Formatted: Font: Bold
Deleted: 0.91
Deleted: 2/3/2003

Web Services for Remote Portlets Specification

Working Draft 0.91, 17 March 2003

Deleted: 3 February 2003

Document identifier:

WSRP_Specification-v0.91 (Word)

Location:

http://www.oasis-open.org/committees/wsia

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

Editors:

Alan Kropp, Vignette Corporation <akropp@vignette.com>
Carsten Leue, IBM Corporation <cleue@de.ibm.com>
Rich Thompson, IBM Corporation <richt2@us.ibm.com>

Contributors:

Chris Braun, Novell <cbraun@silverstream.com>
Jeff Broberg, Novell <jbroberg@silverstream.com>
Mark Cassidy, Netegrity <mcassidy@Netegrity.com>
Michael Freedman, Oracle Corporation <Michael.Freedman@oracle.com>
Timothy N. Jones, CrossWeave <tim@crossweave.com>
Thomas Schaeck, IBM Corporation <schaeck@de.ibm.com>
Gil Tayar, WebCollage <Gil.Tayar@webcollage.com>

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 specification is a joint effort of two OASIS technical committees. Web Services for Interactive Applications (WSIA) and Web Services for Remote Portals (WSRP) aim to simplify the integration effort through a standard set of web service interfaces allowing integrating applications to quickly exploit new web services as they become available. The joint authoring of these interfaces by WSRP and WSIA allows maximum reuse of user-facing, interactive web services while allowing the consuming applications to access a much richer set of standardized web services.

This joint standard layers on top of the existing web services stack, utilizing existing web services standards and will leverage emerging web service standards (such as security) as they become available. The interfaces are defined using the Web Services Description Language (WSDL).

Status:

This draft is an early version of the public specification. Comments about points needing clarification are much appreciated and may be emailed to *Rich Thompson*.

If you are on the wsia@lists.oasis-open.org or wsrp@lists.oasis-open.org list for committee members, send comments there. If you are not on that list, subscribe to the wsia-comment@lists.oasis-open.org or wsrp-comment@lists.oasis-open.org list and send comments there. To subscribe, send an email message to wsia-comment-request@lists.oasis-open.org or wsrp-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

The errata page for this specification is at http://www.oasis-open.org/committees/wsrp/specification_v1_errata.html.

Table of Contents

1 Introduction

The Web Services for Remote Portlets specification defines a web service interface for accessing and interacting with user-facing, interactive presentation-oriented web services. It has been produced through the joint efforts of the Web Services for Interactive Applications (WSIA) and Web Services for Remote Portals (WSRP) OASIS Technical Committees. It is based on the requirements gathered by both committees and on the concrete proposals to both committees.

Scenarios that motivate WSRP/WSIA functionality include:

- Portal servers providing portlets as user-facing web services that can be used by aggregation engines.
- Portal servers consuming user-facing web services provided by portal or non-portal content providers and integrating them into a portal framework.

However this description also applies to non-portal environments, mostly identified by the WSIA use cases[1]. 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.

Deleted: ***[.Net]***

Formatted: English (U.S.)

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-Security and WS-Policy (see section 3.1) [A102] in future versions.

Deleted: ***[SSL/TLS]***

Formatted: English (U.S.)

Formatted: English (U.S.)

Deleted: ***[URI/URL]***

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 an End-User.

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

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 user-facing web service interface that allows the inclusion of and interaction with content from a web service. Such a user-facing web service provides both application logic and presentation logic. This specification provides a common protocol and a set of interfaces for user-facing web services. Thus, aggregating applications can easily adopt these web services by utilizing generic proxy code.

1.2 Actors

This protocol describes the conversation between Producers and Consumers on behalf of End-Users (clients of the Consumer). Producers are user-facing web services that host Portlets which are able to render markup fragments and process user interaction requests. Consumers use these web services to present the generated markup to End-Users and manage the 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:

- *Self Description:* A required interface that allows Consumers to find out the capabilities of the Producer 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 persistent state.

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.

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 via the Producer. 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. 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 intermediary system that communicates with user-facing web services (i.e. Producers and the Portlets they host) on behalf of its users. It gathers and aggregates the markup delivered by the Portlets and presents the aggregation to the End-User. One typical Consumer is a portal, which mediates the markup and the interaction with this markup between End-Users and user-facing web services. Another typical Consumer is an e-Commerce application that aggregates manufacturer-provided content into its own pages. 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.

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 that aggregates content from various Producers/Portlets is the preparation and presentation of 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.

Sasha to propose language for what the protocol enables End-Users to do.

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

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. 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. 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 resources 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]***.

Compliance: Mandatory – relevant to legal rules, regulations or laws. Compliancy is the act of complying with a specification and/or standard. Example: ISO 9001. IEEE defines as complying with laws and regulations.

Conformance: Not mandatory – ISO/IEC Guide 2 defines conformance or conformity as fulfillment of a product, process or service of specified requirements.

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]).

3 General Considerations

The major design goals of 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

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

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

Namespaces – Defines how XML Namespaces are declared and used.

SOAP – Defines how to invoke web service interfaces.

SSL/TLS – Defines secure transport mechanisms.

URL – Defines URI (includes URL) syntax and encoding

Character Sets - Character set encoding

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

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

XACML – Defines a syntax for expressing authorization rules.

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

3.1.2 Emerging Standards

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

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

RLTC – Defines a syntax for expressing authorization rules.

XCBF – Defines how to exchange biometric data.

WS-Attachments - Defines how to encapsulate a SOAP message and zero or more attachments within a DIME message.

WS-I.org - Defines profiles for use of web services standards such that interoperability is maximized.

DIME – A lightweight, binary message format that encapsulates one or more resources in a single message construct.

JSR168 – Java Community Process for standardizing a portlet API.

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 14. 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 calls must be made 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.

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 SHOULD 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 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:

Persistent: This lifecycle starts with an explicit operation to create the item and ends only with an explicit operation to destroy the item. Examples include the `registrationHandle` and 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 any resources at the Consumer related to the item may be reclaimed at an appropriate time. An example of this 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 a persistent lifecycle. This scope is referenced throughout the protocol using a `registrationHandle`. The `registrationHandle` is created and destroyed using either in band mechanism, i.e. by declaring support for the `Registration` portType, or by out of band mechanism, whereby the `registrationHandle` is created and destroyed by means outside this specification. If the Producer exposes the `Registration` portType, then the Consumer MUST respect the registration requirements established by this specification.

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. This scope ends for Consumer Configured Portlets when the reference to the Portlet is explicitly released. As such it encompasses all interactions with the Portlet. This scope has a persistent 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 `PortletMangement` portType, then the Consumer MAY clone the Producer Offered Portlets and uniquely configure them for its own use. The Consumer MAY also choose to directly use the Producer Offered Portlets.

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

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 handle. This scope has a transient lifecycle and is established by the Producer returning a new `SessionContext`. The Consumer MUST respect this new scope as described in section 6.1.1.

3.6 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. The Producer returns this state to the Consumer as `navigationalState` such that it may satisfy these expectations of the End-User. To supply the bookmarking and page refresh capabilities End-Users expect, the Consumer MAY store this 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 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 transient 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. This type of state will be referred to as a **Session** (similar to an HTTP Session) and an opaque reference to one is a `sessionID`.

Persistent state: This is state that is persisted until either the Consumer or Producer explicitly discards it. This specification defines two kinds of persistent 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. 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).

3.7 Persistence and statefulness

This specification does not mandate that either the Producer or the Consumer is stateful [A201]. In the **getMarkup()** and **performInteraction()** calls, the `navigationalState` 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. 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].

3.8 Producer Mediated Sharing

Producers may implement a sharing mechanism 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. The Consumer MUST respect this grouping as detailed in section 6.4.

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.

3.9 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.

3.10 Two-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. For these reasons, there is a “two-step” capability built into the protocol.

In this two-step interaction, the Consumer first invokes either **performInteraction()** or **performBlockingInteraction()** on the Portlet whose markup the End-User interacted with. In the case of **performBlockingInteraction()**, the Consumer MUST block all other invocations within the context of the initiating request from the client of the Consumer until either the receipt of a response or the invocation fails (e.g. times out). The Consumer then invokes **getMarkup()** on the Portlets being aggregated.

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, regardless of whether the interaction may have side effects on other Portlets at the Producer, is well-defined independent of the order of **getMarkup()** invocations on the Portlets. Portlets specifying the invocation of **performInteraction()** (i.e. non-blocking state changes) when URLs are activated from their markup need to ensure this same determinism in the resulting aggregated markup.

3.11 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. 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”.

3.12 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 MUST 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 the cookie is 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,
                                           sendAllLocales);
```

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 is in this interface avoids the problems associated with moving cookies between bindings.

```
MarkupResponse = getMarkup(RegistrationContext, PortletContext, RuntimeContext,
                           UserContext, MarkupParams);
```

```
InteractionResponse = performInteraction(RegistrationContext, PortletContext,
                                         RuntimeContext, UserContext,
                                         MarkupParams, InteractionParams);
```

```
BlockingInteractionResponse = performBlockingInteraction(RegistrationContext,
                                PortletContext, RuntimeContext, UserContext,
                                MarkupParams, InteractionParams);
```

```
ReturnAny = initCookie(RegistrationContext);
```

```
ReturnAny = releaseSessions(RegistrationContext, sessionIDs);
```

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);
```

```
RegistrationState = modifyRegistration(RegistrationContext, RegistrationData);
```

```
ReturnAny = deregister(RegistrationContext);
```

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,
                                sendAllLocales);
```

```
PortletContext = clonePortlet(RegistrationContext, PortletContext, UserContext);
```

```
DestroyPortletsResponse = destroyPortlets(RegistrationContext, portletHandles);
```

```
PortletContext = setPortletProperties(RegistrationContext, PortletContext, UserContext,
                                PropertyList);
```

```
PropertyList = getPortletProperties(RegistrationContext, PortletContext, UserContext,
                                names);
```

```
PortletPropertiesDescriptionResponse = getPortletPropertyDescription(RegistrationContext,
                                PortletContext, UserContext,
                                desiredLocales, sendAllLocales);
```

5 Service Description Interface

A Producer may be discovered through mechanisms such as ***[UDDI]*** or ***[WSIL]***, 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 normative definitions for all data structures are contained in the WSDL referenced in section 0. For the convenience of the reader, this non-normative section uses an IDL like syntax to describe these structures, where the leading [R] indicates a field is required and [O] indicates it is optional. The operations in this section introduce the following data structures:

Deleted: 14

5.1.1 Extension Type

The `Extension` structure contains the payload extension mechanism for vendor and application extensions. This allows arbitrary elements from other namespaces to be sent as part of containing data structures. 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 MUST declare its type using the schema-defined "type" attribute[4]. We would encourage these to either be of type `xsd:string` or be explicitly typed in a WSDL file that carries the relevant type definitions so that Consumers MAY prepare the appropriate serializer/deserializer. 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
    [O] Object  any[]
```

Members:

- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

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. We STRONGLY RECOMMEND these characters be chosen from the first 127 characters of the Unicode character set. Not following this recommendation will likely cause information to be lost as the Consumer stores and retrieves the value. The Consumer MAY ignore any character in a handle that falls outside this range.

