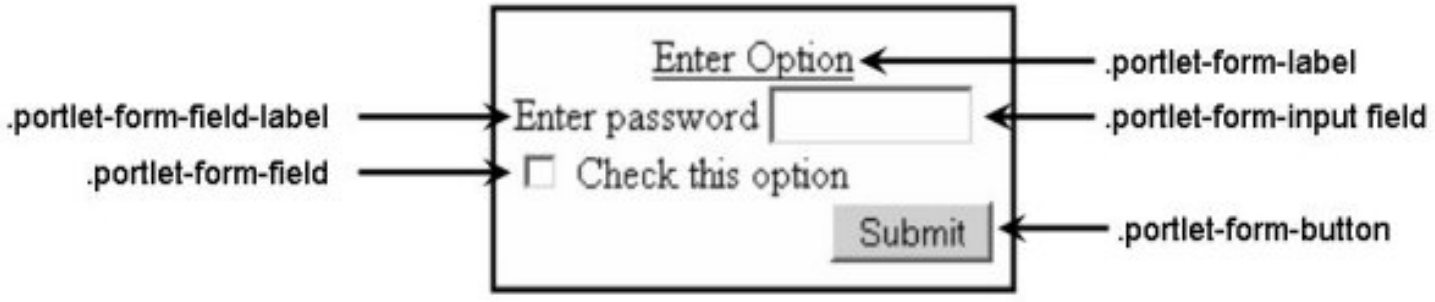
Table style definitions affect the rendering (i.e. alignment, borders, background color, etc.) as well as their text attributes.

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).

10.5.6 Forms

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

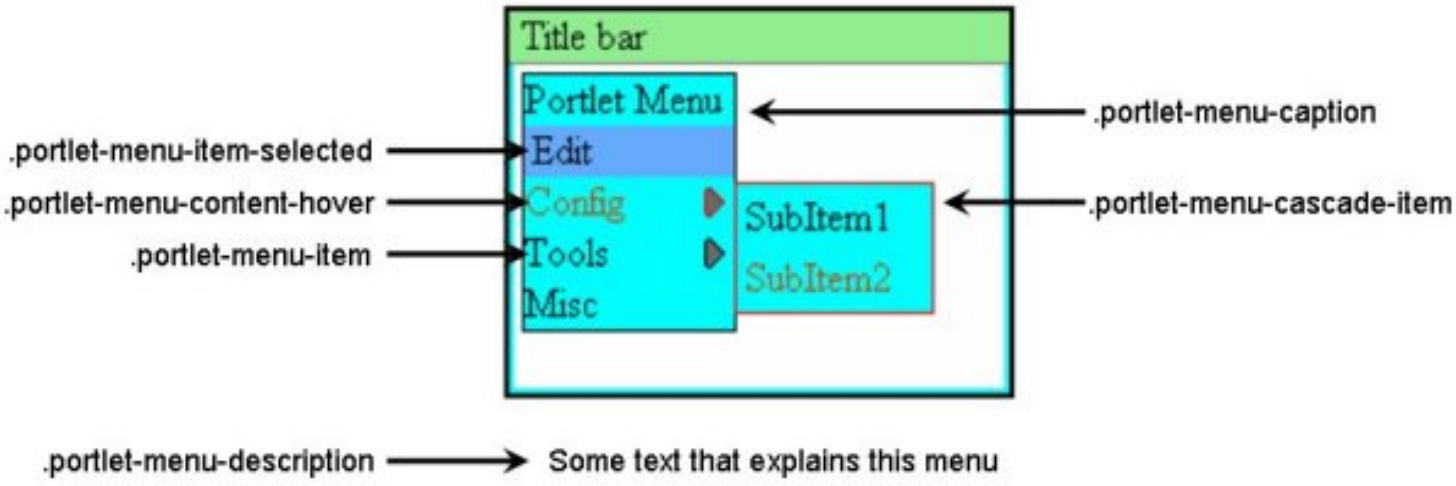
Style	**Description**
`portlet-form-label`	Text used for the descriptive label of the whole form (not the labels for fields
`portlet-form-input-field`	Text of the user-input in an input field
`portlet-form-button`	Text on a button
`portlet-icon-label`	Text that appears beside a context dependent action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