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

```
Handle extends 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. Not following this recommendation will likely cause information to be lost as the value is stored and retrieved.

```
Key extends 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. 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 extends string (maximum length = 4096)
```

5.1.5 LocalizedString Type

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

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

Members:

- `xmlLang`: The locale for this supplied localized value. This is carried in the WSDL using the `xml:lang` attribute.
- `resourceName`: The name assigned to this localized string for dereferencing into a `ResourceList` for values from other locales.
- `value`: The value for this localized string in the declared locale.

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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.9 UserCategoryDescription Type

This structure is used to describe the user categories a Consumer MAY assert for an End-User when interacting with the Portlets at the Producer. The Consumer MUST NOT assert a user category for which no `UserCategoryDescription` was part of the Producer's `ServiceDescription`. Portlets MUST NOT declare support for user categories that are not part of the Producer's `ServiceDescription`. Note that user categories are Producer-wide and therefore are inherently shared by the Producer's Portlets.

```
UserCategoryDescription
    [R] string          categoryName
    [R] LocalizedString description
    [O] Extension       extensions[]
```

Members:

- `categoryName`: The name for this user category. The preferred form is a URI such that it is definitively namespaced.
- `description`: A localized, free form description of the user category. Expected use of this field is for display at the Consumer to someone who will provide a mapping to Consumer information about the End-User.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.10 MarkupType Type

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

```
MarkupType
    [R] string        markupType
    [R] string        locales[]
    [R] string        modes[]
    [R] string        windowStates[]
    [O] Extension     extensions[]
```

Members:

- `markupType`: A mime type supported by the Portlet (e.g. *HTML, XHTML, WML, VoiceXML, cHTML*) for which the remainder of this structure applies.
- locales: An array of locales for which this `markupType` is available.
- modes: The `modes` (defined in section 6.8) that are supported by the Portlet for this `markupType`.
- windowStates: The `windowStates` (defined in section 6.9) that are supported by the Portlet for this `markupType`.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace

5.1.11 PortletDescription Type

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

```
PortletDescription
    [R] Handle          portletHandle
    [R] MarkupType      markupTypes[]
    [O] ID              groupID
    [O] LocalizedString description
    [O] LocalizedString shortTitle
    [O] LocalizedString title
    [O] LocalizedString keywords[]
    [O] string          userCategories[]
    [O] string          userProfileItems[]
    [O] boolean         usesMethodGet
    [O] boolean         defaultMarkupSecure
    [O] boolean         userContextStoredInSession
    [O] boolean         templatesStoredInSession
    [O] boolean         hasUserSpecificState
    [O] boolean         doesUrlTemplateProcessing
    [O] Extension       extensions[]
```

Members:

- `portletHandle`: The handle by which Consumers MAY refer to this Producer Offered Portlet. Note that Handles are restricted to a maximum length of 255 characters.
- `markupTypes`: Each member of this array specifies metadata for a single `markupType`.
- `groupID`: Identifier for the group within which the Producer places this Portlet or any Portlets derived from it via the cloning process.
- `description`: Localized descriptions of the Portlet. This is intended for display in selection dialogs, etc.
- `shortTitle`: Localized short title for the Portlet.
- `titles`: Localized title for the Portlet.

- `keywords`: Array of localized keywords describing the Portlet which can be used for search, etc.
- `userCategories`: Array of category names for the Producer's user categories that the Portlet supports. Note: This support MAY be provided by the Producer service on behalf of the Portlet. Each user category declared as supported by the Portlet MUST have a `UserCategoryDescription` 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 this portlet, 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.
- `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. Not requiring this of all invocations can result in a significant performance difference. Note that the Consumer MAY send `UserContext` information on any invocations regardless of the value of this flag. 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 regardless of the value of this flag. The default value of this flag is "`false`".
- `hasUserSpecificState`: A flag indicating the Portlet will store persistent state specific to each End-User. Setting this flag to "`true`" suggests to the Consumer that it MAY want to clone the Portlet when placing it on an aggregated page rather than waiting for the processing described in section 6.3.3. 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 this flag to "`true`", Consumers MUST provide the URL writing templates. The default value of this flag is "`false`".
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.12 Property Type

The `Property` data structure is used to carry typed information from the Consumer to the Producer.

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

Members:

- `name`: Name of the property, MUST have a non-zero length.
- `xmlLang`: The locale for the supplied localized value. This is carried in the WSDL using the `xml:lang` attribute.
- `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 14 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.

5.1.13 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] string      name
```

Members:

- `name`: Name of the property, MUST have a non-zero length.

5.1.14 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.15 PropertyDescription Type

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

```
PropertyDescription
    [R] string          name
    [R] QName           type
    [O] LocalizedString label
    [O] LocalizedString hint
    [O] Extension       extensions[]
```

Members:

- `name`: Name of the property, MUST have a non-zero length.
- `type`: Type of the property. For those not familiar with the QName type, we note that this is 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.
- `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.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.16 ModelDescription Type

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

```
ModelDescription
    [O] PropertyDescription     propertyDescriptions[]
    [O] object                  modelTypes
    [O] Extension               extensions[]
```

Members:

- `propertyDescriptions`: 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.17 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 the returned cookie 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 the returned cookie with subsequent invocations on behalf of that user targeting Portlets with identical `groupIDs`.

5.1.18 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] UserCategoryDescription userCategoryDescriptions[]
    [O] CookieProtocol          requiresInitCookie
    [O] ModelDescription        registrationPropertyDescription
    [O] string                  locales[]
    [O] ResourceList            resourceList
    [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` field to all operations with this parameter.
- `offeredPortlets`: An array of structures (defined in Section 5.1.11) containing the metadata for the Producer Offered Portlets.
- `userCategoryDescriptions`: An array of `UserCategoryDescription` structures as defined in Section 5.1.9. This array MUST include an entry for any user category the Producer is willing to have the Consumer assert for an End-User, including the user category names defined in section 1B.2 of this specification.
- `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 be supply during registration.
- `locales`: An array of locales for which the localized strings of this `serviceDescription` are available. 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.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.19 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 push the storage of state at registration scope to the Consumer.

```
RegistrationState
    [O] base64Binary   registrationState
    [O] Extension      extensions[]
```

Members:

- `registrationState`: This field is used only when the Producer wants the Consumer to provide persistent storage for the state resulting from processing the registration. If this field has a value, the Consumer MUST return this value on any subsequent calls in the context of this registration [R362].
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.20 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] 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 persistent storage for the state resulting from processing the registration. If this field has a value, the Consumer MUST return this value on any subsequent calls in the context of this registration [R362].
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

5.1.21 desiredLocales and sendAllLocales

These two parameters are used to control what locales are used when 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 the `sendAllLocales` boolean flag is set to “true”, the Producer’s response SHOULD contain 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,
                                           sendAllLocales);
Faults: Security.InvalidRegistration, Interface.OperationFailed
```

Producers may choose to restrict the information returned in `serviceDescription` based on the supplied `RegistrationContext`. The minimum information a Producer MUST return is that which declares what is required for a Consumer to register (i.e. the `requiresRegistration`

flag and 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 description 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 and `sendAllLocales` (a boolean) as an indication that values for all locales are desired.

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

6 Markup Interface

As user-facing, interactive presentation-oriented web services, WSIA or WSRP compliant services MUST 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 normative definitions for all data structures are contained in the WSDL referenced in section 0. For the convenience of the reader, this non-normative section uses an IDL like syntax to describe these structures, where the leading [R] indicates a field is required and [O] indicates it is optional. The operations in this section introduce the following data structures:

Deleted: 14

6.1.1 SessionContext Type

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

```
SessionContext
    [R] ID          sessionID
    [R] 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 resources. A value of –1 indicates that the `sessionID` will never expire.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

If a Producer throws a `Interface.InvalidSession` fault after returning a `sessionID`, the Consumer MUST reinvoke the operation that caused the fault to be thrown without the invalid `sessionID` and include any data that had not been previously sent due to flags indicating the data would be stored in the session.