10.5.7 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 user's pointer (typically a mouse) hovers over it
`portlet-menu-item-hover-selected`	Selected menu item when the user's pointer hovers over it
`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 user's pointer hovers over it
`portlet-menu-cascade-item-hover-selected`	Selected sub-menu item when the user's pointer 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 user's pointer hovers over it
`portlet-menu-content-hover-selected`	Content for a selected menu or sub-menu item when the user's pointer 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 user's pointer hover over it
`portlet-menu-indicator-hover-selected`	Indicator when the associated menu item is selected and has the user's pointer 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



11 User Information

This specification provides a mechanism for Portlets to use End-User information as a means for personalizing behavior to the current user [A600] [A606]. A standard set of user attributes has been derived from P3P User Data. Extensibility is supported in both directions; the Consumer indicates to the Producer during registration what set of user profile extensions it supports, and a Portlet's metadata declares what user profile items it uses (including any extended user profile items). The following table maps the nested profile structures to `userProfileItems`:

Profile Name	Structure 1	Structure 2	Structure 3	Field Name
name/prefix	PersonName			prefix
name/given	PersonName			given
name/family	PersonName			family
name/middle	PersonName			middle
name/suffix	PersonName			suffix
name/nickname	PersonName			nickname
bdate				bdate
gender				gender
employerInfo/employer	EmployerInfo			employer
employerInfo/department	EmployerInfo			employerInfo/jobtitle
EmployerInfo	jobtitle			jobtitle
homeInfo/postal/name	Contact	Postal		name
homeInfo/postal/street	Contact	Postal		street
homeInfo/postal/city	Contact	Postal		city
homeInfo/postal/stateprov	Contact	Postal		stateprov
homeInfo/postal/postalcode	Contact	Postal		postalcode
homeInfo/postal/country	Contact	Postal		country
homeInfo/postal/organization	Contact	Postal		organization
homeInfo/telecom/telephone/intcode	Contact	Telecom	TelephoneNum	intcode
homeInfo/telecom/telephone/loccode	Contact	Telecom	TelephoneNum	loccode
homeInfo/telecom/telephone/number	Contact	Telecom	TelephoneNum	number

homeInfo/telecom/telephone/ext	Contact	Telecom	TelephoneNum	ext
homeInfo/telecom/telephone/comment	Contact	Telecom	TelephoneNum	comment
homeInfo/telecom/fax/intcode	Contact	Telecom	TelephoneNum	intcode
homeInfo/telecom/fax/loccode	Contact	Telecom	TelephoneNum	loccode
homeInfo/telecom/fax/number	Contact	Telecom	TelephoneNum	number
homeInfo/telecom/fax/ext	Contact	Telecom	TelephoneNum	ext
homeInfo/telecom/fax/comment	Contact	Telecom	TelephoneNum	comment
homeInfo/telecom/mobile/intcode	Contact	Telecom	TelephoneNum	intcode
homeInfo/telecom/mobile/loccode	Contact	Telecom	TelephoneNum	loccode
homeInfo/telecom/mobile/number	Contact	Telecom	TelephoneNum	number
homeInfo/telecom/mobile/ext	Contact	Telecom	TelephoneNum	ext
homeInfo/telecom/mobile/comment	Contact	Telecom	TelephoneNum	comment
homeInfo/telecom/pager/intcode	Contact	Telecom	TelephoneNum	intcode
homeInfo/telecom/pager/loccode	Contact	Telecom	TelephoneNum	loccode
homeInfo/telecom/pager/number	Contact	Telecom	TelephoneNum	number
homeInfo/telecom/pager/ext	Contact	Telecom	TelephoneNum	ext
homeInfo/telecom/pager/comment	Contact	Telecom	TelephoneNum	comment
homeInfo/online/email	Contact	Online		email
homeInfo/online/uri	Contact	Online		uri
businessInfo/postal/name	Contact	Postal		name
businessInfo/postal/street	Contact	Postal		street
businessInfo/postal/city	Contact	Postal		city
businessInfo/postal/stateprov	Contact	Postal		stateprov
businessInfo/postal/postalcode	Contact	Postal		postalcode
businessInfo/postal/country	Contact	Postal		country
businessInfo/postal/organization	Contact	Postal		organization
businessInfo/telecom/telephone/intcode	Contact	Telecom	TelephoneNum	intcode
businessInfo/telecom/telephone/loccode	Contact	Telecom	TelephoneNum	loccode
businessInfo/telecom/telephone/number	Contact	Telecom	TelephoneNum	number
businessInfo/telecom/telephone/ext	Contact	Telecom	TelephoneNum	ext
businessInfo/telecom/telephone/comment	Contact	Telecom	TelephoneNum	comment
businessInfo/telecom/fax/intcode	Contact	Telecom	TelephoneNum	intcode
businessInfo/telecom/fax/loccode	Contact	Telecom	TelephoneNum	loccode
businessInfo/telecom/fax/number	Contact	Telecom	TelephoneNum	number
businessInfo/telecom/fax/ext	Contact	Telecom	TelephoneNum	ext
businessInfo/telecom/fax/comment	Contact	Telecom	TelephoneNum	comment
businessInfo/telecom/mobile/intcode	Contact	Telecom	TelephoneNum	intcode
businessInfo/telecom/mobile/loccode	Contact	Telecom	TelephoneNum	loccode
businessInfo/telecom/mobile/number	Contact	Telecom	TelephoneNum	number
businessInfo/telecom/mobile/ext	Contact	Telecom	TelephoneNum	ext
businessInfo/telecom/mobile/comment	Contact	Telecom	TelephoneNum	comment
businessInfo/telecom/pager/intcode	Contact	Telecom	TelephoneNum	intcode
businessInfo/telecom/pager/loccode	Contact	Telecom	TelephoneNum	loccode
businessInfo/telecom/pager/number	Contact	Telecom	TelephoneNum	number
businessInfo/telecom/pager/ext	Contact	Telecom	TelephoneNum	ext

businessInfo/telecom/pager/comment	Contact	Telecom	TelephoneNum	comment
businessInfo/online/email	Contact	Online		email
businessInfo/online/uri	Contact	Online		uri

Portlets that need access to user information declares in its metadata the specific user profile fields it needs using the names specified above.

Consumers supplying additional custom profile fields are encouraged to publish a similar mapping between `userProfileItems` and the custom fields.

11.1 Passing User Information

User information can be supplied to the Producer when a Consumer invokes certain operations. A Consumer SHOULD provide the specific fields the Portlet declared it needs, unless the information is not available or is restricted by policy (e.g. privacy policy).

11.2 User Identity

Mechanisms that support federation of user identity between web service systems are defined in other specifications, such as WS-Security and SAML. If a Consumer and Producer need to share a common identity for an End-User, it is recommended that compliance with these standards be the means to passing the required information [A602].

It is anticipated that some Portlets will interact with one or more back-end applications that require a user identity for the End-User. If the user identity required by the back-end application is not the same as that authenticated or otherwise supplied by the Consumer, the Portlet can request the End-User to provide the necessary information (preferably using secure transport) for use with the back-end application via markup interactions (e.g. display a form that prompts for a user identity and any security tokens (such as a password) for the back-end system) [A603].

---

12 Well Known Extensions

For the purpose of interoperability, the following extension items are defined. Each definition includes the types it is allowed to extend.

12.1 wsrp-extra:doctype

This extension is of type `QNamedString` from the `urn:oasis:names:tc:wsrp:extra:v1` namespace with a name of "wsrp-extra:doctype" with the value carrying the doctype URI. The URIs to use as doctypes have been defined elsewhere (e.g. http://www.w3.org/TR/html4/struct/global.html#h-7.2). This extension can apply to both the MarkupParams (specifying an allowed doctype to the Portlet) and the MarkupContext (specifying the actual doctype of the markup being returned) types.