6.1.2 RuntimeContext Type

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

RuntimeContext	
[R] string	userAuthentication
[O] Key	portletInstanceKey
[O] string	namespacePrefix
[O] Templates	templates
[O] ID	sessionID
[O] Extension	extensions[]

Members:

- `userAuthentication`: String indicating how the End-User was authenticated. Common values include:
  - "`none`": No authentication was done, user information is asserted for informational purposes only.
  - "`password`": The End-User identified themselves using the common userid/password scenario.
  - "`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 MAY supply as a reference to its use of the Portlet. The intent of this reference is to allow the Portlet, whenever needed, to namespace multiple instances of itself in an optimal manner. Examples include within any Producer-defined data sharing mechanisms and multiple inclusions on a Consumer page. Since this reference is a `Key`, its length is restricted to 255 characters. Consumer SHOULD keep it as short as possible.
- `namespacePrefix`: This field provides a string the Portlet MAY use to prefix tokens that need to be unique on the aggregated page (e.g. JavaScript variables, html id attributes, etc.).
- `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`.
- `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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.3 PortletContext Type

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

```
PortletContext
    [R] Handle          portletHandle
    [O] base64Binary    portletState
    [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 persistent state [A205]. If the `portletState` field has a value, the Consumer MUST return this value on subsequent calls using the same `portletHandle`. Note that such uses MAY span multiple starting and stopping cycles of the Consumer and therefore this state MUST be persisted by the Consumer until successfully invoking **destroyPortlets()** with the related `portletHandle`.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.4 UserScope Type

The `UserScope` structure is a restriction on string that enumerates the following values:

- "`perUser`": The markup is specific to the userContext for which it was generated. Changes to the data of the `UserContext` MUST invalidate the markup.
- "`forAll`": The markup is not specific to the `UserContext` and therefore may be supplied to all users of the Consumer.
- "`other`": The caching relative to users fits neither of the above descriptions.

6.1.5 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] UserScope       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 `userScope` has a value of "other, the Consumer SHOULD look in the extensions field for additional information. If no user scope for caching the markup is established, the Consumer SHOULD 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.6 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      actionTemplate
    [O] string      blockingActionTemplate
    [O] string      renderTemplate
    [O] string      resourceTemplate
    [O] string      secureDefaultTemplate
    [O] string      secureActionTemplate
    [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”.
- `actionTemplate`: This template provides the template for URLs that will be directed to the Consumer and processed as a **performInteraction()** on the Portlet.
- `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 as an HTTP GET on the named resource.
- `secureDefaultTemplate`: This template provides the default value for all the secure template fields.
- `secureActionTemplate`: This template provides the template for secure URLs that will be directed to the Consumer and processed as a **performInteraction()** on the Portlet using a secure protocol.
- `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 as an HTTP GET over SSL/TLS on the named resource.

- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.7 ClientData Type

The `MarkupParam` structure carries information concerning the user agent using this type.

```
ClientData
    [R] string        userAgent
    [O] Extension     extensions[]
```

Members:

- `userAgent`: String identifying the user-agent of the End-User.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.8 NamedString Type

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

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

Members:

- `name`: The name to be associated with this value.
- `value`: The associated value.

6.1.9 MarkupParams Type

The `MarkupParams` structure contains a set of fields 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.

```
MarkupParams
    [R] ClientData        clientData
    [R] boolean           secureClientCommuncation
    [R] string            locale[]
    [R] string            markupType[]
    [R] string            mode
    [R] string            windowState
    [O] string            navigationalState
    [O] string            markupCharacterSet[]
    [O] string            validateTag
    [O] string            validNewModes[]
    [O] string            validNewWindowStates[]
    [O] Extension         extensions[]
```

Members:

- `clientData`: A structure (defined in section 0) that provides information (including `userAgent`) about the client device which will render the markup.
- `secureClientCommunication`: A flag indicating whether or not the delivery channel between a client and Consumer is secure [R401]. The Consumer MUST set the `secureClientCommunication` flag as the Portlet MAY render different content when it knows the delivery channel is secure.

- `locale`: An array of locales where the order in the array is the order in which the Consumer would prefer the Portlet generate the markup (e.g. "en-US"). Note that current practice on the Internet uses the format [2 char language code][5] "-" [2 char country code][6] as per the provided example. The Consumer MAY supply this information based on the setting the End-User has requested.
- `markupType`: An array of Mime types[7] (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[8] MAY be specified. Portlets SHOULD generate markup in one of the specified Mime types.
- `mode`: The mode for which the Portlet should render its output. A set of modes is defined in this specification (see section 6.8). The Consumer SHOULD inspect the Portlet's metadata to determine which of these modes the Portlet supports in addition to any Producer-defined modes. The Consumer MUST specify either one of the modes from the Portlet's metadata or "`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 6.9. The Consumer MUST specify either one of the windowStates from the Portlet's metadata or "`normal`" (all Portlets are required to support this windowState).
- `navigationalState`: This field contains the opaque navigational state for this portlet. To exist, navigational state must be set explicitly on each request either by setting its value upon return from a blocking action or in a non-blocking or render URL.
- `markupCharacterSet`: An array of `characterSets`[9] (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 this character set or UTF-8 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 `MarkupContext` structure. When this field has a value, the Consumer is indicating it has markup cached for the Portlet, but the `CacheControl` structure governing the use of that cached markup no longer indicates it is valid. The Consumer is supplying the `validateTag` as a means for the Portlet to avoid generating new markup if the cached markup can be validated. The Portlet sets the `useCachedMarkup` field in the returned `MarkupContext` to "true" to indicate the markup referenced by the `validateTag` is still valid.

---

[5] http://lcweb.loc.gov/standards/iso639-2/langcodes.html
[6] http://www.din.de/gremien/nas/nabd/iso3166ma/codlstp1/en_listp1.html
[7] http://www.isi.edu/in-notes/iana/assignments/media-types/media-types
[8] http://www.ietf.org/rfc/rfc2616.txt
[9] http://www.iana.org/assignments/character-sets

- `validNewModes`: Current set of modes the Producer MAY request changing to. These can be used to specify a mode change either via an `InteractionResponse` 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 MAY assume that all transitions are valid. Consumers wishing to prohibit all transitions SHOULD supply 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 `InteractionResponse`. 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 MAY assume that all transitions are valid. Consumers wishing to prohibit all transitions SHOULD supply just the current `windowState` in this array.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

It is STRONGLY RECOMMENDED that custom `modes`, `windowStates`, and `userAuthentication` values be URI's or prefixed in a manner designed to reduce name clashes with any values that may be defined by future versions of this specification.

6.1.10 MarkupContext Type

The `MarkupContext` structure contains fields relative to returning markup from various invocations.

MarkupContext	
[O] boolean	useCachedMarkup
[O] string	markupType
[O] string	markupString
[O] base64Binary	markupBinary
[O] string	locale
[O] boolean	requiresUrlRewriting
[O] CacheControl	cacheControl
[O] string	preferredTitle
[O] Extension	extensions[]

Members:

- `useCachedMarkup`: A boolean used to indicate whether the markup the Consumer indicated it has cached is still valid. The default value of this field is “false” (i.e. new markup is being returned for the Consumer’s use). When this field’s value is “true” the markup field MUST NOT be returned. If field’s value is “true”, any supplied `cacheControl` field MUST be processed as an update to the `cacheControl` originally supplied with the cached markup.
- `markupType`: The mime type of the returned markup. This field MUST be specified whenever markup is returned, and if the `markupBinary` field is used to return the markup, the mime type MUST include the character set for textual mime types. In this particular case this character set MAY be different than the response message.

- `markupString`: The markup to be used for visualizing the current state of the Portlet. This is a string in order to support non-XML markup (e.g. HTML). 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 `MarkupParams` or be UTF-8. When a SOAP binding is used, the character set of the markup returned by the Producer MUST match the character set of the response message's document. This field is only missing when the `useCachedMarkup` flag is "true". This field is mutually exclusive with returning the markup in the `markupBinary` field.
- `markupBinary`: The markup to be used for visualizing the current state of the Portlet returned in binary. This is useful if the text 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 markup in the `markupString` field.

WSDL subgroup to explore/propose a "natural" way to carry XML conformant markup.

- `locale`: The locale of the returned markup. This field MUST be specified whenever markup is returned.
- `requiresUrlRewriting`: A flag by which the Portlet/Producer indicates whether or not Consumer-side URL rewriting (see section 10.2.1) is required. The Consumer MUST parse the markup for URL rewriting if this flag is set to "true". The default value for this flag is "false".
- `cacheControl`: Defines the caching policies for the returned markup fragment. If this field is not supplied, the Consumer MUST treat the returned markup as not cachable.
- `preferredTitle`: The title the Portlet would prefer to be used in any decoration of the markup. The locale and markup type, if any, of the preferred title MUST be identical to that of the markup.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.11 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.12 InteractionResponse Type

The `InteractionResponse` structure contains the various items **performInteraction()** can return.

```
InteractionResponse
    [O] SessionContext   sessionContext
    [O] PortletContext   portletContext
    [O] MarkupContext    markupContext
    [O] Extension        extensions[]
```

Members:

- `sessionContext`: This structure contains session-oriented fields that may be returned from various operations, including a new `sessionID` and the duration before it `expires`.
- `portletContext`: This structure is where a Portlet using Consumer-side persistent storage may return a change in its persistent state, provided the `portletState` flag in `InteractionParams` had been set to "`readWrite`" or "`cloneBeforeWrite`". When the `portletState` 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 6.3.3.
- `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()**.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.13 UpdateResponse Type

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

```
UpdateResponse
    [R] InteractionResponse  interactionResponse
    [R] string               navigationalState
    [O] string               newWindowState
    [O] string               newMode
```

Members:

- `interactionResponse`: This field holds all the information that **performInteraction()** was able to return as well.
- `navigationalState`: Opaque representation of navigational state which the Portlet is returning to the Consumer to indicate the navigational state to be supplied to **getMarkup()** including for page refreshes and page bookmarks. This state is for the purpose of generating markup. The Consumer SHOULD supply this value as the `navigationalState` 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 in order to maintain End-User state. The Consumer is not required to persist the `navigationalState` for longer than this set of interactions, but MAY provide such a persistence if desired.

- `newWindowState`: A request from the Portlet to change the window state. The Consumer MAY choose to respect this request, 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.
- `newMode`: A request from the Portlet to change the mode. The Consumer MAY choose to respect this request, but since the Portlet cannot depend on that choice it MUST NOT encode this new mode into any of its stateful settings. Rather, the Portlet MUST compute any such impact on stateful settings after the Consumer has actually changed the mode.

6.1.14 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.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.15 StateChange Type

This type is a restriction on the string type that is constrained to the values "`readWrite`", "`cloneBeforeWrite`" or "`readOnly`".

6.1.16 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:

- `uploadData`: A binary data blob that is being uploaded.
- `mimeType`: Mime type of what is in the `uploadData` field.
- `mimeAttributes`: Mime attributes that are not represented elsewhere.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.17 InteractionParams Type

The `InteractionParams` structure contains fields specific to invoking either the **performInteraction()** or the **performBlockingInteraction()** operations.

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

Members:

- `portletState`: 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 persistent state of the Portlet in use, it may permit the Producer to clone the Portlet (i.e. set `portletState` 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 6.3.3.
- `interactionState`: Opaque representation of transient information for use in processing this invocation of either **performInteraction()** or **performBlockingInteraction()**. The value for this field is supplied through the portlet URL parameter `wsrp-interactionState` (see section 10.2.1.3).
- `formParameters`: Name/value pairs reflected, for example, in the case of HTML either from the query string of forms submitted with method=get or in a request with mime type = "application/x-www-form-urlencoded" for 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 (i.e. web browsers) encode posted data in the character set of the response that generated the page the form is on. 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.18 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.