12.2 wsrp-extra:extendedURLParameters

This extension applies to `MarkupParams`, `InteractionParams` and `ResourceParams` and provides a standardized means for carrying name/value pairs from an activated URI which were not either consumed by the Consumer's processing or reflected elsewhere within the protocol. The type of this extension is `wsrp:NamedStringArray`. Consumers should take care with regard to how user-agents encode this data. In particular, common user-agents (e. g. web browsers) encode the data in the character set of the page submitting the URI. As the Producer is ignorant of this encoding and the Consumer is required to consistently encode parameters passed to the Producer in the SOAP message, Consumers MUST ensure the data is properly decoded before it is passed to the Producer.

Consumers MUST send the value received for the `wsrp-urlType` portlet URL parameter using the name "`wsrp-urlType`" as the first element in the array of name/value pairs. Consumers MUST NOT resend a particular set of `wsrp-extra:extendedURLParameters` on later invocations of WSRP defined operations. Portlets needing any of the supplied data in order to properly rerender their markup are responsible for storing that data between requests.

Examples of intended use for this extension include:

- Extensions which define new values for the portlet URL parameter `wsrp-urlType` are likely to also define extended data value to be carried in the `wsrp-extensions` portlet URL parameter. These values would be forwarded to the Portlet using this definition if the extension is using the operations defined by this specification.
- For those Consumers indicating support for forms with method=GET, this extension is used if the `wsrp-urlType` on the submission URL is "`render`".

---

13 Constants

Type	Value	Description
Mode	wsrp:view	Portlet is expected to render markup reflecting its current state.
Mode	wsrp:edit	Portlet is expected to render markup useful for End-User customization.
Mode	wsrp:help	Portlet is expected to render markup useful for helping an End-User understand the Portlet's operation.
Mode	wsrp:preview	Portlet is expected to render markup representative of its configuration, as this might be useful to someone testing a page layout.
Window state	wsrp:normal	The Portlet is sharing space with other Portlets and should restrict its consumption of space accordingly.
Window state	wsrp:minimized	The Portlet, though still aggregated on the page, is expected to restrict its consumption of space to a bare minimum.
Window state	wsrp:maximized	The Portlet is being offered significantly more that the normal share of the space available to Portlets on the Consumer's aggregated page.
Window state	wsrp:solo	The Portlet is the only Portlet being rendered on the Consumer's aggregated page.
Events	wsrp:eventHandlingFailed	This event is used to signal that the Consumer has detected that event processing encountered an error.

Events	wsrp:newNavigationalContextScope	This event is used to signal that Consumer has determined that a new scope is starting for the Portlet's `navigationalContext`. This enables those Portlets making technology choices which require either the Producer or Portlet to store/manage a portion of the overall state related to navigation to manage that portion in a manner consistent with the Consumer's management of this state for other Portlets.

14 Fault Messages

In addition to generic fault messages that may be generated by the web service stacks of the Consumer and/or Producer, a variety of messages specific to this protocol are defined. The following WSRP error codes are defined within the same namespace as the rest of the types defined by this specification. The SOAP 1.1 faultcode SHOULD be set to the WSRP error code being raised, namespace qualified to be in the "`urn:oasis:names:tc:wsrp:v1:types`" namespace. In addition, the SOAP 1.1 fault's detail element MUST contain the corresponding namespaced fault element from the WSRP v1 WSDL as its only content. When using SOAP 1.2, the Subcode element MUST be present and carry the corresponding WSRP error code in its mandatory Value sub-element. The Subcode element MUST be contained in a SOAP 1.2 Receiver fault code, for the fault messages defined in this specification. SOAP 1.2 faults MAY carry additional content in the Detail sub-element, but MUST carry the corresponding WSRP namespaced fault element.

Fault Code	Description
`AccessDenied`	Policy has denied access. This may be related to the Consumer's registration
`ExportByValueNotSupported`	The Consumer required exportByValue, but the Producer does not support this feature
`ExportNoLongerValid`	The export references items which are not on the Producer
`InconsistentParameters`	Used when a Consumer supplies inconsistent parameters (e.g. when a `portletHandle` is not scoped by the supplied `registrationHandle`).
`InvalidRegistration`	Used when a Consumer supplies a `registrationHandle/registrationState` pair that is not recognized by the Producer. This includes when no `registrationHandle` is supplied, but a registration is required.
`InvalidCookie`	Used only when the environment at the Producer has timed out AND the Producer needs the Consumer to invoke **initCookie** again and resend data that may have been stored in sessions related to a cookie.
`InvalidHandle`	Used when the Consumer supplies an invalid `Handle`
`InvalidSession`	Used only when a Producer session has timed out and the Producer needs the Consumer reinvoke the operation resending user profile and URL templates that may have been stored in the Producer session. If both userContextStoredInSession and templatesStoredInSession fields of the PortletDescription type are "`false`", Producers are encouraged to reinitialize the session in stead of returning the InvalidSession fault.
`InvalidUserCategory`	The specified `userCategory` is not supported
`ModifyRegistrationRequired`	Used when a modification to the information supplied during registration is required in order to continue using the registration
`MissingParameters`	Used when required parameters are missing

`OperationFailed`	Normal execution of the operation failed. Check the detailed message for reasons why.
`OperationNotSupported`	The Producer does not support the optional feature enable by the operation
`ResourceSuspended`	The indicated resource is no longer available for normal use, but is available for renewal
`PortletStateChangeRequired`	Used when a Portlet needs to modify its enduring state, but has been prevented from doing so.
`UnsupportedLocale`	The Portlet does not support generating markup for the requested locale. Since the Portlet is not required to generate markup in the requested locale, a Portlet returning this fault message is indicating that it processes locales in a stricter manner and has no markup for the requested locales. The Consumer can treat this as a specialization of the OperationFailed fault message and does not have to retry getting the markup in other locales.
`UnsupportedMimeType`	The Portlet does not support generating markup for the requested mime type
`UnsupportedMode`	The Portlet does not support generating markup for the requested mode
`UnsupportedWindowState`	The Portlet does not support generating markup for the requested window state

15 WSDL Interface Definition

Normative copies of the WSDL that MUST be referenced by Producers implementing this specification can be found at:

http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-types.xsd - The type, message and fault definitions for this specification. These definitions form the "`urn:oasis:names:tc:wsrp:v2:types`" namespace.

http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-interfaces.wsdl - The portType definitions for this specification. These definitions form the "`urn:oasis:names:tc:wsrp:v2:intf`" namespace.

http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-bindings.wsdl - The standard binding definitions for this specification. These definitions form the "`urn:oasis:names:tc:wsrp:v2:bind`" namespace.

While this version of the WSRP specification does not define bindings for serializing some portions of the messages using the **[MTOM]** specification, it is expected these will be defined as an errata item once web stack support allows for interoperability testing using the definitions.

In addition the TC has provided definitions in http://docs.oasis-open.org/wsrp/wsrp-extra-2.0.xsd for reuse in extension elements.

This WSDL defines the following portTypes:

- **WSRP_v2_Markup_PortType:** All Producers MUST expose this portType.
- **WSRP_v2_ServiceDescription_PortType:** All Producers MUST expose this portType.
- **WSRP_v2_Registration_PortType:** Only Producers supporting in-band registration of Consumers need expose this portType.
- **WSRP_v2_PortletManagement_PortType:** Producers supporting the Portlet management interface expose this portType. If this portType is not exposed, the Portlets of the service are not configurable by Consumers.