```
UserProfile
    [O] UserName        personname
    [O] dateTime        birthdate
    [O] string          gender
    [O] EmployerInfo    employerInfo
    [O] LocationInfo    homeInfo
    [O] LocationInfo    workInfo
    [O] Extension       extensions[]
```

Members:

- `name`: A structure containing the various fields for the End-User's name.

- `birthdate`: 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.
- `workInfo`: The End-User's work location information.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.18.1 UserName Type

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

```
UserName
    [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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.18.2 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.18.3 LocationInfo Type

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

```
LocationInfo
    [O] Address     address
    [O] string      telephone[]
    [O] string      email[]
    [O] string      online[]
    [O] Extension   extensions[]
```

Members:

- `address`: A structure for various fields holding portions of the postal address.
- `telephone`: An array of telephone numbers for the End-User.
- `email`: An array of email addresses for the End-User.
- `online`: An array of URIs for the End-User (usually web-sites).
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.18.4 Address Type

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

```
Address
    [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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

6.1.19 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 (see section 3.1.2) defines how to carry User Information in a SOAP header).

```
UserContext
   [R] ID          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. This key MUST remain invariant for the duration of a Consumer's registration. The Producer MAY use this key as a reference to the user.
- `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. See the discussion of user categories in section 6.10.
- `profile`: End-User profile data structure as defined in section 0. Note that while the `UserContext` structure is passed to many operations, only the interaction oriented operations need this optional field to be supplied.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

A stateful Producer MUST relate its private conversational state with the `userContext` the Consumer supplies so that the Consumer's mapping of a `userContext` structure to a set of stateful interactions produces the intended result for the End-User.

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: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle, Interface.InvalidCookie, Interface.InvalidSession,
        Interface.UnsupportedMode, Interface.UnsupportedWindowState
        Interface.UnsupportedLocale, Interface.UnsupportedMarkupType
```

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 cachability 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 Cachability

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 to keep the cache up-to-date from section 6.2.1.2. If the `cacheControl` field is empty, the Consumer MUST NOT cache the markup fragment.

6.2.1.2 Cache Invalidation

The `expires` field of the `cacheControl` provides a time duration for when the markup SHOULD be considered valid. Once this time has elapsed, counting from the point in time when the `MarkupContext` structure was returned, the Consumer SHOULD 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.

6.3 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 the `navigationalState` or 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. Two operations are defined for processing interactions and the state changes they may cause, one carrying the additional semantics of blocking the Consumer from both beginning the generation of the aggregated page and gathering markup from other Portlets on the page.

6.3.1 performInteraction() Operation

This operation does not carry the semantics of blocking the Consumer's processing of gathering markup:

```
InteractionResponse = performInteraction(RegistrationContext, PortletContext,
                                         RuntimeContext, UserContext,
                                         MarkupParams, InteractionParams);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle, Interface.InvalidCookie, Interface.InvalidSession,
        Interface.UnsupportedMode, Interface.UnsupportedWindowState
        Interface.UnsupportedLocale, Interface.UnsupportedMarkupType
        Interface.PortletStateChangeRequired
```

6.3.2 performBlockingInteraction() Operation

This operation is distinguished from **performInteraction()** in that both the Consumer beginning the generation of the aggregated page (often because the invocation MAY return a `redirectURL`) and the gathering of markup from other Portlets on the page (often because shared state, including via a database, impacts the markup of other Portlets) are blocked. The Consumer MUST initiate at most one **performBlockingInteraction()** for any one client request.

```
BlockingInteractionResponse = performBlockingInteraction(RegistrationContext,
                                         PortletContext, RuntimeContext, UserContext,
                                         MarkupParams, InteractionParams);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle, Interface.InvalidCookie, Interface.InvalidSession,
        Interface.UnsupportedMode, Interface.UnsupportedWindowState
        Interface.UnsupportedLocale, Interface.UnsupportedMarkupType
        Interface.PortletStateChangeRequired
```

Since this is a blocking operation, the Consumer MUST wait for the response 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 also 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 `navigationalState` that will be returned from the operation and not the `navigationalState` that was passed to it. This ensures consistency when the optimization is not used and the Consumer invokes **getMarkup()** after **performBlockingInteraction()** returns.

6.3.3 Updating Persistent Portlet State

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

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

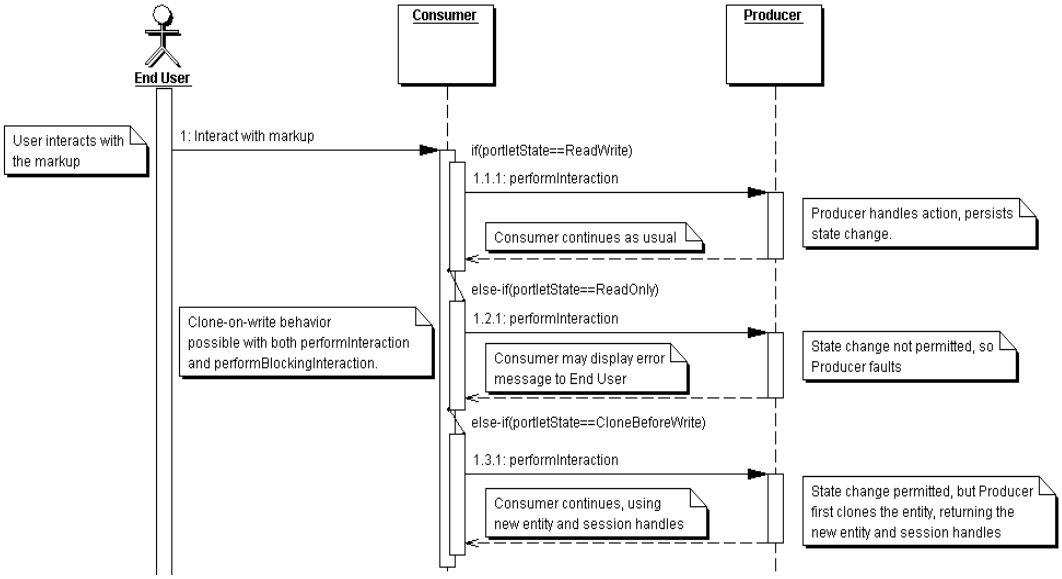
This combination requires that all persistent Portlet state changes happen in a manner that has Consumer approval for the change to occur, while the Portlet decides both when the change is required and its exact character. The Consumer indicates whether or not it is safe for the Portlet to modify its persistent state by setting the `portletState` field in the `interactionParams` structure. If the Consumer has set the `portletState` flag to "`readWrite`", the Portlet MAY modify its persistent state regardless of whether it is persisted on the Producer or Consumer.

If the Consumer has set the `portletState` field to "`cloneBeforeWrite`", persistent state changes are allowed only if the Producer first clones the Portlet. If the Producer does not clone the Portlet, processing attempts to modify persistent state MUST proceed as if the Consumer had specified "`readOnly`" for `portletState`. If the Producer clones the Portlet, processing attempts to modify persistent state on the new Portlet SHOULD proceed as if the Consumer had specified "`readWrite`" for `portletState`. 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`.

If the Consumer has set the `portletState` flag to "`readOnly`", the Portlet MUST NOT modify its persistent 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 persistent state. Commonly Consumer's will only set the `portletState` 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 persistent state and a Portlet is unable to process the interaction without such an update, then the fault "`Interface.PortletStateChangeRequired`" MAY be thrown indicating the interaction processing failed.

This set of possibilities is depicted in the following figure:



6.4 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);
Faults: Security.AccessDenied, Security.InvalidRegistration,
        Interface.OperationFailed
```

If the Producer's metadata has set the `requiresInitCookie` field to any value other than "`none`", then the Consumer MUST invoke **initCookie()** and supply the returned cookie according to the semantics of the value of `requiresInitCookie` as defined in section 5.1.2. If at any time the Producer throws a fault message ("`Interface.InvalidCookie`") indicating the supplied cookie has 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.

6.5 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 resources related to the `sessionIDs`):

```
ReturnAny = releaseSessions(RegistrationContext, sessionIDs);
Faults: Security.AccessDenied, Security.InvalidRegistration,
        Interface.MissingParameters, Interface.OperationFailed
```

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

6.6 Consumer Transitions across Bindings

Consumers SHOULD 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[10]. Consumers MUST respect the security setting on each cookie.

6.7 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.

6.7.1 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()** MAY 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
performInteraction or performBlockingInteraction	sessionContext / sessionID	Producer provided sessionContext / sessionID	No value as this Portlet uses no Producer-side state.
	interactionParameters / interactionState	Consumer extracts value from link.	Interaction state encoded on the URLs in the markup only.
	markupParameters / navigationalState	Consumer extracts value from link.	Navigational state encoded on the URLs in the markup only.

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

	interactionResponse / navigationalState	This type of Portlet does not return navigationalState.	
getMarkup	sessionContext / sessionID	Producer provided sessionContext / sessionID	No value as this Portlet uses no Producer-side state.
	markupParameters / navigationalState	Consumer extracts value from link.	Navigational state from the URL.

6.7.2 Navigational State Only

This type of Portlet does not maintain state at the Producer, but does push navigational state out 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 `navigationalState` field) back to the Consumer. It is then the responsibility of the Consumer to retransmit the `navigationalState` to the Producer with each request [A206].

A stateless Consumer can store the `navigationalState` 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 `navigationalState` 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 `navigationalState`. Only invocations of **performInteraction()** or **performBlockingInteraction()** are allowed to modify the `navigationalState` of a Portlet.

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

Method	Parameter/Field	Value	Comments
performInteraction or performBlockingInteraction	sessionContext / sessionID	Producer provided sessionContext / sessionID	No value as this Portlet uses no Producer-side state.
	interactionParameters / interactionState	Consumer extracts value from link.	Interaction state encoded on the URLs in the markup only.
	markupParameters / navigationalState	Consumer extracts value from link or previous value.	
	interactionResponse / navigationalState	Portlet may compute a changed navigationalState.	
getMarkup	sessionContext / sessionID	Producer provided sessionContext / sessionID	No value as this Portlet uses no Producer-side state.
	markupParameters / navigationalState	From link or from performInteraction.	

6.7.3 Local state

Portlets storing state locally on the Producer establish a session and return an opaque reference (i.e. a `sessionID`) which the Consumer MUST then return on all subsequent invocations of this Portlet instance on the aggregated page for this End-User. These Portlets MAY also push 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
performInteraction or performBlockingInteraction	sessionContext / sessionID	Producer provided sessionContext / sessionID	With Producer side state, the session handle offers ability to store information without impacts message size to Consumer.
	interactionParameters / interactionState	Consumer extracts value from link.	Interaction state encoded on the URLs in the markup only.
	markupParameters / navigationalState	Consumer extracts value from link or previous value.	
	interactionResponse / navigationalState	Portlet may compute a changed navigationalState.	
getMarkup	sessionContext / sessionID	Producer provided sessionContext / sessionID	With Producer side state, the session handle offers ability to store information without impacts message size to Consumer.
	markupParameters / navigationalState	From link or from performInteraction.	

6.8 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`. Whether or not such a request is honored is up to the Consumer and often will depend on access control settings for the End-User.

A Portlet MUST support the `view` mode. During **getMarkup()**, **performInteraction()**, and **performBlockingInteraction()** invocations the Consumer indicates to the Portlet its current mode via the `MarkupParams` data structure.

6.8.1 “view” Mode

The expected functionality for a Portlet in `view` mode is to render markup reflecting the current state of the Portlet. The `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 `view` mode may depend on configuration, personalization and all forms of state.

All Portlets MUST support the `view` mode.

6.8.2 “edit” Mode

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

Typically, Portlets in `edit` mode will set or update Portlet state by making these changes persistent for the Portlet. How such changes impact Consumer management of Portlet usage by End-Users is discussed in section 6.3.3.

6.8.3 “help” Mode

When in `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.

6.8.4 “preview” Mode

In preview, a Portlet should provide a rendering of its standard `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.

6.8.5 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. A Consumer MUST NOT set a mode the Portlet’s description has not specified as supported. The Portlet SHOULD be prepared to 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 `view` mode. It is STRONGLY RECOMMENDED that custom mode values be URI's or prefixed in a manner designed to reduce name clashes with any values that may be defined by future versions of this specification.

6.9 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.

A Portlet MUST support the `normal` window state.

6.9.1 “normal” Window State

The `normal` window state indicates the Portlet is likely sharing the aggregated page with other Portlets. It MAY also indicate that the target device has limited display capabilities. Therefore, a Portlet SHOULD restrict the size of its rendered output in this window state.

All Portlets MUST support the `normal` window state.

6.9.2 “minimized” Window State

When the window state is `minimized`, the Portlet SHOULD NOT render visible markup, but is free to include non-visible data such as javascript or hidden forms. The Producer MUST expect the **getMarkup()** operation to be invoked for the `minimized` state just as for all other window states

6.9.3 “maximized” Window State

The `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 `maximized`.

6.9.4 “solo” Window State

The `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 `solo`.

6.9.5 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. A Consumer MUST NOT set a window state the Portlet’s description has not specified as supported. The Portlet SHOULD be prepared to 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 `normal`. It is STRONGLY RECOMMENDED that custom `windowState` values be URI's or prefixed in a manner designed to reduce name clashes with any values that may be defined by future versions of this specification.

6.10 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 5.1.11. 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.

6.10.1 User Category Assertions

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

1. 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.
2. 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. The Consumer controls the mechanism by which this mapping occurs.
3. 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.
4. 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 and the Consumer MUST NOT assert any user categories in the `UserContext` structure.

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 MAY offer out-of-band processes to register a Consumer. 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 5.1.20).

7.1 Data Structures

The normative definitions for all data structures are contained in the WSDL referenced in section 0. For the convenience of the reader, this non-normative section uses an IDL like syntax to describe these structures, where the leading [R] indicates a field is required and [O] indicates it is optional. The operations in this section introduce the following data structures:

Deleted: 14

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        customUserProfileData[]
    [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]. This string 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 MAY 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 the Consumer is willing to manage. This specification defines a set of constants for a base set of modes (see section 12). This array may reference both those constants and additional custom modes of the Consumer.
- `consumerWindowStates`: An array of window states the Consumer is willing to manage. This specification defines a set of constants for a base set of window states (see section 12). This array may reference both those constants and additional custom window states of the Consumer.
- `customUserProfileData`: An array of strings each of which name a `UserProfile` extension the Consumer supports.
- `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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

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]. 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 so that the Producer may release related resources. The Producer MAY end the registration by invalidating the registration identifier and MUST inform the Consumer of this through a fault message on the next invocation specifying this `registrationHandle` so that the Consumer may release related resources.

```
RegistrationContext = register(RegistrationData);
Faults: Interface.MissingParameters, Interface.OperationFailed
```

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 Consumer MAY register itself multiple times to a Producer with potentially different settings (e.g. security settings) resulting in multiple `registrationHandles` [R351]. Different registration contexts MUST be identified by different `registrationHandles`.

7.3 modifyRegistration() Operation

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

```
RegistrationState = modifyRegistration(RegistrationContext, RegistrationData);
Faults: Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed
```

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

7.4 deregister() Operation

The Consumer MUST NOT consider a relationship with a Producer ended until a successful invocation of **deregister()**.

```
ReturnAny = deregister(RegistrationContext);
Faults: Security.InvalidRegistration, Interface.OperationFailed
```

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 resources time out. In either case, a Consumer MUST NOT attempt to use the invalidated `registrationHandle` for subsequent invocations. An attempt to use an invalidated `registrationHandle` MUST result in a fault message indicating the registration is not valid. If the **deregister()** operation fails, the Producer MUST return a fault message specifying the reason for the failure.

8 Portlet Management Interface

Producers MUST expose one or more logically distinct ways of generating markup and handling interaction 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 MAY 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 **performInteraction()** invocations SHOULD support the **releasePortlet()** operation.

8.1 Data Structures

The normative definitions for all data structures are contained in the WSDL referenced in section 0. For the convenience of the reader, this non-normative section uses an IDL like syntax to describe these structures, where the leading [R] indicates a field is required and [O] indicates it is optional. The operations in this section introduce the following data structures:

Deleted: 14

8.1.1 DestroyFailed Type

The `DestroyFailed` structure contains a `portletHandle` that was not destroyed and the reason for the failure.

```
DestroyFailed
    [R] Handle   portletHandle
    [R] string   reason
```

Members:

- `portletHandle`: The `portletHandle` that was not destroyed.
- `reason`: A fault code from section 13 describing the reason the destroy failed.

8.1.2 DestroyPortletsResponse Type

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

```
DestroyPortletsResponse
    [O] DestroyFailed    destroyFailed[]
    [O] Extension        extensions[]
```

Members:

- `destroyFailed`: 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

8.1.3 PortletDescriptionResponse Type

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

```
PortletDescriptionResponse
    [O] 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`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

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.
- `resourceList`: This is an array of `Resource` structures, each of which carries the values for a localized resource in various locales.
- `extensions`: A mechanism implementations MAY choose to use for extending this structure provided those extensions come from a different namespace.

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, sendAllLocales);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle
```

Producers may choose to restrict 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 Security.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 and `sendAllLocales` to control whether values for all available locales are included.

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);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle
```

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. In the case of a Consumer Configured Portlet that pushes the Portlet's persistent 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 Producer chooses to push the persistent 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 MAY 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 MUST attempt to release the new `portletHandle` by invoking **destroyPortlets()** when it is no longer needed.

If the Producer's metadata declares registration is not supported (i.e. `requiresRegistration` flag was set to "false"), then the Consumer MUST invoke **destroyPortlets()** when it would have deregistered, passing each `portletHandle` that would have been scoped by a registration.

8.4 destroyPortlets() Operation

The Consumer MUST inform the Producer that a Consumer Configured Portlet will no longer be used by invoking **destroyPortlets()** and MUST NOT consider a Portlet to have been destroyed until **destroyPortlets()** has been successfully invoked for that Portlet.

```
DestroyPortletsResponse = destroyPortlets(RegistrationContext, portletHandles);
Faults: Security.InconsistentParameters, Security.InvalidRegistration,
        Interface.MissingParameters, Interface.OperationFailed
```

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 resources related to the `portletHandles` are immediately reclaimed or whether transient resources are allowed to timeout first. A Consumer MUST NOT reference any of the supplied `portletHandles` after successfully invoking **destroyPortlets()** and MAY reclaim resources related to the supplied `portletHandles` (e.g. `portletState`).

8.5 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. Each property declaration includes a name and type (default = `xsd:string`) [A505][A507]. This operation enables the Consumer to interact with this published portion of a Portlet's state.

```
PortletContext = setPortletProperties(RegistrationContext, PortletContext, UserContext,
                                      PropertyList);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle
```

Since **setPortletProperties()** is interacting only with the published portion of the Portlet's state, it MUST always be safe for the Portlet to modify its state (i.e. equivalent to `portletState` set to "`readWrite`" for a **performInteraction()** 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.

8.6 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);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle
```

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.

8.7 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.

```
PortletPropertyDescriptionResponse = getPortletPropertyDescription(
                                     RegistrationContext, PortletContext,
                                     UserContext, desiredLocales,
                                     sendAllLocales);
Faults: Security.AccessDenied, Security.InconsistentParameters,
        Security.InvalidRegistration, Interface.MissingParameters,
        Interface.OperationFailed, Interface.InvalidUserCategory,
        Interface.InvalidHandle
```

The `modelDescription` returned in the `portletPropertyDescriptionResponse` structure is a typed `property` view onto the portion of the Portlet's persistent state that the user (referenced through the `userContext`) is allowed to modify. While it is possible the set of properties MAY 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 and `sendAllLocales` to controls whether values for all available locales are included.