This WSDL defines the following SOAP bindings for these portTypes:

- **WSRP_v2_Markup_Binding_SOAP:** All Producers MUST expose a port with this binding for the WSRP_v2_Markup_PortType (the `Markup` portType).
- **WSRP_v2_ServiceDescription_Binding_SOAP:** All Producers MUST expose a port with this binding for the WSRP_v2_ServiceDescription_PortType (the `ServiceDescription` portType).
- **WSRP_v2_Registration_Binding_SOAP:** Producers supporting the `Registration` portType MUST expose a port with this binding for the WSRP_v2_Registration_PortType.
- **WSRP_v2_PortletManagement_Binding_SOAP:** Producers supporting the `PortletManagement` portType MUST expose a port with this binding for the WSRP_v2_PortletManagement_PortType.

Producers SHOULD NOT expect Consumers to use a mixture of WSRP v1 and v2 ports when interacting with it. Producers SHOULD use a single wsdl file with distinct service elements for their v1 and v2 support if they support artifacts, such as registrationHandle or portletState, being used with all operations regardless of the version of the standard defining that operation and SHOULD use separate wsdl files for specifying support for different versions of WSRP when artifacts are usable only within the operations defined by a specific version of the WSRP standard.

---

16 References

16.1 Normative References

[CC/PP]	http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/
[Character Sets]	http://www.iana.org/assignments/character-sets
[MTOM]	http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/
[Namespaces]	http://www.w3.org/TR/REC-xml-names/
[RFC2119]	S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
[Schema]	http://www.w3.org/TR/xmlschema-0/
[SchemaTypes]	http://www.w3.org/TR/xmlschema-2/
[SOAP]	http://www.w3.org/TR/SOAP/
[SSL/TLS]	http://www.ietf.org/html.charters/tls-charter.html
[URI/URL]	http://www.ietf.org/rfc/rfc2396.txt
[WSDL]	http://www.w3.org/TR/wsdl

16.2 Non-Normative References

[Boyer-Moore]	http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/
[J2EE]	http://java.sun.com/j2ee/

[JSR168]	http://www.jcp.org/jsr/detail/168.jsp
[.NET]	http://www.microsoft.com/net/
[P3P]	http://www.w3.org/TR/P3P/
[Security Tech Note]	http://docs.oasis-open.org/wsrp/TechNotes/wsrp-security.doc
[Requirements]	http://www.oasis-open.org/committees/wsia/documents/Requirements2002-09-17.html
[SAML]	https://www.oasis-open.org/committees/security/
[UDDI]	http://www.uddi.org/specification.html
[WS-I.org]	http://www.ws-i.org/
[ebXML Registry]	http://www.oasis-open.org/committees/regrep/
[WSRP Whitepaper]	Thomas Schaeck and Richard Thompson, Web Services for Remote Portals (WSRP) Whitepaper, http://docs.oasis-open.org/wsrp/Misc/Whitepaper.doc , 28 May, 2003
[WS-N]	http://www.oasis-open.org/committees/wsn/
[WS-Security]	http://www.oasis-open.org/committees/wss/
[XACML]	https://www.oasis-open.org/committees/xacml/
[XCBF]	http://www.oasis-open.org/committees/xcbf/
[XForms]	http://www.w3.org/TR/xforms/
[XML Digital Signatures]	http://www.w3.org/Signature/
[XML Encryption]	http://www.w3.org/TR/xmlenc-core/

Appendix A. Glossary (Non-Normative)

Action	A term often used elsewhere for what this specification calls "Interaction".
Attribute	A distinct characteristic of an object. An object's attributes are said to describe the object. Objects' attributes are often specified in terms of their physical traits, such as size, shape, weight, and color, etc., for real-world objects. Objects in cyberspace might have attributes describing size, type of encoding, network address, etc. Salient attributes of an object is decided by the beholder.
Authentication	To confirm a system entity's asserted principal identity with a specified, or understood, level of confidence.
Client	A system entity that accesses a web service.
Consumer	A system entity invoking Producers in a manner conforming to this specification. For example a portal aggregating content from Portlets accessed using the WSRP protocol.
End-User	A person who uses a device specific User-Agent to access a Web site.
Fragment	A piece of markup that is not part of a full document • part of aggregate • generally a markup language • can aggregate a set of fragments

Portlet	Producer hosted component that generates content design for aggregating and processes interactions generated from that content.
Producer	A web service conforming to this specification.
Session	A finite duration interaction between system entities, often involving a user, typified by the maintenance of some state of the interaction for the duration of the interaction.
System Entity	An active element of a computer/network system. For example, an automated process or set of processes, a subsystem, a person or group of persons that incorporates a distinct set of functionality.
Time-Out	A period of time after which some condition becomes true if some event has not occurred. For example, a session that is terminated because its state has been inactive for a specified period of time is said to "time out".
Uniform Resource Locator (URL)	Defined as "a compact string representation for a resource available via the Internet." URLs are a subset of URI.
User-Agent	A system entity that is used by an End-User to access a Web site. A user-agent provides a run-time environment for distributed application components on the client device.
Web Service	A Web Service is a software component that is described via WSDL and is capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP.
Web Site	A hosted application that can be accessed by an End user using a user-agent.
WSRP Service	Presentation oriented, interactive web services that can be aggregated by consuming applications • WSRP services can be published, found, and bound in a standard manner, describing themselves with standardized metadata - WSRP services can be published, found, and bound in a standard manner, describing themselves with standardized metadata
XML (Extensible Markup Language)	Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879] See http://www.w3.org/TR/REC-xml.
XML Namespace	A name, identified by a URI reference, which are used in XML documents as element types and attribute names. An XML namespace is often associated with an XML schema. See http://www.w3.org/TR/REC-xml-names/ .

Appendix B. Common Values (Non-Normative)

There is significant value to defining values for various fields that Consumers and Producers share without having to map to different values with the same semantic meaning. The following sections define such common values.

B.1 Standard User Categories

To ease the mapping of End-Users to user categories and to facilitate plug-and-play, the following standard category names are provided along with an abstract definition of semantics associated with each. The specific semantics of these categories are left to each Portlet's implementation.

- `wsrp:full`: The content for this user category will typically encompass the full functionality of the Portlet.
- `wsrp:standard`: This user category is typically associated with End-Users who may customize some set of properties for a Portlet.
- `wsrp:minimal`: This user category is typically associated with End-Users who may view a Portlet on a page but not modify any of its properties.

Appendix C. Types of state (Non-Normative)

This specification defines a number of types of state and state-passing mechanisms. The following table is an attempt to elucidate the differences between these in one place:

Type of state	Comments
`interactionState`	This type of state is encoded by Portlets in interaction URLs generated by the Portlet, and is supplied to the Portlet by the Consumer when the End-User activates the URL.
`uploadData`	Data the End-User is uploading to the Portlet.
`formParameters`	Name/value pairs the End-User is supplying to the Portlet, usually via a form submission.
Navigational state	This type of state is intended to support page reloads (including from bookmarks) without the view generated by the Portlet changing in a manner unexpected by the End-User. The Consumer resupplies a portlet's most recent navigational state on relevant operations. The Portlet may return the values it wants to be supplied in the future on the response from relevant operations or encode such values in URLs such that they get applied when the URL is activated. The two aspects of this type of state are: 1. **opaque:** This aspect of navigational state is reflected in the protocol in a single field (i. e. it is the Portlet/Producer's task to encode all items which are not being exposed to the Consumer into a single field). 2. **publicly exposed:** This aspect of navigational state requires the Portlet to provide a description of each exposed item which then permits the Consumer to impact the value for that item such that state-based coordination can be accomplished. Both aspects of navigational state can be set anywhere either of them can be set and the Consuemr is required to manage both aspects in an equivalent manner.
Session state	This type of state provides for communication between the Producer and Consumer regarding transient state that is instantiated and managed on the Producer. Either can invalidate this state as part of their normal processing (The Producer by destroying the session and the Consumer by not supplying the session information on future operations). This session state defined by this protocol is opaque to the Consumer and reflected in the protocol in the `sessionID` fields. The Producer returns a value for the `sessionID` whenever a new session is created and the Consumer returns it for future operations to enable use of the session for processing a request.