9 Security

Systems compliant with this specification will be exposed to the same security issues as other web service systems. For a representative summary of security concerns, refer to the Security and Privacy Considerations document produced by the XML-Based Security Services Oasis TC.

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. These standards generally fall into two main categories: document-level mechanisms and transport-level mechanisms.

The uses of document-level security mechanisms are not covered in this version of the specification since several important standards (in particular, security policy declarations) are not yet available. 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 important role in future versions of this specification.

For this version of the specification, emphasis is placed on using transport-level security standards (e.g. SSL/TLS) to address the security issues involved in Consumers invoking Producers on behalf of End-Users. These only require that a Producer's WSDL declare bindings 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.

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.

9.2 Confidentiality & Message Integrity

SSL/TLS may be used to ensure the contents of messages are neither tampered with nor decipherable by an unauthorized third party. 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 MAY 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 MAY implement access control mechanisms that restrict which End-Users may access which Portlets and operations on those Portlets. Additionally, a Producer MAY implement access control programmatically through the use of facilities such as an authenticated user identity.

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 MUST indicate to the Portlet the preferred character encoding, using the `characterSet` field of the `MarkupParams` structure. It is up to the Portlet to generate markup that complies with this encoding. The Portlet MAY generate markup in the UTF-8 character set encoding if it is unable to generate the requested character set. If it is unable to generate markup in either of these character sets, the Portlet MUST return a fault message to the Consumer. When the SOAP binding is in use, the Producer MUST use the same character set for the XML response message as was used to generate the markup.

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, 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:

- 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.
- 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 SHOULD seek to minimize this impact on performance by using efficient encoding and parsing mechanisms (for example, the Boyer-Moore algorithm[11]).

As there is no clear advantage to either technique, both styles of portlet URL encoding are supported (see sections 10.2.1 and 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:

1. If a Portlet's metadata declares it is willing to process URL templates, then the Consumer MUST supply templates for the Portlet to use.
2. If a Portlet is unable to completely write the portlet URLs for its markup, it MUST set the `needsUrlRewriting` flag in the returned `MarkupResponse` structure to "true".
3. If the `needsUrlRewriting` flag in the `MarkupResponse` structure is "true", then the Consumer MUST parse the returned markup and rewrite URLs conforming to the definitions in Section 9 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.

1. Represent each character in UTF-8 (see [RFC2279]) as one or more bytes.
2. 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).

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

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[12] to which the document carrying the URI may have been transcoded.

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 (preceded 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 what 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
```

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, 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 = action

Activation of the portlet URL will result in an invocation of **performInteraction()** on the Portlet that generated the markup. Prior to invoking this operation the Consumer MUST analyze the information encoded in the URL to determine if a mode and/or window state change is specified. The Consumer MUST process all mode and window state change requests and either honor or reject them prior to invoking the operation. All query string parameters not used to encode parameters defined by this specification MUST be passed to **performInteraction()** as `formParameters`. In addition the Consumer MUST check for the presence of the `wsrp-navigationalState` and `wsrp-interactionState` portlet URL parameters. If these parameters are present, their values MUST be supplied in the `navigationalState` and `interactionState` fields of the `MarkupParams` and `InteractionParams` structures, respectively. When these portlet URL parameters are not present, the Consumer MUST NOT supply the respective field.

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

10.2.1.1.2 wsrp-urlType = blockingAction

Activation of the URL will result in an invocation of **performBlockingInteraction()** on the Portlet that generated the markup. Prior to invoking this operation the Consumer MUST analyze the information encoded in the URL to determine if a mode and/or window state change is specified. The Consumer MUST process all mode and window state change requests and either honor or reject them prior to invoking the operation. All query string parameters not used to encode parameters defined by this specification MUST be passed to **performBlockingInteraction()** as `formParameters`. In addition the Consumer MUST check for the presence of the `wsrp-navigationalState` and `wsrp-interactionState` portlet URL parameters. If these parameters are present, their values MUST be supplied in the `navigationalState` and `interactionState` fields of the `MarkupParams` and `InteractionParams` structures, respectively. When these portlet URL parameters are not present, the Consumer MUST NOT supply in the respective field.

10.2.1.1.3 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 a `wsrp-navigationalState` portlet URL parameter, whose value the Consumer MUST supply in the `navigationalState` field of the `MarkupParams` structure. If there is no such portlet URL parameter, the Consumer MUST NOT supply a value for this field.

10.2.1.1.4 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, both the `wsrp-url` and `wsrp-rewriteResource` portlet URL parameters MUST also be specified.

10.2.1.1.4.1 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 likely use HTTP GET and it will have no base for use in fetching a relative URL. Also note that since this URL will appear as a parameter value, 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.

10.2.1.1.4.2 wsrp-rewriteResource

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. Consumers MAY want to process such resources in a manner that allows caching of the resulting resource by the End-User's user-agent. In particular, Consumers MAY 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.

10.2.1.1.5 wsrp-urlType = namespace

This tells the Consumer that the portlet "URL" contains a name that needs to be unique on the aggregated page (e.g. a form field's name or the name of a javascript method). While this is not technically a URL, providing this functionality in this manner limits the performance impacts of Consumer parsing to a single pass of the markup. The actual name that needs to be rewritten is encoded in the `wsrp-token` parameter which MUST be supplied whenever `wsrp-urlType` has a value of "namespace". See section 10.3 for more details on namespacing.

10.2.1.1.5.1 wsrp-token

This parameter provides the actual token that is to be namespaced.

10.2.1.2 wsrp-navigationalState

The value of this portlet URL parameter defines the 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 `navigationalState` field of the `MarkupParams`.

10.2.1.3 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.

10.2.1.4 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. This value MUST be one of the modes detailed in section 6.8 or a custom mode the Consumer specified as supported during registration. The Consumer MUST process this URL request to change the mode prior to invoking operations on the Portlet. If the requested mode change is for an invalid or unavailable mode (e.g. policy prohibits this End-User from entering that mode), the Consumer SHOULD leave the current value of the mode unchanged.

10.2.1.5 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. This value MUST be one of the values detailed in section 6.9 or a custom window state the Consumer specified as supported during registration. The Consumer MUST process this URL request to change the window state prior to invoking operations on the Portlet. If the requested window state change is for an invalid or unavailable window state (e.g. policy prohibits this End-User from entering that window state), the Consumer SHOULD leave the current value of the window state unchanged.

10.2.1.6 wsrp-fragmentID

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

10.2.1.7 wsrp-secureURL

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.

10.2.1.8 URL examples

Here are some examples of portlet URLs following this format:

- Load a resource http://test.com/images/test.gif:
  - wsrp-rewrite?wsrp-urlType=resource&wsrp-url=http%3A%2F%2Ftest.com%2Fimages%2Ftest.gif/wsrp-rewrite
- Declare this token as needing namespacing:
  - wsrp-rewrite?wsrp-urlType=namespace&wsrp-token=myFunc/wsrp-rewrite
- Declare a secure interaction back to the Portlet:
  - wsrp-rewrite?wsrp-urlType=action&wsrp-secureURL=true&wsrp-navigationalState=a8h4K5JD9&interactionState=fg4h923mdk/wsrp-rewrite
- Request the Consumer render the Portlet in a different mode and window state:
  - wsrp-rewrite?wsrp-urlType=render&wsrp-mode=help&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://www.Consumer.com/path/{wsrp-urlType}?mode={wsrp-mode}&…
```

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 that should be replaced. All content outside the {} pairs 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 10.2.2.11.

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 MUST supply these templates for Portlets specifying `doesUrlTemplatesProcessing` 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 actionTemplate

Activation of the portlet URL will result in an invocation of **performInteraction()**. The Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState`, `wsrp-interactionState`, `wsrp-mode` and `wsrp-windowState` in its template. The Consumer MUST process requests to change either mode or window state before invoking **performInteraction()**.

10.2.2.2 secureActionTemplate

`secureActionTemplate` is equivalent to `actionTemplate`, but using secure communication.

10.2.2.3 blockingActionTemplate

Activation of the URL will result in an invocation of **performBlockingInteraction()**. The Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState`, `wsrp-interactionState`, `wsrp-mode` and `wsrp-windowState` in its template. The Consumer MUST process requests to change either mode or window state before invoking **performBlockingInteraction()**.

10.2.2.4 secureBlockingActionTemplate

`secureBlockingActionTemplate` is equivalent to `blockingActionTemplate`, but using secure communication.

10.2.2.5 renderTemplate

Activation of the URL will result in an invocation of **getMarkup()**. The Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState`, `wsrp-mode` and `wsrp-windowState` in its template. The Consumer MUST apply changes resulting from requests to change either mode or window state before invoking **getMarkup()**.

10.2.2.6 secureRenderTemplate

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

10.2.2.7 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 SHOULD integrate placeholders for at least the portlet URL parameter `wsrp-url` to allow the Portlet to place the address of the URL.

10.2.2.8 secureResourceTemplate

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

10.2.2.9 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 an action oriented template, the Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState`, `wsrp-interactionState`, `wsrp-mode` and `wsrp-windowState` in this template.

10.2.2.10 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 an action oriented template, the Consumer SHOULD integrate placeholders for at least the portlet URL parameters `wsrp-navigationalState`, `wsrp-interactionState`, `wsrp-mode` and `wsrp-windowState` in this template.

10.2.2.11 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 any of these 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 name	Structure name	Field name
wsrp-portletHandle	`PortletContext`	`portletHandle`
wsrp-userContextKey	`UserContext`	`userContextKey`
wsrp-portletInstanceKey	`RuntimeContext`	`portletInstanceKey`
wsrp-sessionID	`RuntimeContext`	`sessionID`

10.2.3 BNF Description of URL formats

```
ConsumerURL = BeginToken UrlTypePair NameValuePairs EndToken
BeginToken = "wsrp-rewrite?"
EndToken = "/wsrp-rewrite"
UrlTypePair = "wsrp-urlType" "=" UrlType
UrlType = "action" | "blockingAction" | "render" | "resource" | "namespace"
NameValuePairs = ("&" NameValuePair)*
NameValuePair = TextPair | BooleanPair
TextPair = TextName "=" Text
TextName = "wsrp-navigationalState" | "wsrp-interactionState" | "wsrp-mode" |
"wsrp-windowState" | "wsrp-url" | "wsrp-token" | "wsrp-fragmentID"
Text = <any URL-encoded textual characters>
BooleanPair = BooleanName "=" BooleanValue
BooleanName = "wsrp-secureURL" | "wsrp-rewriteResource"
BooleanValue = ("true" | "false")

ProducerURLTemplate = (Text* ReplacementToken*)*
ReplacementToken = "{" ParameterName "}"
ParameterName = TextName | BooleanName | "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 MUST format their portlet URLs such that portlet URL activations are processed correctly, regardless of whether Consumer or Producer URL writing is in use.

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, etc. Such tokens SHOULD therefore be encoded to a Portlet-instance specific namespace [A301]. The Portlet MAY do this by prefixing the name of the resource with the `namespacePrefix` from the `RuntimeContext` structure.

If, during Consumer URL rewriting, the Consumer adds namespace prefixes to parameter names within the URL (note this does not include field names in HTML forms), then the Consumer MUST remove those prefixes before passing the parameter name to the Producer so that the Producer receives information in the manner it placed it in the markup. We would note that many things Producers and Portlets could namespace, such as field names in HTML forms, are not required to be unique and therefore are preferably not namespaced as this adds processing burden to generate and remove the namespacing and the Consumer will not do this removal for the Producer. This likely has an impact on client side code that may currently use such items to locate items on the page. In the case of field names within HTML forms, we would note that namespacing an id attribute and using it to locate the field provides the desired uniqueness within the aggregated environment without introducing a processing burden as would be caused if the name attribute were namespaced instead.

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

10.3.1 Consumer Rewriting

The Portlet uses the static, predefined method (section 10.2.1) to denote tokens that need a namespace prefix. The Consumer parses the markup fragment to locate these tokens and replace them with a namespaced token unique in the context of the page aggregation. This namespaced token MUST be the same for all occurrences of the token in the set of documents processed for one instance of the Portlet while assembling the aggregated page. This is done using the same method as Consumer URL rewriting and is described in section 10.2.1. It is expected that the length this method adds to names might make it unwieldy to content authors, but the expectation that tooling/runtime support can alleviate most of this burden led to reusing this singular technique for the Consumer parsing/rewriting the markup.

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 MUST ensure its markup 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 certified as being a cross-platform 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 Netscape Navigator 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>.

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 certified as being a cross-platform 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 Netscape Navigator 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>.

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 certified as being a cross-platform 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 Netscape Navigator 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>.

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. 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 these style definitions in order to participate in a uniform display of their content by various Consumers.

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

Example1: 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

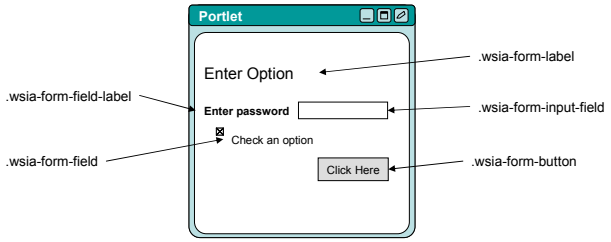
Section style definitions affect the rendering of markup sections such as table, div and span (i.e. alignment, borders, background color, etc.) as well as their text attributes.

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

10.5.5 Forms

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

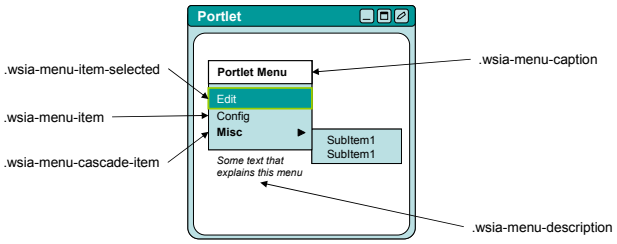
Style	Description
portlet-form-label	Text used for the descriptive label of the whole form (not the labels for fields.
portlet-form-input-field	Text of the user-input in an input field.
portlet-form-button	Text on a button
portlet-icon-label	Text that appears beside a context dependent action icon.
portlet-dlg-icon-label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



10.5.6 Menus

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

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



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 profileNames:

Profile Name	Structure 1	Structure 2	Field Name
name/prefix	name		prefix
name/given	name		given
name/family	name		family
name/middle	name		middle
name/suffix	name		suffix
name/nickname	name		nickname
birthDate			birthDate
gender			gender
employerInfo/employer	employerInfo		employer
employerInfo/department	employerInfo		department
employerInfo/jobtitle	employerInfo		jobtitle
homeInfo/address/name	homeInfo	address	name
homeInfo/address/street	homeInfo	address	street
homeInfo/address/city	homeInfo	address	city
homeInfo/address/stateprov	homeInfo	address	stateprov
homeInfo/address/postalcode	homeInfo	address	postalcode
homeInfo/address/country	homeInfo	address	country
homeInfo/address/organization	homeInfo	address	organization
homeInfo/telephone	homeInfo		telephone
homeInfo/email	homeInfo		email
homeInfo/online	homeInfo		online
workInfo/address/name	workInfo	address	name
workInfo/address/street	workInfo	address	street
workInfo/address/city	workInfo	address	city
workInfo/address/stateprov	workInfo	address	stateprov

workInfo/address/postalcode	workInfo	address	postalcode
workInfo/address/country	workInfo	address	country
workInfo/address/organization	workInfo	address	organization
workInfo/telephone	workInfo		telephone
workInfo/email	workInfo		email
workInfo/online	workInfo		online

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

Consumers supplying additional custom profile fields are encourage to publish a similar mapping between profileNames and the custom fields.

11.1 Passing User Information

User information MAY be passed 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 services 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.

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 MAY 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).

12 Constants

Type	Value	Description
Mode	view	Portlet is expected to render markup reflecting its current state.
Mode	edit	Portlet is expected to render markup useful for End-User customization.
Mode	help	Portlet is expected to render markup useful for helping an End-User understand the Portlet's operation.
Mode	preview	Portlet is expected to render markup representative of its configuration, as this might be useful to someone testing a page layout.
Window state	normal	The Portlet is sharing space with other Portlets and should restrict its consumption of space accordingly.
Window state	minimized	The Portlet, though still aggregated on the page, is expected to restrict its consumption of space to a bare minimum.
Window state	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	solo	The Portlet is the only Portlet being rendered on the Consumer's aggregated page.

13 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. WSDL defines fault codes to be strings using "." as a delimiter to scope the error codes. The following are defined for constructing a WSRP error code within the same namespace as used for the rest of the types defined by this specification:

Top Level	Specific Code	Description
Security	AccessDenied	Policy has denied access. This may be related to the Consumer's registration.
Security	InconsistentParameters	Used when a Consumer supplies a `portletHandle` that is not scoped by the supplied `registrationHandle`.
Security	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.
Interface	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.
Interface	InvalidHandle	Used when the Consumer supplies an invalid `Handle`.
Interface	InvalidSession	Used only when a Producer session has timed out AND the Producer needs the Consumer to invoke resend data that may have been cached in the session.
Interface	InvalidUserCategory	The specified `UserCategory` is not supported.
Interface	MissingParameters	Used when required parameters are missing.
Interface	OperationFailed	Normal execution of the operation failed. Check the detailed message for reasons why.
Interface	PortletStateChangeRequired	Used when a Portlet needs to modify its persistent state, but has been prevented from doing so.
Interface	UnsupportedLocale	The Portlet does not support generating markup for the requested locale.
Interface	UnsupportedMarkupType	The Portlet does not support generating markup for the requested markupType.
Interface	UnsupportedMode	The Portlet does not support generating markup for the requested mode.
Interface	UnsupportedWindowState	The Portlet does not support generating markup for the requested window state.

14 WSDL Interface Definition

The WSDL that MUST be referenced by Producers implementing this specification are located at:

http://www.oasis-open.org/committees/wsrp/v1/WSRP_v1_Types.xsd - The type, message and fault definitions for this specification.

http://www.oasis-open.org/committees/wsrp/v1/WSRP_v1_Interfaces.wsdl - The portType definitions for this specification.

http://www.oasis-open.org/committees/wsrp/v1/WSRP_v1_Bindings.wsdl - The standard binding definitions for this specification.

This WSDL defines the following portTypes:

1. WSRP_v1_Markup_PortType: All Producers MUST expose this portType.
2. WSRP_v1_ServiceDescription_PortType: All Producers MUST expose this portType.
3. WSRP_v1_Registration_PortType: Only Producers supporting in-band registration of Consumers need expose this portType.
4. WSRP_v1_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:

5. WSRP_v1_Markup_Binding_SOAP: All Producers MUST expose this binding for the `Markup` portType.
6. WSRP_v1_ServiceDescription_Binding_SOAP: All Producers MUST expose this binding for the `ServiceDescription` portType.
7. WSRP_v1_Registration_Binding_SOAP: Producers supporting the `Registration` portType MUST expose this binding to that portType.
8. WSRP_v1_PortletManagement_Binding_SOAP: Producers supporting the `PortletManagement` portType MUST expose this binding to that portType.

15 References

15.1 Normative

[Character Sets] *http://www.iana.org/assignments/character-sets*

[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/*

[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*

15.2 Non-Normative

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

[DIME] *http://www.ietf.org/internet-drafts/draft-nielsen-dime-02.txt*

[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/*

[Requirements] *http://www.oasis-open.org/committees/wsia/documents/Requirements2002-09-17.html*

[RLTC] *http://www.oasis-open.org/committees/rights/*

[SAML] *https://www.oasis-open.org/committees/security/*

[UDDI] *http://www.uddi.org/specification.html*

[WS-Attachments] *http://www-106.ibm.com/developerworks/webservices/library/ws-attach.html*

[WS-I.org] *http://www.ws-i.org/*

[WSIL] *http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html*

[WSRP Whitepaper] *Thomas Schaeck,* Web Services for Remote Portals (WSRP) Whitepaper, *http://www.oasis-open.org/committees/wsrp/documents/wsrp_wp_09_22_2002.pdf, 22 September, 2002.*

[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
Portal Page	Complete document rendered by a portal.
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
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 Registration Properties

Interest has been expressed in developing a non-normative list of common property names with well defined semantics. The following table is a beginning of such a list. Those looking to contribute additional names may send an email to the wsrp-comment@lists.oasis-open.org list with a subject line of "Registration Property Proposal". The email should specify the property name, type, access and semantics.

Name	Type	Access	Semantics
producerAgent	String	Read only	Equivalent to `consumerAgent` from `RegistrationData`. Specifies the Producers' software and uses the same format, i.e. starts with "productName.majorVersion.minorVersion".

B.2 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. These definitions suggest the Portlet generates progressively more restrictive content and are provided as guidelines only. The specific semantics of these categories are left to each Portlet's implementation.

full: The content for this user category will typically encompass the full functionality of the Portlet.

standard: This user category is typically associated with End-Users who may customize some set of properties for a Portlet.

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. Data Structures List (Non-Normative)

The following lists the data structures this specification defines including the section numbers for the defintions.

BlockingInteractionResponse (6.1.14)
CacheControl (6.1.5)
ClientData (6.1.7)
CookieProtocol (5.1.17)
DestroyFailed (8.1.1)
DestroyPortletsResponse (8.1.2)
Extension (5.1.1)
Handle (5.1.2)
ID (5.1.4)
InteractionParams (6.1.17)
InteractionResponse (6.1.12)
Key (5.1.3)
LocalizedString (5.1.5)
MarkupContext (6.1.11)
MarkupParams (6.1.9)
MarkupResponse ()
MarkupType (5.1.10)
ModelDescription (5.1.16)
NamedString (6.1.8)
PortletContext (6.1.3)
PortletDescription (5.1.11)
PortletDescriptionResponse (8.1.3)
PortletPropertyDescriptionResponse (8.1.4)
Property (5.1.12)
PropertyDescription (5.1.15)
PropertyList (5.1.14)
RegistrationContext (5.1.20)
RegistrationState (5.1.19)
RegistrationData (7.1.1)
ResetProperty (5.1.13)
Resource (5.1.7)
ResourceList (5.1.8)
ResourceValue (5.1.6)
RuntimeContext (6.1.2)
ServiceDescription (5.1.18)
SessionContext (6.1.1)
StateChange (6.1.15)
Templates (6.1.6)
UpdateResponse (6.1.13)
UploadContext (6.1.16)
UserCategoryDescription (5.1.9)
UserContext (6.1.19)
UserProfile (6.1.18)
- UserName (6.1.18.1)
- EmployerInfo (6.1.18.2)
- LocationInfo (6.1.18.3)
- Address (6.1.18.4)
UserScope (6.1.4)

Appendix D. Acknowledgments (Non-Normative)

D.1. WSIA committee members

The following individuals were members of the WSIA committee during the development of this specification (* indicates voting member when specification was finalized):

- Sasha Aickin*, Plumtree Software
- Patel Ashish, France Telecom
- Stefan Beck, SAP
- Dan Bongard, Kinzan
- Kevin Brinkley, Intel
- Jeffery C. Broberg*, Novell
- Rex Brooks*, Individual
- Tyson Chihaya, Netegrity
- Carlos Chue, Kinzan
- Terry Cline, Peregrine Systems
- William Cox*, BEA Systems Inc
- Suresh Damodaran, Sterling Commerce
- Alan Davies, SeeBeyond
- Jacques Durand, Fujitsu
- John Evdemon, Vitria
- Sean Fitts, CrossWeave
- Greg Giles, Cisco
- Dan Gisolfi*, IBM Corporation
- Timothy N. Jones, CrossWeave
- Aditi Karandikar, France Telecom
- John Kelley, Individual
- John Kneiling, Individual
- Ravi Konuru*, IBM Corporation
- Alan Kropp*, Vignette Corporation
- Bruce Lucas*, IBM Corporation
- Eric van Lydegraf*, Kinzan
- Michael Mahan, Nokia
- Monica Martin*, Drake Certivo
- Dale Moberg, Cyclone Commerce
- Dean Moses, Epicentric
- Peter Quintas, Divine
- T.V. Raman, IBM Corporation
- Shankar Ramaswamy, IBM Corporation
- Eilon Reshef*, WebCollage
- Graeme Riddell*, Bowstreet
- Don Robertson, Documentum
- Royston Sellman, HP
- Sim Simeonov, Macromedia
- Davanum Srinivas, Computer Associates
- Sandra Swearingen, DoD
- Gil Tayar*, WebCollage
- Rich Thompson*, IBM Corporation
- Srinivas Vadhri*, Commerce One
- Vinod Viswanathan, Pointgain Corp.
- Charles Wiecha, IBM Corporation (chair)
- Chad Williams, Epicentric
- Kirk Wilson*, Computer Associates
- Garland Wong, Kinzan

D.2. WSRP committee members

The following individuals were members of the WSRP committee during the development of this specification (* indicates voting member when specification was finalized):

- Alejandro Abdelnur*, Sun Microsystems
- Sasha Aickin*, Plumtree Software
- Subbu Allamaraju*, BEA SYSTEMS INC
- Olin Atkinson*, Novell
- Amir Blich*, SAP
- Chris Braun*, Novell
- Jeff Broberg, Novell
- Rex Brooks*, Individual
- Mark Cassidy, Netegrity
- Dave Clegg, Sybase
- Ugo Corda, SeeBeyond
- T.J. Cox*, Novell
- William Cox*, BEA SYSTEMS INC
- Michael C. Daconta, McDonald Bradley
- Ron Daniel Jr., Interwoven
- Angel Luis Diaz, IBM Corporation
- Brian Dirking*, Stellent
- Jane Dynin, Plumtree Software
- Gino Filicetti*, Bowstreet
- Adrian Fletcher, BEA SYSTEMS INC
- Michael Freedman*, Oracle Corporation
- Ross Fubuni*, Plumtree Software
- Tim Granshaw, SAP
- Mike Hillerman, Peoplesoft
- Scott Huddleston, Divine
- Richard Jacob*, IBM Corporation
- Timothy N. Jones, CrossWeave
- Aditi Karandikar, France Telecom
- Jon Klein*, Reed Elsevier
- Andre Kramer*, Citrix Systems, Inc
- Alan Kropp*, Vignette Corporation
- Andreas Kuehne, Individual
- Carsten Leue*, IBM Corporation
- Susan Levine, Peoplesoft
- Eric van Lydegraf*, Kinzan
- Dan Machak*, TIBCO Software Inc.
- Monica Martin*, Drake Certivo
- Khurram Mahmood, Peoplesoft
- Lothar Merk, IBM Corporation
- Madoka Mitsuoka*, Fujitsu
- Takao Mohri, Fujitsu
- Adam Nolan*, Reed Elsevier
- Petr Palas*, Moravia IT
- Gregory Pavlik, HP
- Peter J Quintas*, Divine
- Raj Ramesh*, CommerceOne
- Sunit Randhawa*, Fujitsu
- Nigel Ratcliffe*, Factiva
- Eilon Reshef*, WebCollage
- Mark Rosenberg, TIBCO Software Inc.
- Joe Rudnicki*, U.S. Department of the Navy
- Thomas Schaeck*, IBM Corporation (chair)
- Robert Serr, Divine
- Gennady Shumakher*, SAP
- Steven Smith*, Capitol College
- Davanum Srinivas, Computer Associates
- Joseph Stanko, Plumtree Software
- Andrew Sweet, Perficient
- David Taieb, IBM Corporation
- Yossi Tamari*, SAP
- Gil Tayar*, WebCollage
- Rich Thompson*, IBM Corporation
- Srinivas Vadhri, CommerceOne
- Stephen A. White, SeeBeyond
- Charles Wiecha*, IBM Corporation
- Michael Young*, Plumtree Software

Formatted: Font: Bold
Deleted: 0.91
Deleted: 2/3/2003

Appendix E. Revision History (Non-Normative)

Rev	Date	By Whom	What
0.1	6/03/2002	Rich Thompson	Initial Draft
0.1.1	6/04/2002	Carsten Leue	Worked in some additional WSRP requirements
	6/05/2002	Rich Thompson	Added exemplary section to overview
	6/06/2002	Carsten Leue	Added request data to getFragment and invokeAction
0.1.2	6/06/2002	Rich Thompson	Added cloneEntities() & descriptive text
0.2	7/09/2002	Alan Kropp, Rich Thompson	Modified as per face-2-face discussions
0.21	7/10/2002	Rich Thompson	Refactored data objects
0.22	7/19/2002	Rich Thompson	Reflect discussion on email list
0.23	7/25/2002	Carsten Leue Rich Thompson	Added WSDL and included some explanations Reformat style Reflect discussion
0.3	8/01/2002	Rich Thompson	Migrate to OASIS spec template Reflect email list and concall discussions
0.31	8/08/2002	Rich Thompson Alan Kropp Chris Braun	Reflect discussion Fill out more of spec template Markup section Environment initialization section
0.32	8/10/2002	Rich Thompson Carsten Leue Chris Braun	Incorporated misc. comments/discussion Introduction section and explanation of sections Updated Markup section
0.4	8/16/2002	Rich Thompson Thomas Schaeck Alan Kropp	Rewrote Markup section, reflect discussion WSRP Use cases Cross references to requirements
0.5	8/30/2002	Rich Thompson Carsten Leue Mark Cassidy	Incorporated misc. comments/discussion UDDI, Additional data structure factoring Updated Security section
0.7	9/27/2002	Rich Thompson Carsten Leue Alan Kropp Charlie Wiecha	Reflect Sept. F2F discussion/decisions Rewrite Intro Propose caching support Propose Properties support
0.8	10/22/2002	Rich Thompson Carsten Leue Alan Kropp	Reflect decisions from weekly TC calls
0.85	12/10/2002	Rich Thompson Alan Kropp	Reflect decisions from Nov. F2F and TC calls
0.90	01/17/2003	Rich Thompson	Reflect issue resolutions
0.91	02/03/2003	Rich Thompson	Reflect TC decision through Jan ’03 F2F

Appendix F. Notices

Deleted: WSRP_Specification-v0.91