`portletState`	This state reflects customizations relative to a particular use of a Portlet. This tends to have a long span and therefore be stored in a manner that is unaffected by the Producer application being stopped and restarted. Note that this type of state only appears in the protocol when the Consumer is tasked with managing it for the Producer, but conceptually applies even when the Producer manages the storage of the state.
`registrationState`	This state reflects the business and technical relationship between the Producer and Consumer. Note that this type of state only appears in the protocol when the Consumer is tasked with managing it for the Producer, but conceptually applies even when the Producer manages the storage of the state.
`exportContext`	This contains shared state for a set of exported Portlets.
`exportState`	This contains state specific to a particular exported Portlet.

Appendix D. Coordination mechanisms (Non-Normative)

This specification defines a number of coordination mechanisms. The following table is an attempt to elucidate the differences between these in one place:

Coordination Mechanism	Comments
Events	Events provide a notification mechanism that something has occurred. This notification may, but is not required to, carry data related to the notification.
`publicValues`	This portion of a Portlet's navigational state is intended to support Consumer-mediated sharing of values across multiple Portlets. The value for any particular `publicValue` may be supplied by a Portlet to the Consumer at any of the times where the opaque portion of navigational state can be supplied. Consumer policy controls when values derived from sources other than the Portlet impact one of these items. Note that navigational state is stored and managed on the Consumer and therefore all components of this state (incuding `publicValues` are included on operations where any of it is included.
Producer-mediated coordination	While not strictly defined by this specification, the groupID parameter is intended as an enabling means for Producer-mediated coordination between a set of Portlets deployed just on that Producer.

Appendix E. Data Structures List (Non-Normative)

Alphabetical listing of the defined data structures along with section numbers.

Appendix F. Acknowledgments (Non-Normative)

F.1 WSRP Technical Committee members

The following individuals were voting members of the WSRP technical committee during the development of this

specification:

Polina Alber, Novell
Subbu Allamaraju, BEA Systems Inc.
Olin Atkinson, Novell
Atul Batra, Sun Microsystems
Wesley Budziwojski, Sun Microsystems
Rex Brooks, Individual
Mike Caffyn, NetUnity Software
Andrew Datars, Microsoft Corporation
Michael Freedman, Oracle Corporation
Stefan Hepper, IBM Corporation
Richard Jacob, IBM Corporation
Andre Kramer, Citrix Systems, Inc
Dan Machak, Tibco Software, Inc.
Satish Ramaswamy, Vignette Corporation
Rich Thompson, IBM Corporation

Footnotes

[1] http://www.oasis-open.org/committees/wsia/use_cases/index.shtml

[2] http://www.w3.org/TR/xforms/

[3] http://www.w3.org/TR/xmlschema-1/#xsi_type

[4] http://www.ietf.org/rfc/rfc2109.txt

[5] http://www.w3.org/TR/xmlschema-1/#xsi_type

[6] http://www.ietf.org/rfc/rfc2616.txt

[7] http://lcweb.loc.gov/standards/iso639-2/langcodes.html

[8] http://www.iso.org/iso/en/prods-services/iso3166ma/index.html

[9] http://lcweb.loc.gov/standards/iso639-2/langcodes.html

[10] http://www.iso.org/iso/en/prods-services/iso3166ma/index.html

[11] http://www.iana.org/assignments/media-types/

[12] http://www.ietf.org/rfc/rfc2616.txt

[13] http://www.iana.org/assignments/character-sets

[14] http://www.ietf.org/rfc/rfc1522.txt

[15] http://www.ietf.org/rfc/rfc1522.txt

[16] http://www.ietf.org/rfc/rfc2109.txt

[17] http://www.cs.utexas.edu/users/moore/best-ideas/string-searching/

[18] http://www.w3.org/TR/html40/charset.html - doc-char-set

[19] http://www.ietf.org/rfc/rfc2109.txt

[20] http://www.ietf.org/rfc/rfc3986.txt

[21] http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip