



WSIA - WSRP Core Specification

Working Draft 0.7, 30 September 2002

Document identifier:

5 Draft-WSIA-WSRP_Core_Interface-v0.56 ([Word](#))

Location:

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

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

Editors:

10 Alan Kropp, Epicentric, Inc. <akropp@epicentric.com>
Carsten Leue, IBM Corporation <cleue@de.ibm.com>
Rich Thompson, IBM Corporation <richt2@us.ibm.com>

Contributors:

15 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>
20 Thomas Schaeck, IBM Corporation <schaeck@de.ibm.com>
Gil Tayar, WebCollage <Gil.Tayar@webcollage.com>

Abstract:

25 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, had to write special adapters for applications and content providers to accommodate the variety of different interfaces and protocols those providers used. 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.

30 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 services as they become available. The joint authoring of these interfaces by WSRP and WSIA allows maximum reuse of user facing, interactive web
35 services while allowing the consuming applications to access a much richer set of standardized web services.

5 This joint standard layers on top of the existing web services stack, utilizing existing web services standards and will leverage the emerging web services standards (such as security) as they become available. The interfaces are defined using the Web Services Description Language (WSDL). Metadata and conventions for publishing services to UDDI directories and finding them in UDDI directories are also defined.

Status:

10 This draft is still at the level of an expert group spec, though it is rapidly moving toward being a public spec. Various concepts continue to be debated. Points needing clarification as this evolves into the final specification are much appreciated and may be emailed to *Rich Thompson*.

15 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/xxx/yyy>.

Copyright © 2001, 2002 The Organization for the Advancement of Structured Information Standards [OASIS]

20

Table of Contents

1	Introduction	7
	1.1 Motivation	7
	1.2 Components	8
5	1.2.1 Producer	8
	1.2.2 Consumer	9
	1.2.3 Interaction	9
	1.3 Example Scenarios	9
	1.3.1 SimpleProducer	9
10	1.3.2 SophisticatedProducer	9
	1.3.3 SimpleConsumer	9
	1.3.4 SophisticatedConsumer	10
	1.3.5 Interaction between levels of sophistication	10
	2 Terminology	15
15	3 General Interface Design Issues	15
	3.1 Related Standards	15
	3.1.1 Existing Standards	15
	3.1.2 Emerging Standards	15
	3.2 Data Objects	16
20	3.3 Lifecycles	16
	3.4 Scopes	16
	3.5 Types of Stateful Information	17
	3.6 Persistence and statefulness	17
	3.7 Sessions	18
25	3.8 Producer Mediated Sharing	18
	3.9 Information Passing Mechanisms	18
	3.10 Event Handling	18
	3.11 Two-phase protocol	19
	3.12 Interaction Lifecycle States	19
30	3.12.1 Assumptions:	19
	3.12.2 State 0: Unknown	19
	3.12.3 State 1: Known	19
	3.12.4 State 2: Active	20
	3.13 Transport Issues	20
35	4 Service Description Interface	20
	4.1 New Data Structures	20
	4.1.1 ServiceDescription	20
	4.1.2 UserContext	21
	4.1.3 RegistrationCore	21
40	4.1.4 RegistrationContext	22
	4.2 getServiceDescription() Operation	22

5	Registration Interface	22
	5.1 New Data Structures	22
	5.1.1 RegistrationData	23
	5.2 register() Operation	23
5	5.3 modifyRegistration() Operation	24
	5.4 deregister() Operation	24
6	Entity Management Interface	24
	6.1 New Data Structures	25
	6.1.1 refHandle	25
10	6.1.2 EntityDescription	25
	6.1.3 EntityHandle	25
	6.1.4 RefHandleContext	25
	6.1.5 EntityContext	26
	6.1.6 EntityResponse	26
15	6.1.7 InteractionParams	27
	6.1.8 InteractionResponse	27
	6.1.9 properties	28
	6.1.10 PropertyDescription	28
	6.1.11 PropertyDescriptionList	28
20	6.2 getEntityDescription() Operation	29
	6.3 cloneEntity() Operation	29
	6.4 destroyEntities() Operation	29
	6.5 setEntityProperties() Operation	30
	6.6 getEntityProperties() Operation	30
25	6.7 getEntityPropertyDescription() Operation	30
7	Markup Interface	31
	7.1 New Data Structures	31
	7.1.1 groupID	31
	7.1.2 CacheControl	31
30	7.1.3 MarkupParams	32
	7.1.4 MarkupResponse	33
	7.1.5 Templates	34
	7.2 getMarkup() Operation	35
	7.2.1 Caching of markup fragments	35
35	7.3 performInteraction() Operation	36
	7.3.1 Updating Persistent Entity State	36
	7.4 initEnvironment() Operation	37
	7.4.1 Load Balancing	37
	7.4.2 Consumer Transitions across Bindings	38
40	7.5 Stateful Entity Scenarios	38
	7.5.1 No State	38
	7.5.2 Navigational State Only	39
	7.5.3 Local state	39
	7.6 Modes	40

	7.6.1	VIEW Mode	40
	7.6.2	EDIT Mode	41
	7.6.3	HELP Mode	41
	7.6.4	CONFIG Mode	41
5	7.6.5	PREVIEW Mode	41
	7.6.6	PREVIOUS Mode	41
	7.6.7	Custom Modes	41
	7.7	Window States	41
	7.7.1	NORMAL Window State	42
10	7.7.2	MINIMIZED Window State	42
	7.7.3	MAXIMIZED Window State	42
	7.7.4	PREVIOUS Window State	42
	7.7.5	Custom Window States	42
8		Security	42
15	8.1	Authentication of Consumer	43
	8.2	Confidentiality & Message Integrity	43
	8.3	Access control	43
	8.4	Roles	43
	8.4.1	Role Assertions	44
20	8.4.2	Standard Roles	44
9		Markup	44
	9.1	Encoding	44
	9.2	URL Considerations	44
	9.2.1	Consumer URL Writing	46
25	9.2.2	Producer URL Writing	48
	9.2.3	BNF-like Description of URL formats	49
	9.3	Namespace Encoding	50
	9.3.1	Consumer Rewriting	50
	9.3.2	Producer Writing	50
30	9.3.3	General issues	50
	9.4	Markup Fragment Rules	51
	9.4.1	HTML	51
	9.4.2	XHTML	52
	9.4.3	XHTML Basic	52
35	9.5	CSS Style Definitions	52
	9.5.1	Links (Anchor)	53
	9.5.2	Fonts	53
	9.5.3	Messages	53
	9.5.4	Sections	54
40	9.5.5	Forms	54
	9.5.6	Menus	55
10		User Information	55
	10.1	Passing User Information	56
	10.2	User Identity	57

11	Data Structures	57
	11.1 Handle Types	57
	11.2 Property Type	58
	11.3 PropertyDescription Type	58
5	11.4 RegistrationData Type	58
	11.5 RegistrationCore Type	58
	11.6 RegistrationContext Type	58
	11.7 UserContext Type	58
	11.8 ServiceDescription Type	58
10	11.9 EntityDescription Type	58
	11.10 EntityContext Type	58
	11.11 EntityResponse Type	58
	11.12 MarkupParams Type	58
	11.13 MarkupResponse Type	59
15	11.14 InteractionParams Type	59
	11.15 InteractionResponse Type	59
	11.16 CacheControl Type	59
	11.17 Templates Type	59
	11.18 ClientData Type	59
20	11.19 User Profile Type	59
	11.20 UserName Type	60
	11.21 EmployerInfo Type	60
	11.22 LocationInfo Type	60
	11.23 Address Type	61
25	11.24 LocaleData Type	61
	11.25 EntityType Type	62
	11.26 RoleDescription Type	63
	12 Producer Roles	64
	13 Constants	65
30	14 Fault Messages	66
	15 WSDL Interface Definition	66
	16 References	67
	16.1 Normative	67
	16.2 Non-Normative	67
35	Appendix A. Glossary	68
	Appendix B. Acknowledgments	73
	Appendix C. Revision History	75
	Appendix D. Notices	76

40

1 Introduction

Both Web Services for Interactive Applications (WSIA) and Web Services for Remote Portals (WSRP) define a web service interface for accessing and interacting with user-facing, interactive presentation oriented web services.

5

This specification defines the joint WSIA/WSRP interfaces. It is based on the requirements gathered by both committees and on the concrete proposals to both committees.

WSRP/WSIA functionality is motivated by scenarios including:

10

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

15

However this description also applies to non-portal environments, mostly identified by the [WSIA use cases](#). For a detailed overview of Web Services, Portal Environments and the application of WSRP to these environments please refer to the [\[WSRP Whitepaper\]](#) and additional documents at <http://www.oasis-open.org/committees/wsrp/>.

20

The standard accounts for the fact that Producers and Consumers may be implemented on very different platforms, be it as a Java/[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 language independence.

25

These services are built on standard technologies including [\[WSDL\]](#), [\[SOAP\]](#) and will leverage future applicable Web Service standards, such as [WS-Security and WS-Policy \[A102\]](#).

1.1 Motivation

30

Portals render and aggregate information from different sources and provide it in a compact and easily consumable form to an End-User. Typically, this information consists of markup fragments that are surrounded by a decoration containing controls. The whole construct is commonly referred to as a “portlet” and the content as “markup” or “markup fragment”.

35

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

40

This specification solves this problem by introducing a web service interface that allows the inclusion and interaction with content from a web service. Such a service provides both application logic and presentation logic and corresponds to the remote presentation paradigm well known from distributed systems. It provides a common protocol and a set of interfaces for all user-facing web services. Thus portals can easily adopt these services, i.e. by utilizing generic proxy code.

1.2 Components

5 This protocol describes the conversation between Producers (web service providers) and Consumers (web service requestors). Producers provide user-facing web services that are able to render markup fragments and process user-interaction requests. Consumers use these services to present the generated markup to End-Users and manage the user's interaction with the markup.

1.2.1 Producer

10 Producers are modeled as containers of the actual content generators (e.g. portlets from the portal scenario), which are called entities by this specification. Functionally, the container provides a set of services including:

- *self description*: Allows Consumers to find out the capabilities of the Producer and about the entities it hosts.
- *registration*: Is used to establish a [trust] relationship between a Producer and a Consumer (e.g. for billing or book keeping purposes).
- 15 • *entity management*: Grants access to the life cycle of the hosted entities.

In order to allow different levels of sophistication for both the Producer and Consumer, parts of this functionality are optional. A Producer must expose at least one entity. The following functionality refers to the communication with entities:

- 20 • *markup and interaction related interfaces* are used to render and interact with markup fragments.
- *property management* enables programmatic access to an entity's persistent state.

25 Entities exposed by the Producer are declared via the *self description* interface. The exposed metadata tells Consumers how to interact with the entity.

A particular entity is identified with an `entityHandle`. The Consumer uses `entityHandles` throughout the communication to address and interact with entities via the Producer. Entities a Producer publishes as available for all Consumers to interact with are called "Producer Offered Entities". Producer Offered Entities are pre-configured and not modifiable by Consumers. An example of a Producer Offered Entity would be a flight schedule service that shows a table of scheduled flights between two airports..

35 If the Producer chooses to expose the *entity management* interface, it is allowing Consumers to clone and customize the entities offered by the Producer. Such a uniquely configured entity is called "Consumer Configured Entity". Like Producer Offered Entities, Consumer Configured Entities are addressed by an `entityHandle` that is generated when they are cloned. A stock quote entity that offers the possibility to select/configure different ticker symbols to display is an example of such an entity.

40 Besides entity management, the Producer also manages Consumer *registrations*. The Producer may require Consumers to register prior to discovering and interacting with entities. A registration represents a relationship (often including both trust and business aspects) between the Consumer and Producer.

1.2.2 Consumer

5 A Consumer is an intermediary system that communicates with user-facing web services (i.e. Producers and the entities they host) on behalf of its users. It gathers and aggregates the markup delivered by the entities 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.

1.2.3 Interaction

10 The main aspect of user-facing web services is offering an End-User interface in form of markup and then processing interactions with that markup. The markup related interface defines two operations **getMarkup()** and **processInteraction()** for this purpose:

- **getMarkup()** is called to obtain the markup fragments from an entity. The markup returned depends on various things such as the entity's current state, the user context, the markup type requested and so on.
- 15 • **performInteraction()** is called when an End-User interacts with the markup from the entity. This interaction may result in a state change of the entity which often causes changes in the markup returned on a subsequent **getMarkup()** call.

1.3 Example Scenarios

20 This specification supports Consumers and Producers of various levels of sophistication interacting with one another. While not exhaustive, the following cases and how they can interact with each other are explored as examples of the broad range of possibilities.

1.3.1 SimpleProducer

Does not support registration or persistence. May only offer 1 type of entity.

Examples:

- 25 • A flight schedule display that is publicly available. Neither user registration nor persistent state is required. The service may however maintain interaction state using a session.
- News feed service that allows the user to browse news topics.

1.3.2 SophisticatedProducer

30 Requires Consumers to register and the returned reference is required for all future invocations. Provides metadata relevant for interacting with the service. Supports a number of entities, which publish metadata declaring supported markup types and properties for interacting with the entity.

Examples:

- 35 • Portal server that exposes portlets available through a compliant service endpoint. Each portlet may contain End-User specific settings and information that are persisted on the portal server.

1.3.3 SimpleConsumer

40 Does not persist any registration/entity information across cycling of the Consumer. Have explicit declarations for binding to and interacting with a set of Producer services.

1.3.4 SophisticatedConsumer

Supports persisting Producer, Consumer and End-User related data. Supports single sign on for its End-Users (may require End-User to trust Consumer with sign-on data). May support discovery of new Producers by either Administrators and/or End-Users.

5 Examples:

- Typical portal server that access WSIA/WSRP services for content aggregated onto pages.
-

10 1.3.5 Interaction between levels of sophistication

The following illustrate how the interaction between the various parties (End-Users, a Consumer and a Producer) **might** flow for each combination of these examples.

1.3.5.1 Sophisticated Consumer / Sophisticated Producer (stateful +configurable)

15

User Agent		Consumer		Producer
		Consumer Administrator registers	→ registerConsumer()	Supports registration
		Consumer stores the registration (cc) for use later.	← registratoinContext	Returns new handle in registrationContext
		Page Designer looks up Producer Offered Entites	→ getDescription()	Validates Consumer registration
		Display discovered capabilities	← description	Returns full description of service/entities
		Page Designer clones an entity	→ CloneEntity()	Clones entity from supplied reference
		Adds new Entity into the Designer's toolbox	← entityContext	Applies changes and returns new handle in entityContext
		Page Designer adds Entity to page		
User directs browser to the Consumer's URL	→ http get	Get markup from Entity	→ GetMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← Markup	Return generated markup
User interacts with	→	Consumer	→	Referenced entity

the markup	http post	determines correct Producer/entity	performInteraction()	handles logical interaction
		Store navigationalState for use on a page refresh	← navigationalState	Returns modified navigational state
		Request new markup	→ GetMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← Markup	Return generated markup
		Consumer Administrator deregisters	→ releaseHandles()	Producer releases resources

Sophisticated Consumer / Simple Producer (stateful)

User Agent		Consumer		Producer
		Page Designer looks up Producer Offered Entites	→ getDescription()	Validates Consumer registration
		Display discovered capabilities	← description	Returns full description of service/entities
		Page Designer looks up Producer Offered Entites ... clones one	→ cloneEntity()	Clones a new entity from supplied reference
		Adds new Entity into the Designer's toolbox	← entityContext	Applies changes and returns new handle in entityContext
		Page Designer adds Entity to page		
User directs browser to the Consumer's URL	→ http get	Get markup from Entity	→ getMarkup()	Request handled by the referenced entity
Render returned page	← Page	Wrap markup into a valid page	← markup	Return generated markup
User interacts with the markup	→ http post	Consumer determines correct Producer/entity	→ performInteraction()	Referenced entity handles logical interaction
		Store navigationalState for use on a page refresh	← navigationalState	Returns modified navigational state
		Request new markup	→ getMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← markup	Return generated markup
		Consumer Administrator deregisters	→ releaseHandles()	Producer releases resources

1.3.5.2 Simple Consumer / Simple Producer (no state)

User Agent		Consumer		Producer
User directs browser to the Consumer's URL		Get markup from a Producer Offered Entity	→ getMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← markup	Return generated markup

1.3.5.3 Simple Consumer / Simple Producer (with state)

5

User Agent		Consumer		Producer
User directs browser to the Consumer's URL		Get markup from a Producer Offered Entity	→ getMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← markup	Return generated markup
User interacts with the markup	→ http post	Consumer determines correct Producer/entity	→ performInteraction()	Referenced entity handles logical interaction
		Store navigationalState for use on a page refresh	← navigationalState	Returns modified navigational state
		Request new markup	→ getMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← markup	Return generated markup

1.3.5.4 Simple Consumer / Sophisticated Producer (stateful +configurable)

User Agent		Consumer		Producer
User directs browser to the Consumer's URL	→ http get	Consumer attempts to register	→ registerConsumer()	Supports registration
		Consumer stores the registrationContext for use later.	← registrationContext	Returns new registrationHandle in RegistrationContext
		Get markup from a Producer Offered Entity	→ getMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← markup	Return generated markup
User interacts with the markup	→ http post	Consumer determines correct Producer/entity	→ performInteraction()	Referenced entity handles logical interaction
		Store navigationalState for use on a page refresh	← navigationalState	Returns modified navigational state
		Request new markup	→ getMarkup()	Request handled by the referenced entity
Render returned page	← page	Wrap markup into a valid page	← markup	Return generated markup
User stops interacting	→ Session w/Consumer times out	Consumer deregisters	→ releaseHandles()	Producer releases resources

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]**.

5 **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.

10 **Conformance:** Not mandatory – ISO/IEC Guide 2 defines conformance or conformity as fulfillment of a product, process or service of specified requirements. Note that many times providers use “comply” to a standard to sidestep because they don’t actually “conform” to a standard because either the standard is not approved yet OR the product does not actually conform.

15 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 Interface Design Issues

The major design goals of this specification are simplicity, extensibility and efficiency.

20 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.

25 [SOAP](#) – Defines how to invoke remote interfaces.

[UDDI](#) – Defines how web services are published, queried and found using standardized directories.

[SSL/TLS](#) – Defines secure transport mechanisms.

[URL](#) – Defines URI (includes URL) syntax and encoding

30 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.

[P3P](#) – Defines how a Producer/entity may publish its privacy policy so that a Consumer could enforce End-User privacy preferences.

35 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.

[XACML](#) – Defines a syntax for expressing authorization rules.

[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.

5 [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 Data Objects

10 It is often necessary to pass data to operations. Wherever possible typed data objects are defined as the transport mechanism. Property arrays are included in each data structure for vendor or application specific data extensions. Producers/entities employing these extensions SHOULD provide typing information for the extended data items [\[A505\]](#). This allows Consumers to provide type checking outside of that done by typical interface layers. This specification introduces data structures as they are needed for operations and then
15 summarizes them all in [Section 11](#).

3.3 Lifecycles

There are several different types of lifecycle included in this specification:

Deployment: Deployment is the process by which a particular application becomes available for interactions. Various items have their lifecycle governed by the deployment. Examples
20 include the `registrationData` for a Producer and Producer Offered Entities.

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 Entities.

25 **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 (a duration in seconds) whenever such an item may be 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 as a refinement on an `entityHandle`.

30 3.4 Scopes

There are several scopes referred to in various places in this specification.

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 entities the Consumer configures and any interactions with the entities of the Producer. This
35 scope has a persistent lifecycle.

Entity scope: This scope is initiated when an entity is cloned and as such will be encapsulated by a registration scope. This scope ends when the reference to the entity is explicitly released. As such it encompasses all interactions with the entity. This scope has a persistent lifecycle.

40 **Session scope:** This scope is initiated when the entity needs to store local state and as such will be encapsulated by an entity scope. This scope ends when the session holding that state is released (either via an explicit operation OR via a timeout mechanism). As such it encompasses a set of operation invocations in which the Consumer has supplied the refined entity handle which also encodes the session. This scope has a transient lifecycle.

3.5 Types of Stateful Information

Because WSIA and WSRP are connectionless protocols, 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:

5

Navigational state: This is the state that allows the current page to be correctly generated, including on a page refresh. Web applications typically store this type of state in the URL so that both page refresh and bookmarked pages will generate approximately 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 capability End-Users expect, the Consumer may store this state, or a reference to it, in the URL. The Consumer may also choose to not supply this functionality to its End-Users.

10

Transient state: This is state stored on the Producer related to a sequence of operations (for example, an e-Commerce site may store a shopping cart in its transient state). Once this type of state is generated, the Producer returns a reference to it and the Consumer MUST return this reference on future invocations. This type of state will be referred to as a **Session** (similar to http) and an opaque reference to one is encoded into a refinement on an `entityHandle`.

15

Persistent state: This is state that the Producer persists until either the Consumer or Producer explicitly discards it. This specification defines two kinds of persistent state, each is referred to via a handle which MUST remain invariant once the Producer supplies it to the Consumer. This simplifies a number of issues related to Consumer processing when changes occur relative to a particular persistent state reference. These two kinds of persistent state are:

20

Consumer Registration: Represents a relationship between a Consumer and Producer. 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**.

25

Entity: In addition to the entities a Producer offers for all Consumers to use, the ability of a Consumer to create a unique configuration of one of those entities for its own use is defined. The opaque reference to a configured entity is referred to as an **entityHandle**.

30

3.6 Persistence and statefulness

This specification makes no assumption about the existence of persistence mechanisms at either the Producer or the Consumer [A201]. In the **getMarkup()** and **performInteraction()** calls, the `navigationalState` field carries the state necessary for the entity 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, then it stores the conversational state in an implementation dependent manner, and returns a refined `refHandle` to the Consumer for use during the lifetime of the session.

35

If the Consumer is operating in a stateless manner, then it may choose the way to achieve this. In the case of an HTTP transport the Consumer may employ standard HTTP mechanisms (cookies or URL-rewriting) to push the navigational state or `refHandle` out to its client. If operating in a stateful manner, the Consumer may employ any number of persistence/caching mechanisms [A202].

40

45

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]. Also, a stateful Producer can be expected to relate its private conversational state with the specific client.

3.7 Sessions

In addition to any persistent data, each entity may use a runtime data area (**Session**). An entity MAY establish such a session, and return a refined `refHandle` to reference it within the context of the underlying `entityHandle`, in operations such as `getMarkup()`, `performInteraction()` and `setProperties()`. The Consumer MUST supply this `refHandle` on subsequent requests to the entity until either the Producer indicates it is invalid or it timeouts [A205].

3.8 Producer Mediated Sharing

Producers may implement a sharing mechanism through techniques such as a shared area within sessions for entities to use. The Consumer indicates to the Producer which entities could share such data areas via the `groupID` parameter. The Producer is not required to establish a shared data area for each `groupID`, but if the Producer provides shared data areas, this parameter SHOULD be used as a Consumer hint as to which entities could be grouped together. The `groupID` is scoped to a registration. Therefore only entities in the same registration context with the same `groupID` SHOULD have access to a common shared data area and thereby be shielded from other possible shared data areas.

3.8.1.1 Implementation hints

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 entities with a common shared data area have access to the shared data even in such situations. Possible implementation choices include:

- The Producer stores the shared data in a database and accesses the same database from all cluster nodes.
- In the case of HTTP transport, a Producer can use HTTP sessions to store the shared data. It must implement a mechanism that ensures only one shared HTTP session is established for each `groupID`, even for concurrent requests.

3.9 Information Passing Mechanisms

All information passing enabled by this specification is between exactly one Producer and one Consumer. Any sharing of information within a particular Producer service is outside the scope of this specification. If the Consumer wants the information to be shared by multiple Producer services, the Consumer must “mediate” this sharing (again, using means that are outside the scope of this version of the specification).

3.10 Event Handling

Event handling is explicitly not part of this version of the specification. It might be included in a future version of WSRP.

3.11 Two-phase protocol

5 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 entities 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-phase” capability built into the protocol.

10 In a two-phase interaction, the Consumer first invokes **performInteraction()** on the entity whose markup the End-User interacted with and MUST block other invocations on any entity on the same page for this user until receipt of the response. Those Consumers interacting with the published state of entities MAY update the state of any of the aggregated entities. The Consumer then invokes **getMarkup()** on the entities being aggregated.

15 Interaction semantics are well defined across the spectrum of interaction styles supported in the protocol. In other words, the results of the Consumer invoking **performInteraction()** on an entity, regardless of whether the interaction may have side effects on other entities at the Producer, is well-defined independent of the order of **getMarkup()** invocations on the entities.

20 Side-effects that may cross Producer boundaries are out of scope for this version of the specification, though mechanisms to standardize such interactions are intended for future versions.

3.12 Interaction Lifecycle States

This section defines the state transitions for the relationship between a Producer and a Consumer.

25 3.12.1 Assumptions:

In general the Producer is a web service endpoint exposing one or more entities that generate markup and handle interactions with that markup. How these entities are implemented and managed is not defined by this specification, though it is anticipated that the model of how requests are conveyed to the entities by the Producer will be strongly influenced by this specification.

3.12.2 State 0: Unknown

35 The Consumer has no knowledge that the Producer exists. From this state the Consumer transitions to the **Known** state via discovery; namely by learning the location of the Producer. Examples of mechanisms for accomplishing this discovery include UDDI query, WSIL declarations or other ad hoc mechanisms [A110].

3.12.3 State 1: Known

40 In this state the Consumer knows the location (i.e. its access point and at least the WSDL of the self description method) of the Producer. From this state the Consumer can transition back to the **Unknown** state, but typically transitions to the **Active** state through a process called registration. Additionally, this is the earliest state at which the Consumer MAY request a Producer to describe itself [A104]. This ability is present in all states other than **Unknown**.

3.12.4 State 2: Active

5 Most of the interesting things happen while the Producer is in the **Active** state. This is where both page designers and users can interact with the Producer. It is possible to transition back to the **Known** state by deregistering. The Consumer is free to perform this state transition multiple times.

3.13 Transport Issues

10 Since the transport layer is often used to store various pieces of information (e.g. J2EE load balancing depends on the JSessionID cookie) and this pieces of information often will pertain to the End-User rather than the Consumer, we strongly recommend that Consumers manage transport layer issues, such as cookies, in a manner that correctly returns them to the Producer only for subsequent invocations on behalf of the same End-User.

4 Service Description Interface

15 A Producer may be discovered through mechanisms such as **[UDDI]** or **[WSIL]**, which provide the capabilities of the service also. Other discovery mechanisms (e.g. emailed URL to a properly enabled browser) do not expose these capabilities. The **getServiceDescription()** operation provides a discovery mechanism agnostic means for a Consumer to ascertain a Producer's or entity's capabilities [\[A110\]](#).

4.1 New Data Structures

The operations in this section introduce the following new data structures:

20 4.1.1 ServiceDescription

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

25	ServiceDescription	
	[R] EntityType[]	entities
	[O] String	wSDLURL
	[O] RoleDescription[]	roleDescriptions
	[O] boolean	usesCookies
	[O] boolean	doInitEnvironment
30	[O] boolean	requiresRegistration
	[O] PropertyDescription[]	registrationProperties
	[O] Object	any

Members:

- `entities`: An array of structures (defined in [Section 11](#)) containing the metadata for the Producer Offered Entities.
- `wSDLURL`: The URL for the WSDL abstract description of the Producer.
- `roleDescriptions`: An array of role description structures as defined in [Section 11](#).
- `usesCookies`: A boolean (default value = false) indicating whether or not the Producer uses the cookie support of the http protocol and therefore requires that the Consumer properly manages these for the End-Users as an intermediary.

- `doInitEnvironment`: A boolean (default value = false) indicating whether or not the Producer requires the Consumer invoke **initEnvironment()** for each groupID for each End-Users in order to allow the Producer the opportunity to properly initialize itself (usually related to Producer mediated data sharing). Due to the blocking nature of this invocation, it is recommended that Producers not set this flag unless such invocations are truly required in order to initialize properly.
- `requiresReistration`: A boolean (default value = true) indicating whether or not the Producer requires Consumer registration. If `requiresReistration` is set to false then it MUST be valid to pass null for the `registrationContext` parameter to all operations having this parameter.
- `registrationProperties`: An array of property descriptions for what may and must be supplied during registration.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

4.1.2 UserContext

The `UserContext` structure supplies End-User specific data to operations.

```

UserContext
  [R] String    profileKey
  [O] String[]  producerRoles
  [O] UserProfile profile
  [O] Object    any

```

Members:

- `profileKey`: A string that MAY be used as a reference to the `UserProfile` and that MUST remain invariant for the duration of a Consumer's registration. Establishment of the link between the user profile data and the `profileKey` SHOULD only occur within the context of a secure communication between the Consumer and Producer.
- `producerRoles`: An array of strings each of which specifies a role from the entity's metadata that the Consumer authorizes the End-User to use when interacting with the entity.
- `profile`: End-User profile data structure as defined in [Section 11](#).
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

4.1.3 RegistrationCore

The `RegistrationCore` structure contains fields related to a particular registration of a Consumer with a Producer. It is returned by the **modifyRegistration()** operation and is the portion of a `registrationContext` that allows a Producer to push the storage of state at registration scope to the Consumer.

```

RegistrationCore
  [O] String registrationState
  [O] Object any

```

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 is non-null, the Consumer MUST return this value on any subsequent calls in the context of this registration [\[R362\]](#).

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

4.1.4 RegistrationContext

5 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 extends RegistrationCore
[R] String registrationHandle
```

Members (in addition to those from `RegistrationCore`):

- 10
- `registrationHandle`: An 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.

4.2 getServiceDescription() Operation

15 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, userContext);
```

20 Producers may choose to restrict the information returned in `serviceDescription` based on the supplied registration and user contexts. The minimum information a Producer MUST return is that which declares what is required for a Consumer to register with the Producer [R300][R301][R303]. Producers may also find it useful to restrict the information returned to those portions of the service the registration and user contexts allow the End-User to access on subsequent invocations. It is recommended that Consumers invoke `getServiceDescription()` after registering in order to receive the full capabilities the Producers is offering within the context of that registration.

25

5 Registration Interface

30 A Producer that supports in-band registration of Consumers exposes the optional registration portType. 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 an opaque token that may be used to refer to the registration. This specification calls this token a `registrationHandle`.

5.1 New Data Structures

The operations in this section introduce the following new data structures:

5.1.1 RegistrationData

The `RegistrationData` structure provides the means for the Consumer to supply the data required for registration with a Producer [R355][R356].

RegistrationData		
5	[R] String	consumerName
	[R] String	consumerVendor
	[O] String[]	userProfileExtensions
	[O] String[]	consumerModes
	[O] String[]	consumerWindowStates
10	[O] String[]	consumerExtensions
	[O] Property[]	registrationProperties
	[O] Object	any

Members:

- 15 • `consumerName`: Unique name that identifies the Consumer [R355]. An example of such a name would be the Consumer's URL.
- `consumerVendor`: Name and version of the Consumer's vendor [R356].
- `userProfileExtensions`: An array of named extensions to the user profile structures defined in this specification. While this specification does not provide any such named extensions, it is expected that vendors will define named sets (e.g. SAP-R3-UserProfile).
- 20 • `consumerModes`: An array of modes the Consumer is willing to manage. This specification defines a set of constants for a base set of modes. This array may both reference 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. This array may both reference those constants and additional custom window states of the Consumer.
- 25 • `consumerExtensions`: An array of named extensions the Consumer supports. While this specification does not define what extensions exist, this field allows the Consumer to supply this information to the Producer at registration.
- 30 • `registrationProperties`: List of registration properties. The names of these properties SHOULD be from the set declared in the `RegistrationData` array from the Producer's service description and are not part of this specification.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.
- 35

5.2 register() Operation

Registration describes the transition between Producer state 1 (known) and state 2 (active) [R352]. The Consumer establishes a relationship with the Producer that will be referenced via an opaque handle in subsequent invocations the Consumer makes of the Producer within this relationship [R350]. Both the Consumer and the Producer are free to end this relationship at any time [R500]. The Consumer MUST end the relationship via an explicit call to **deregister()** operation, whereas the Producer MAY end the registration by invalidating the registration identifier.

```
registrationContext = register(registrationData);
```

The returned `registrationContext` is used in all subsequent invocations to reference this registration [R362]. If the Producer's metadata declares registration is not supported (ie. `requiresReistration` flag was set to false), then it is valid to pass null to operations that require a `registrationContext`. If the registration fails (e.g. failed authentication), 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`.

10 5.3 modifyRegistration() Operation

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

```
registrationCore = modifyRegistration(registrationContext, registrationData);
```

15 The supplied parameters reference a pre-existing registration and the modifications desired. If the Producer is choosing to have the Consumer provide persistent storage, the change in registration state is carried in the `registrationState` field of the returned `registrationCore`.

5.4 deregister() Operation

This operation provides means for the Consumer to end a relationship with a Producer.

```
20 void deregister(registrationContext);
```

25 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 `registrationContext` for subsequent invocations. An attempt to use an invalidated `registrationContext` MUST throw a fault indicating the registration is not valid. If the **deregister()** operation fails, the Producer MUST throw a fault message specifying the reason for the failure.

6 Entity Management Interface

30 Producers MUST expose 1 or more logically distinct ways of generating markup and dealing with interaction with that markup [A205], which this specification refers to as **Entities**. The Producer declares the entities it exposes through its description [A104]. This declaration contains a number of descriptive parameters; in particular it includes an `entityHandle` that Consumers may use to refer to the so-called "ProducerOfferedEntity". These entities are pre-
35 configured and non-modifiable by Consumers.

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

6.1 New Data Structures

The operations in this section introduce the following new data structures:

6.1.1 refHandle

5 The `refHandle` parameter provides the finest scoped handle the Consumer has for accessing the entity. When first interacting with an entity for a particular End-User/Consumer session, the Consumer initializes this parameter with the `entityHandle` for the entity targeted by the invocation. If the Producer returns a `newRefHandle` in a `RefHandleContext` from any invocation for this use of the entity for the End-User, the Consumer MUST use the new `refHandle` on subsequent invocations until the Consumer determines the `refHandle` has timed out (i.e. based on either a fault message from the Producer or the expires information supplied with the `refHandle`) [A206].

```
String refHandle
```

15 A common reason for a Producer to return a new `refHandle` is the initialization of a session (e.g. data store for local state). Since the session will have a transient lifetime, the Producer SHOULD encode the base `entityHandle` in the new `refHandle` so that it may cleanly deal with re-establishing a timed out session without requiring additional invocations from the Consumer.

6.1.2 EntityDescription

20 The `EntityDescription` structure contains a set of fields that provide the metadata to describe the entity.

```
EntityDescription
  [R] EntityType  entity
  [O] String      wsdlURL
  [O] Object      any
```

25 **Members:**

- `entity`: A structure (defined in [Section 11](#)) containing the metadata for this particular entity.
- `wsdlURL`: An optional URL for a WSDL abstract description of the Entity. This is provided for those entities that are also directly accessible and for entities to publish any extensions to the data structures this specification defines.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

6.1.3 EntityHandle

35 The `entityHandle` parameter supplies an opaque reference for the Producer to determine what descriptive information to return. For Producer Offered Entities this reference may be extracted from the array on `EntityTypes` in the response to `getServiceDescription()`. For Consumer Configured Entities this reference is returned from a `cloneEntity()` invocation. Once established, an `entityHandle` MUST be invariant until it is released.

```
String entityHandle
```

6.1.4 RefHandleContext

The `RefHandleContext` structure contains fields that may be returned in response to invocations of a number of different operations.

```

RefHandleContext
  [R] String  newRefHandle
  [R] int    refHandleExpires
  [O] Object any

```

5 **Members:**

- `newRefHandle`: A new transient and opaque handle the Producer is supplying for use on future invocations for this use of the entity. The Consumer **MUST** use this new value as the `refHandle` on subsequent invocations until either the Producer supplies another `newRefHandle` or either a fault message from the Producer or the `refHandleExpires` duration indicates that the `refHandle` is no longer valid [A206].
- `refHandleExpires`: An integer value for the number of seconds before the `newRefHandle` will be “timed out” by the Producer and the resources reclaimed. A value of `-1` means the Producer does not time out the resource referred to by `newRefHandle`. This field has no default value. The Producer **MUST** supply this value whenever it supplies a `newRefHandle`.
- `any`: A schema declaration that implementations may extend this structure provided those extensions come from a different namespace.

6.1.5 EntityContext

20 The `EntityContext` structure is returned whenever an entity is cloned or modified. This is then used as a parameter on many other operations to identify the configured entity to handle the invocation.

```

EntityContext
  [O] String  entityState
  [O] Object any

```

25 **Members:**

- `entityState`: An opaque string the entity uses when it depends on the Consumer to store its persistent state [A205]. If `entityState` has a non-null value, the Consumer **MUST** return this value on subsequent calls using the same `entityHandle`. Note that such uses **MAY** span various cycling of the Consumer and therefore this state **MUST** be persisted by the Consumer until successfully invoking **destroyEntities()** with the related `entityHandle`.
- `any`: A schema declaration that implementations may extend this structure provided those extensions come from a different namespace.

6.1.6 EntityResponse

35 The `EntityResponse` structure contains fields returned by the **cloneEntity()** operation.

```

EntityResponse extends EntityContext
  [R] String          entityHandle
  [O] RefHandleContext response

```

Members (in addition to those from `EntityContext`):

- `entityHandle`: An opaque and invariant handle the Producer is supplying for use on future invocations targeted at the new entity.
- `response`: This structure contains fields that are returned from various operations, including a new `refHandle` and the duration before it expires.

6.1.7 InteractionParams

The `InteractionParams` structure contains fields specific to invoking the `performInteraction()` operation.

InteractionParams		
5	[O] boolean	entityStateChangeOK
	[O] Base64	uploadData
	[O] Object	any

Members:

- 10 • `entityStateChangeOK`: A flag (default value= false) by which a Consumer indicates whether or not the processing of the interaction is allowed to return a modified `entityState`. This flag is needed as only the Consumer knows whether or not such a state change would be acceptable. In many cases, the Consumer may choose to invoke `cloneEntity()` to produce an entity directly associated with this End-User before allowing an entity state change to proceed. The sequence by which this scenario is enabled by the protocol is described in section 7.3.1.
- 15 • `uploadData`: A binary data blob that is used if a file is being uploaded.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

6.1.8 InteractionResponse

20 The `InteractionResponse` structure contains the various items `performInteraction()` and `setEntitiesProperties()` can return.

InteractionResponse		
25	[O] RefHandleContext	response
	[O] String	entityState
	[O] String	navigationalState
	[O] String	newWindowState
	[O] String	newMode
	[O] String	redirectURL
	[O] Object	any

30 Members:

- `response`: This structure contains fields that are returned from various operations, including a new `refHandle` and the duration before it expires.
- 35 • `entityState`: This field is where an entity using Consumer side persistent storage may return a change in its persistent state, provided the `entityStateChangeOK` flag in `InteractionParams` had been set to true. The sequence by which an entity can request changing this state when this flag is not set is described in section 7.3.1.
- 40 • `navigationalState`: Opaque representation of navigational state which the entity is returning to the Consumer to indicate a change from what was supplied on the invocation. This field allows the Consumer to properly support End-User page refreshes and page bookmarks. The Consumer MUST supply this value as the `navigationalState` on subsequent invocations for this use of the entity until the entity supplies a replacement value. This enables the correct state of the entity is used when processing the invocation. If this field is not returned to the Consumer, the Consumer MUST supply the same `navigationalState` to the next invocation (often `getMarkup()`) as was supplied to `performInteraction()`.
- 45

- `newWindowState`: A request from the entity to change the window state. The Consumer MAY choose to respect this request, but since the entity can not depend on that choice it MUST NOT encode this new window state into any of its stateful settings. Rather, the entity MUST compute any such impact on stateful settings after the Consumer has actually changed the window state.
- `newMode`: A request from the entity to change the mode. The Consumer MAY choose to respect this request, but since the entity can not depend on that choice it MUST NOT encode this new mode into any of its stateful settings. Rather, the entity MUST compute any such impact on stateful settings after the Consumer has actually changed the mode.
- `redirectURL`: As a result of processing this interaction, the entity is indicating to the Consumer that it would like the user to view a different URL. The Consumer MAY choose whether this request is a replacement for the aggregated page the Consumer is presenting to the End-User or whether some other means of satisfying the entity's request is appropriate (e.g. spawning a new browser window, etc).
- `any`: A schema declaration that implementations may extend this structure provided those extensions come from a different namespace.

6.1.9 properties

The `properties` parameter provides the Consumer a means for setting the portion of an entity's persistent state that has been published as properties. This programmatic means of setting this state enables Consumers to use interfaces other than the `EDIT_MODE` or `CONFIG_MODE` pages generated by the entity for administering the entities accessible through the Consumer

```
Property[] properties
```

6.1.10 PropertyDescription

The properties of an entity are described in its metadata using the following structure.

```
PropertyDescription
  [R] String    name
  [O] String    type
  [O] String    description
  [O] Object    any
```

Members:

- `name`: Name of the property, must not be null.
- `type`: Type of the property, default value = `xsd:string`.
- `description`: A free form description of the property. Intended purpose is for display in any Consumer generated user interface for administering the entity.
- `any`: Opaque extension to the `PropertyDescription` structure, elements from a different namespace.

6.1.11 PropertyDescriptionList

The set of properties of an entity are described in its metadata using the following structure.

```
PropertyDescriptionList
  [R] PropertyDescription[] propertyDescriptions
  [O] Object                any
```

Members:

- `propertyDescriptions`: Array of property descriptions, must not be null.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

6.2 `getEntityDescription()` Operation

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

```
entityDescription = getEntityDescription(entityHandle, registrationContext,
                                         userContext);
```

10 Producers may choose to restrict the information returned in `entityDescription` based on the supplied registration and user contexts. Consumers may choose to alter how they interact with an entity based on the metadata contained in the returned `entityDescription`.

6.3 `cloneEntity()` Operation

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

```
15 entityResponse = cloneEntity(refHandle, registrationContext, entityContext,
                               userContext);
```

20 The supplied `refHandle` MUST refer to either a `ProducerOfferedEntity` or a previously cloned `ConsumerConfiguredEntity`. The initial state of the new entity MUST be equivalent to the state of the entity the supplied handle references. In the case of a `ConsumerConfiguredEntity` that pushes the entity's persistent state to the Consumer, the `entityState` field of the `entityResponse` structure will supply that state. The new `entityHandle` MUST be scoped by the supplied `registrationHandle`.

25 If a Producer chooses to push the persistent state of its entities to the Consumer, it is recommended that the handle referring to the entity encode the `registrationHandle` that scopes the `entityHandle`. It is also recommended, in this case, that the `entityState` encode the `entityHandle` so that the Producer can do reasonable cross checks that it is receiving a consistent set of handles and state.

30 The returned `entityResponse` contains both the `entityHandle` and `entityState` fields for use in subsequent invocations on the configured entity. No relationship between the supplied entity and the new entity is defined by this specification. The Consumer MUST release the new `entityHandle` by invoking **`destroyEntities()`** when it is no longer needed.

35 If the Producer's metadata declares registration is not supported (i.e. `requiresReistration` flag was set to false), then the Consumer MUST invoke **`destroyEntities()`** passing each `entityHandle` that would have been scoped by a registration.

6.4 `destroyEntities()` Operation

40 The Consumer must inform the Producer that a Consumer Configured Entity will no longer be used by invoking this operation.

```
void destroyEntities(registrationContext, entityHandles);
```

The supplied `entityHandles` is an array of type `entityHandle`, each of which the Consumer is informing the Producer it will no longer use. The Producer MUST throw a fault message if not all of the supplied `entityHandles` are scoped by the supplied `registrationContext`. It is a choice of the Producer's implementation whether the resources related to the `entityHandles` are immediately reclaimed or whether transient resources are allowed to timeout first. A Consumer MUST NOT reference any of the supplied `entityHandles` after successfully invoking **destroyEntities()** and MAY reclaim resources related to the supplied `entityHandles` (e.g. `entityState`). If a fault message is generated in the processing of **destroyEntities()**, then the Producer MUST NOT invalidate any of the supplied `entityHandles`. While a `refHandle` is a refinement on an `entityHandle`, Consumers MUST NOT pass `refHandles` to **destroyEntities()** as the purpose of this operation is destroying the persistent entity and `refHandles` refer to a transient refinement on such an entity.

6.5 setEntityProperties() Operation

The entity state in the previous operations was opaque to the Consumer (e.g. as `entityState`). In addition, means are defined by which a Producer may publish information about state in an entity-specific manner. This is enabled through **Properties** that are declared in the metadata specific to an entity. Each property declaration includes a name, type (default = `xsd:string`) [A505][A507]. This operation enables the Consumer to interact with this published portion of an entity's state.

```
interactionResponse = setEntityProperties(entityHandle, registrationContext,  
                                         entityContext, userContext,  
                                         properties);
```

Since **setEntityProperties()** is interacting only with the published portion of the entity's state, it MUST always be safe for the entity to modify its state (i.e. equivalent to `entityStateChangeOK` being true for a **performInteraction()** invocation).

6.6 getEntityProperties() Operation

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

```
Property[] = getEntityProperties(refHandle, registrationContext, entityContext,  
                                userContext, properties);
```

The supplied `properties` parameter declares the set of properties for which the Consumer is requesting their values. The values of the supplied `Property` elements are ignored, only their names are used to determine the queried set of properties. The returned set of `Properties` declares the current values for these properties. If the Consumer passes a null `properties` parameter, the Producer MUST treat this as a request to enumerate the properties of the entity.

6.7 getEntityPropertyDescription() Operation

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

```
propertyDescriptionList = getEntityPropertyDescription(entityHandle,  
                                                        registrationContext,  
                                                        userContext);
```

7 Markup Interface

As user facing web services, one of the required portTypes a WSIA or WSRP compliant service MUST implement is the generation of markup which is to be used to represent the current state of an entity to an End-User and the processing of interactions with that markup [A300]. This section both explains the signatures for the operations related to markup generation and processing interactions and how the concepts of mode and window state impact the generation of the markup.

7.1 New Data Structures

The operations in this section introduce the following new data structures:

7.1.1 groupID

The `groupID` parameter provides means for the Consumer to inform the Producer about how grouping could be done for the entities it is aggregating onto a page. If the Producer provides any capabilities for entities to share runtime data, the `groupID` from the Consumer SHOULD be used, at least in part, to control which entities MAY share data with each other. An example of when this is important is when a Consumer is aggregating onto a single page more than one instance of a type of entity (presumably with different configuration settings). If these are placed in the same shared data area, they will commonly write the same data items with different values and the End-User would likely receive an unusable aggregated page.

```
String groupID
```

Since Consumers will not know what entities could benefit from any Producer mediated data sharing, they SHOULD place all entities from a Producer within a single group until a reason is identified for specifying a separate `groupID`.

7.1.2 CacheControl

The `CacheControl` structure contains a set of fields needed for the entity to managed cached markup fragments.

```
CacheControl
  [R] int      markupExpires
  [O] String[] cacheHints
  [O] Object   any
```

Members:

- `markupExpires`: Number of seconds the markup fragment referenced by this cache control entry remains valid. A value of `-1` means indicates that the markup fragment will never expire.
- `cacheHints`: Array of `CacheHint` constants that define what information of the input space for `getMarkup()` the generated fragment depends on. The Consumer can use this information to check prior to a call to `getMarkup()` if it may get content from the fragment cache or needs to retrieve an updated fragment based on updated input data. The following constants are defined:
 - a. `CACHE_USER`: parts of the `userContext` have been used to generate the markup. The Consumer needs to retrieve new content if the `userContext` structure changes between requests.
 - b. `CACHE_REGISTRATION`: parts of the `registrationContext` have been used to generate the markup. The Consumer needs to retrieve new content if the `registrationContext` structure changes between requests.

- 5 c. `CACHE_MARKUP_PARAMS`: parts of the `markupParams` have been used to generate the markup. The Consumer needs to retrieve new content if the `markupParams` structure changes between requests. This flag only needs to be set if parts of the markup parameters other than the ones listed in Section 7.2.1.2 have been changed.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

7.1.3 MarkupParams

10 The `MarkupParams` structure contains a set of fields needed for the entity to generate markup that will enable the End-User to visualize the state of the entity.

	MarkupParams	
	[R]	Clientdata clientData
	[R]	boolean secureClientCommunications
	[R]	String locale
15	[R]	String markupCharacterSet
	[R]	String markupType
	[O]	String mode
	[O]	String windowState
	[O]	String navigationalState
20	[O]	Property[] clientParameters
	[O]	Object any

Members:

- `clientData`: A structure (defined in [Section 11](#)) that provides information (including `userAgent` and `deviceInfo`) about the client device which will render the markup.
- 25 • `secureClientCommunications`: A flag indicating whether or not the communications between a client and Consumer is secure [\[R401\]](#). A Consumer MUST set this flag as the entity MAY render different content when it knows the delivery channel is secure.
- `locale`: Locale to generate the markup for. The Consumer SHOULD supply this information based on the setting the End-User has requested.
- 30 • `markupCharacterSet`: The `characterSet` the Consumer would like the entity to use for encoding the markup (i.e. the character set for the aggregated page). This encoding may be different from the character set used for the transport of the invocation from the Consumer to Producer. The Producer will either need to use this character set for the response message or properly escape any characters that would otherwise not be properly represented in the character set of the response message.
- 35 • `markupType`: The type of markup to generate (e.g. HTML, XHTML, cHTML).
- `mode`: The mode the entity should render its output for. A set of modes are defined in this specification (see [Section 11](#)). The Consumer SHOULD inspect the entity's metadata to determine which of these modes the entity supports in addition to any Producer defined modes. The Consumer MUST specify either one of the modes from the entity's metadata or `VIEW_NORMAL` (all entities are required to support this mode).
- 40 • `windowState`: The state of this entity's virtual window relative to other entities on the aggregated page (e.g. *normal*, *minimized*, *maximized*, *detached*). Constants for these specification-defined states are found in [Section 11](#).
- 45 • `previousMode`: This field is only supplied when the Consumer has changed the mode for the entity's markup. The Consumer MUST supply the previous value of the mode when such a change has occurred.

- `previousWindowState`: This field is only supplied when the Consumer has changed the window state for the entity's markup. The Consumer MUST supply the previous value of the window state when such a change has occurred.
- 5 • `navigationalState`: This field contains the opaque navigational state for this entity either from the appropriate URL parameter (see [Section 9](#)) or the most recently returned value for this End-User.
- `clientParameters`: Name/value pairs reflected either from the querystring of a URL or from transport level (e.g. http) headers of the initial client request or additional parameters the Consumer supplies from data stores that it manages.
- 10 • `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

7.1.4 MarkupResponse

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

15	<code>MarkupResponse</code>	
	[O] <code>RefHandleContext</code>	<code>response</code>
	[O] <code>String</code>	<code>markup</code>
	[O] <code>CacheControl</code>	<code>cacheControl</code>
20	[O] <code>String</code>	<code>preferredTitle</code>
	[O] <code>boolean</code>	<code>needsUrlRewriting</code>
	[O] <code>Object</code>	<code>any</code>

Members:

- `response`: This structure contains fields that are returned from various operations, including a new `refHandle` and the duration before it expires.
- 25 • `markup`: The markup to be used for visualizing the current state of the entity. 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 either by the entity or the Producer's runtime (e.g. '<' and '>'). The character set of the markup an entity 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 SOAP envelope.
- 30 • `cacheControl`: Defines the caching policies for the returned markup fragment.
- `preferredTitle`: The title the entity would prefer to be used in any decoration of the markup.
- 35 • `needsUrlRewriting`: A flag by which the entity/Producer indicate whether or not Consumer side URL rewriting (see [Section 9](#)) is needed. The Consumer MUST parse the markup for URL rewriting if this flag is set to 'true'.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

7.1.5 Templates

The `Templates` structure contains a set of fields that enable Producer URL writing. The template style format of these fields is defined in [Section 9](#).

5	Templates	
	[O] String	DefaultTemplate
	[O] String	ActionTemplate
	[O] String	RenderTemplate
	[O] String	ResourceTemplate
10	[O] String	SecureDefaultTemplate
	[O] String	SecureActionTemplate
	[O] String	SecureRenderTemplate
	[O] String	SecureResourceTemplate
	[O] String	NameSpacePrefix
	[O] Object	any

15 Members:

- `DefaultTemplate`: This template provides the default value for all the other template fields. Note that the `SecureDefaultTemplate` field MAY provide a first level override of this default value for the fields whose name begin with “Secure” as these frequently involve a different protocol specification.
- 20 • `ActionTemplate`: This template provides the template for URLs that will be directed to the Consumer and processed as a **performInteraction()** on the entity.
- `RenderTemplate`: This template provides the template for URLs that will be directed to the Consumer and processed as a **getMarkup()** on the entity.
- 25 • `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 entity using a secure protocol.
- 30 • `SecureRenderTemplate`: This template provides the template for secure URLs that will be directed to the Consumer and processed as a **getMarkup()** on the entity using a secure protocol.
- `SecureResourceTemplate`: This template provides the template for secure URLs that will be directed to the Consumer and processed as a get using http over SSL/TLS on the named resource.
- 35 • `NameSpacePrefix`: This field provides a string the entity MAY use to prefix tokens that need to be unique on the aggregated page (e.g. JavaScript variables, html id attributes, etc.).
- 40 • `extensions`: An array of properties for use when implementations choose to extend this structure.

7.2 getMarkup() Operation

The Consumer requests the markup for rendering the current state of an entity by invoking:

```
5      markupResponse = getMarkup(refHandle, groupID, registrationContext,  
                                entityContext, userContext, markupParams,  
                                templates);
```

10 As described in section 6.1.1, the Consumer MUST initialize `refHandle` to the value of the `entityHandle` for the targeted entity. Once the Producer has returned a refined `refHandle` (via the `newRefHandle` field of a `RefHandleContext` structure), the Consumer MUST supply the new `refHandle` on subsequent invocations until either the Producer provides yet another replacement or the `refHandle` times out (based on the `refHandleExpires` field in the `RefHandleContext` that supplied the new `refHandle`). All `refHandles` returned in a `RefHandleContext` structure are required to have a transient lifetime and an indication of when they will be invalidated by the Producer due to inactivity.

7.2.1 Caching of markup fragments

15 For performance reasons the Consumer might need to cache the output of **getMarkup()** across a series of requests. The Producer passes information about the cachability of the markup fragment returned by **getMarkup()** in the `cacheControl` structure returned in `MarkupResponse`. 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()**.

20 7.2.1.1 Cachability

Whenever the `cacheControl` field of `MarkupResponse` is filled in the Consumer MAY cache the markup fragment. It must then follow the following invalidation policies to keep the cache up-to-date.

If the `cacheControl` structure is missing the Consumer MUST not cache the markup fragment.

25 7.2.1.2 Cache Invalidation

The Consumer knows when to invalidate its cached fragments from one of the following mechanisms:

- 30 • Time based expiry: The Consumer MUST expire a markup fragment after the `markupExpires` returned in the `cacheControl` of markup response has timed out, counting from the point in time when the **getMarkup()** invocation returned. If the value of `markupExpires` equals -1 the fragment will never time out. If the value of `markupExpires` equals zero the fragment MUST NOT be cached.
- 35 • Hint based expiry: The Producer MAY return hints to the Consumer on what input parameters the generated markup depends on (see definition of the `cacheControl` structure). The Consumer MUST use these hints to invalidate the cached fragments if the designated input parameters changed since the request that resulted in the last invocation of **getMarkup()**.
- 40 • State based expiry: The Consumer MUST expire the markup fragment if parts of the Consumer managed state of the entity changed. This includes changes in the `windowState`, `mode`, `navigationalState` and `entityState`. These attributes are not included in the semantics of the `CACHE_MARKUP_PARAMS` hint.
- Action based expiry: An invocation of **performInteraction()** MUST invalidate all cache entries for the targeted entity or `refHandle`.

7.3 performInteraction() Operation

End-User interactions with the generated markup may result in invocations for the entity to respond to the interactions [A400]. In the case where these may change the navigationalState or some data the entity is storing in a shared data area (including a database), a blocking call is needed to process the interaction prior to markup being generated for a page refresh to the End-User. Therefore this blocking operation is defined for processing interaction and the state changes they may cause:

```
interactionResponse = performInteraction(refHandle, groupID,  
                                       registrationContext, entityContext,  
                                       userContext, markupParams,  
                                       interactionParams)
```

Since this is a blocking operation, the Consumer MUST wait for the response before invoking **getMarkup()** on the entities it is aggregating. This permits any Producer mediated sharing to proceed safely (provided it happens in a synchronous manner). Since this is the operation that potentially returns state to the Consumer for storage, this also allows Consumers who wish to store this by pushing it to their client to do so before opening the stream for the aggregated page. This allows storage in the URL of the aggregated page that often enables the End-User to bookmark the page for reuse later.

7.3.1 Updating Persistent Entity State

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

1. Only the entity 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 entity 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 entity state changes happen in a manner that has Consumer approval for the change to occur while the entity decides both when the change is required and its exact character. The `interactionParams` parameter to the **performInteraction()** and **setEntityProperties()** operations indicates whether or not it is safe for the entity to modify its persistent state through the `entityStateChangeOK` flag. If the Consumer has set the `entityStateChangeOK` flag to true, the entity MAY modify its persistent state regardless of whether it is persisted on the Producer or Consumer. If the Consumer has set the `entityStateChangeOK` flag to false, the entity MAY 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. A common Consumer processing for this fault message is to determine whether the End-User is allowed to personalize an instance of the entity on this aggregated page. If the End-User is allowed such personalization in general but not on the currently referenced entity, then the Consumer SHOULD clone the entity (i.e. invoke **cloneEntity()**) and change the reference on the aggregated page for this End-User to the new `refHandle` and then re-invoke the **performInteraction()** operation with the new `refHandle`.

[RT: Possibly a sequence diagram]

7.4 **initEnvironment()** Operation

5 In general, the Producer implicitly manages its own environment. There are cases, however, when assistance from the Consumer in initializing the environment is useful. Initializing any Producer mediated sharing prior to (potentially concurrent) requests for markup can greatly simplify the Producer's complexity in managing the environment. The Producer indicates it is interested in this assistance from the Consumer by setting to true the `doInitEnvironment` flag in the `ServiceDescription` structure returned by **getServiceDescription()**. This operation is how the Consumer provides such assistance:

```
void initEnvironment(registrationContext, groupID);
```

15 If the Producer's metadata has set the `doInitEnvironment` flag to true, then the Consumer MUST invoke **initEnvironment()** once for the `groupID` prior to invoking **getMarkup()** for this End-User for any entity using the same `groupID`. The Consumer MAY invoke **initEnvironment()** concurrently, each with a different `groupID`, for the interactions with the End-User. If at any time the Producer throws a fault message [RT: define it] indicating the environment for this `groupID` with this End-User has been invalidated at the Producer, then the Consumer MUST again invoke **initEnvironment()** for this `groupID` and SHOULD then reprocess the invocation that caused the fault message to be thrown.

25 An example of the usefulness of this operation is a portal page with multiple entities from a Producer involved in a series of interactions, all tied to a single `groupID` (i.e. may be involved in Producer mediated data sharing). Since portals usually issue concurrent requests to each of the entities represented on the page, the complexity of properly initializing the Producer's environment is significant, particularly if operating in a load balanced environment (as discussed in the next section). If the Producer set the `doInitEnvironment` flag to true, then the Consumer MUST assist the Producer by invoking this operation once per `groupID` and block any other invocation targeted at an entity sharing the same `groupID` until this invocation returns. Failure to do so will often result in the entities not interacting with each other in the expected manner.

7.4.1 Load Balancing

35 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 Consumer detects the presence of cookies in a response from the Producer, it MUST ensure these same cookies are properly encoded in subsequent requests for the End-User. If the Producer requires such support of Consumers, it MUST indicate so using the `usesCookies` metadata flag.

7.4.2 Consumer Transitions across Bindings

5 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 regardless of whether the operation is in the same binding as where the cookie was set. Another implication of the Producer indicating it uses cookies is that the Consumer SHOULD NOT do any protocol transitions (e.g. from http to https) as cookies are often managed in a manner that does not allow the information to be shared across such a transition. Switching between protocols (e.g., going from http to https) will likely break load balancing on the Producer. This is because the request being routed via the new binding will almost certainly go through a different load balancer and likely a different session manager.

7.5 Stateful Entity Scenarios

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

15 7.5.1 No State

This type of entity maintains no state, but encodes everything required to generate the markup on the URL causing the invocation of **getMarkup()** [A201]. Often these entities 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.

20

Note: Invocations of **performInteraction()** MAY happen in this scenario if the entity impacts some backend system as a result of the invocation as this impact could change the markup some other entity will generate.

25

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

Method	Parameter/Field	Value	Comments
performInteraction	refHandle	Producer Offered Entity or Consumer Configured Entity	No Producer-side state, so this handle never changes.
	markupParameters / navigationalState	Consumer extracts value from link.	Navigational state encoded on the URLs in the markup only.
	interactionResponse / navigationalState	This type entity does not return navigationalState.	
getMarkup	refHandle	Producer Offered Entity or Consumer Configured Entity	No Producer-side state, so no refined handle is ever generated.
	markupParameters / navigationalState	Consumer extracts value from link.	Navigational state from the URL.

7.5.2 Navigational State Only

5 This type of entity does not maintain state at the Producer, but does push navigational state out to the Consumer. Both to support these entities and to assist Consumers in properly supporting End-User page refreshes and bookmarks, entities are allowed to return their navigational state (`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].

10 A stateless Consumer can store the `navigationalState` for all of its aggregated entities 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 entities 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 entities aggregated on the page, a `getMarkup()` invocation is not allowed to modify the `navigationalState`. Only an invocation of `performInteraction()` is allowed to modify the `navigationalState` of an entity.

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

Method	Parameter/Field	Value	Comments
<code>performInteraction</code>	<code>refHandle</code>	Producer Offered Entity or Consumer Configured Entity	No Producer-side state, so this handle never changes.
	<code>markupParameters / navigationalState</code>	Consumer extracts value from link or previous value.	
	<code>interactionResponse / navigationalState</code>	Entity may compute a changed <code>navigationalState</code> .	
<code>getMarkup</code>	<code>RefHandle</code>	Producer Offered Entity or Consumer Configured Entity	No Producer-side state, so no refined handle is ever generated.
	<code>markupParameters / navigationalState</code>	From link or previous value or from <code>performInteraction</code> .	

7.5.3 Local state

20 Entities storing state locally on the Producer establish a **Session** and return an opaque reference (a `refHandle` refinement on the underlying `entityHandle`) the Consumer is then required to return on all subsequent invocations on the entity for this End-User. These entities 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].

25

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

Method	Parameter/Field	Value	Comments
performInteraction	refHandle	Producer Offered Entity or Consumer Configured Entity or refined handle that also encodes session info	With Producer side state, the refined handle offers ability to store information without impacts message size to Consumer.
	markupParameters / navigationalState	Consumer extracts value from link or previous value.	
	interactionResponse / navigationalState	Entity may compute a changed navigationalState.	
getMarkup	refHandle	Producer Offered Entity or Consumer Configured Entity or refined handle that also encodes session info	With Producer side state, the refined handle offers ability to store information without impacts message size to Consumer.
	markupParameters / navigationalState	From link or previous value or from performInteraction.	

7.6 Modes

5 An entity 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 reflect 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. Entities may request mode changes either through parameters on a link that an End-User activates or through a returning a `newMode` from **performInteraction()**, but 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.

10 An entity must support the `VIEW` mode and may support the modes `EDIT`, `HELP`, `CONFIG` and `PREVIEW`. During **getMarkup()** and **performInteraction()** invocations the Consumer indicates to the entity its current mode via the `MarkupParams` data structure.

15 7.6.1 VIEW Mode

20 The expected functionality for an entity in `VIEW` mode is to render markup reflecting the current state of the entity. The `VIEW` mode of an entity 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 an entity in the `VIEW` mode may depend on configuration, personalization and all forms of state.

All entities must support the `VIEW` mode.

7.6.2 EDIT Mode

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

5

Typically, entities in `EDIT` mode will set or update entity state by making these changes permanent for the entity. How such changes interact with the Consumer managing the usage of entities by End-Users is discussed in section 7.3.1.

7.6.3 HELP Mode

10

When in `HELP` mode, an entity may provide a simple help screen that explains the entity and its expected usage. Some entities will provide context-sensitive help based on the markup the End-User was viewing when entering this mode.

7.6.4 CONFIG Mode

15

Within the `CONFIG` mode, an entity should provide content and logic that let a user with administrator privileges (Administrator portletRole, or equivalent) customize the behavior of the entity. The `CONFIG` mode may include one or more screens which users can navigate to enter their customization data.

20

Typically, entities in `CONFIG` mode will set or update entity state by making these changes permanent for the entity.

7.6.5 PREVIEW Mode

25

In `PREVIEW`, an entity should provide a rendering of its standard `VIEW` mode content, as a visual sample of how this entity 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, for example.

7.6.6 PREVIOUS Mode

30

The Consumer **MUST NOT** set the mode for an entity to `PREVIOUS_MODE`. It is only provided as a means for the entity to request a mode change to the previous mode. If the Consumer does not have a previous mode for the entity, then it **MUST** process `PREVIOUS_MODE` as a switch into `VIEW_MODE`.

7.6.7 Custom Modes

35

The extensible `RegistrationData` structure provides a field for Consumers to declare additional custom modes. In addition, the extensible `EntityType` structure provides a field for entities to declare what modes they understand. A Consumer **SHOULD NOT** set a mode an entity does not understand. An entity **MUST** map any mode it does not understand to `VIEW_MODE`.

7.7 Window States

40

Window state is an indicator of the amount of page space that will be assigned to the content generated by an entity. This hint is provided by the Consumer for the entity to use when deciding how much information to render in the generated markup.

An entity **MUST** support the `VIEW_NORMAL` window state and **SHOULD** support the `VIEW_MINIMIZED` and `VIEW_MAXIMIZED` window states.

7.7.1 NORMAL Window State

5 The `VIEW_NORMAL` window state indicates the entity is likely sharing the aggregated page with other entities. It **MAY** also indicate that the target device has limited display capabilities. Therefore, an entity **SHOULD** restrict the size of its rendered output in this window state.

All entities **MUST** support the `VIEW_NORMAL` window state.

7.7.2 MINIMIZED Window State

10 When the window state is `VIEW_MINIMIZED`, the entity **SHOULD** render itself using minimal space. The entity **SHOULD** render no visible markup in this case, but is free to include non-visible data such as javascript or hidden forms. The Consumer **MUST** invoke the **getMarkup()** operation for the `VIEW_MINIMIZED` state just as for all other window states. The Consumer **MAY** render the title, controls and decorations related to the entity.

15 7.7.3 MAXIMIZED Window State

The `VIEW_MAXIMIZED` window state is an indication the entity is likely the only entity being rendered in the aggregated page, or that the entity has more space compared to other entities in the aggregated page. An entity **SHOULD** generate richer content when its window state is `VIEW_MAXIMIZED`.

20 7.7.4 PREVIOUS Window State

The Consumer **MUST NOT** set the window state for an entity to `VIEW_PREVIOUS`. It is only provided as a means for the entity to request a window state change to the previous window state. If the Consumer does not have a previous window state for the entity, then it **MUST** process this request as a switch into `VIEW_NORMAL`.

25 7.7.5 Custom Window States

30 The extensible `RegistrationData` structure provides a field for Consumers to declare additional custom window states. In addition, the extensible `EntityType` structure contains a field for entities to declare what window states they understand. A Consumer **SHOULD NOT** set a window state an entity does not understand. An entity **MUST** map any window state it does not understand to `VIEW_NORMAL`.

8 Security

WSIA and WSRP compliant systems 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](#).

35 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.

40

5 The uses of document-level security mechanisms are not covered in this version of the specification since several important standards (particularly 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.

10 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.

8.1 Authentication of Consumer

15 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.

8.2 Confidentiality & Message Integrity

20 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.

25 For Producer - Consumer communications, the Producer declares the use of SSL/TLS via the service's WSDL having an https: service endpoint.

30 For Consumer – End-User client communications, the Consumer indicates in the `MarkupParams` structure whether or not communications with the End-User is happening in a secure manner. The entity 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 communications are required.

8.3 Access control

35 A Consumer MAY implement access control mechanisms that restrict which End-Users may access entities and operations on those entities. Additionally, a Producer MAY implement access control programmatically through the use of either a federated user identity or user roles. A standard set of roles is defined to facilitate easy mapping of common roles. In addition, a Producer's `ServiceDescription` MAY declare support for additional roles. If a Producer declares such roles, a Consumer MAY map End-Users to those roles in any manner it chooses. Common mapping choices are expected to be: ignoring the additional roles and administrator mapping of these additional roles to those supported by the Consumer. Producers that use roles (standard or custom) SHOULD implement appropriate default behavior in the event a Consumer does not assert any role for the End-User.

8.4 Roles

45 A Producer declares the roles each entity is capable of supporting in the [EntityType metadata](#) described in [Section 11](#).

8.4.1 Role Assertions

When a Producer declares roles for an offered entity, Consumers SHOULD assert roles for End-Users when interacting with that entity and any entities cloned from it. The mechanism by which an End-User is mapped to entity roles is entirely controlled by the Consumer.

5 8.4.2 Standard Roles

10 To ease the mapping of End-Users to roles and to facilitate plug and play, the following standard role names are provided along with an abstract definition of semantics associated with each. These definitions suggest progressively restrictive levels of access to the entity and are provided as guidelines only. The specific semantics of these roles is left to each Producer's implementation.

15 Administrator:	This role typically grants the highest level of access to functionality of an entity.
User:	This role is typically associated with End-Users who may personalize some set of properties for an entity.
15 Guest:	this role is typically associated with End-Users who may view an entity on a page but not modify any of it's properties

9 Markup

20 A variety of concerns related to entity's generating markup that Consumers can safely aggregate into a page and then properly process End-User interactions are addressed by this section [\[A301\]](#).

9.1 Encoding

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

9.2 URL Considerations

30 As part of its markup, an entity will often need to create URLs that reference the entity itself. For example ,when an End-User activates one of these URLs, by clicking a link or submitting a form, the result should be a new invocation targeted to the entity. This section deals with the different possibilities for how the entity can encode these URLs in its markup.

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

- URLs the entity 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 browser 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.

5 This implies that URLs must be encoded so that the Consumer intercepts them and re-routes them to the correct entity at the Producer, including the proper context. Because the same entity can be instantiated more than once in a single page, encoded URL's will have to allow the Consumer to track the entity 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
10 two options to make the encoded URLs point back to the Consumer and consist of all necessary information for the Consumer to properly process the activation of an URL:

- The Consumer can pass information on its context to the entity. The entity 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 disadvantage is that the
15 entity will not be able to serve static content as the content depends on Consumer runtime settings.
- The entity can use a special syntax to encode its URLs. It is then the task of the Consumer to detect URLs in the markup and modify them according to its requirements. The markup generated by the entity is now Consumer independent,
20 allowing the entity to exploit caching mechanisms or even serve static content. However, the Consumer will be more complex as it needs to parse the markup to locate and rewrite the URLs, requiring extra processing time. Consumers SHOULD seek to minimize this impact on performance by using efficient encoding and parsing mechanisms (for example, the Boyer-Moore algorithm).
25

As there is no clear advantage to either technique, both styles of URL encoding are supported (see sections 9.2.1 and 9.2.2). This facilitates both the capabilities of the Producer and Consumer with regards to the ability to adapt the generated markup and requires that the following semantics are followed:

1. IF an entity's metadata declares it is will to process URL templates, then the Consumer
30 MUST supply templates for the entity to use.
2. IF an entity is unable to completely write the URLs for its markup, it MUST set the `needsUrlRewriting` flag in `MarkupResponse` to 'true'.
3. If the Consumer does not supply appropriate templates or the `needsUrlRewriting` flag in
35 `MarkupResponse` 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 URLs in a special way. The Consumer could always analyze the markup semantically and syntactically, detect URLs and rewrite them.
40 This approach however would be very difficult and time consuming to implement for the Consumer, for reasons that include such a rewriting algorithm being dependent on the markup type and version. Therefore both the Consumer and the Producer URL writing scenarios are introduced for convenience.

45 Entities MUST adopt the following convention for including non-ASCII characters within 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).

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 <http://www.w3.org/TR/html40/charset.html> - doc-char-set encoding to which the document carrying the URI may have been transcoded.

5 9.2.1 Consumer URL Writing

All URLs the Consumer needs to write are demarcated in the markup by a token (`wsia: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 write the 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 querystring (name/value pairs separated by '&' characters) with several well-known parameter names specifying what the Consumer needs in order to both correctly write the URL and then process it when an End-User activates it. This results in an URL declaration of the form:

[3rd F2F: Change token to `wsia:rewrite` for now. Revisit value of less human readable token once an implementation is available to test the impact.]

```
wsia:rewrite?name1=value1&name2=value2 .../wsia:rewrite
```

Where the following well-known parameter names are defined:

20 9.2.1.1 `wsia:navigationalState`

The value of this 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 send the current navigational state.

9.2.1.2 `wsia:mode`

25 Activating this URL includes a request to change the `mode` parameter in `markupParams` into the mode specified as the value for this parameter name. This must be one of the modes detailed in section 7.6. The Consumer MUST process this URL request to change the mode prior to invoking operations on the entity.

9.2.1.3 `wsia>windowState`

30 Activating this URL includes a request to change the `windowState` parameter in `markupParams` into the window state specified as the value for this parameter name. This must be one of the values detailed in section 7.7. The Consumer MUST process this URL request to change the window state prior to invoking operations on the entity.

9.2.1.4 `wsia:url`

35 When the `urlType` parameter is "Resource", 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 instead of a web service invocation.

9.2.1.5 `wsia:token`

40 When the `urlType` parameter is "NameSpace", this parameter provides the actual token that is to be namespaced.

9.2.1.6 wsia:urlType

This required parameter controls how the resulting URL is processed when activated. The following values are defined:

9.2.1.6.1 Action

5 Activation of the URL will result in invoking **processInteraction()** on the entity that
generated the markup. Prior to invoking this method the Consumer will analyze the
querystring parameters of the URL to determine if a mode and/or window change is
specified. The Consumer MUST process all mode and window state change requests
invoking the operation. All querystring parameters not defined by this specification will be
10 passed to **processInteraction()** as `clientParameters`. In addition the Consumer MUST
check for the presence of the `wsia:navigationalState` parameter. If this parameter is
present its value will be passed in the `navigationalState` field of the `MarkupParams`
structure. If there is no such parameter, the Consumer MUST supply the current
navigational state of the entity instead.

15 9.2.1.6.2 Render

 Activation of the URL will result in invoking **getMarkup()**. This mechanism permits an
entity's markup to contain URLs, which do not involve changes to local state, to avoid the
overhead of the two phase processing by directly invoking **getMarkup()**. The URL MAY
specify a `wsia:navigationalState` parameter that allows for state changes that result in
20 different markup being rendered. The Consumer MUST pass all the URL's querystring
parameters not defined by this specification as `clientParameters` in the `MarkupParams`
data structure.

9.2.1.6.3 Resource

25 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. For the HTTP protocol this maps to a
"get" on the underlying resource. The URL for the resource (including any querystring
parameters) is encoded as the value of the `wsia:url` parameter.

9.2.1.6.4 Namespace

30 This tells the Consumer that the "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 keeps the performance
impacts of the Consumer parsing to a single pass of the markup. The actual name that
needs to be rewritten is encoded in the `wsia:token` parameter. See section 9.3 for more
details on namespace rewriting.

35 9.2.1.7 wsia:secureURL

A boolean indicating the resulting URL should involve secure communications between the client
and Consumer as well as between the Consumer and Producer. The default value of this boolean
is false.

9.2.1.8 wsia: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 individually. Consumers MAY want to process such resources in a manner that allows caching of the resulting resource by the End-User's browser. In particular, Consumers MAY process namespace rewriting by using a prefix that is unique to the user/entity pair provided any such prefix is held constant for the duration of the user's session with any one entity.

[RT: A section for some URL examples would be helpful here.]

9.2.2 Producer URL Writing

Entities often are willing to properly write URLs for the Consumer as this decentralizes the preparation of the page for rendering and thereby may provide better page load performance to the End-User. At other times entities choose to dynamically compute the URL in script on the End-User's machine just before activating the URL. To enable these functionalities, several properties are defined by which the Consumer MAY indicate how it needs URLs formatted in order to process them properly. These all take the form of a simple template, namely:

```
http://www.Consumer.com/path?wsia:urlType={wsia:urlType}&...
```

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 and contain the name of the parameter that should be replaced. All content outside the {} pairs MUST be treated by the entity as constants the Consumer wishes to receive when the URL is activated. The list of defined parameter names matches those in the section 9.2.1 with the addition of `wsia:clientParameters` and `wsia:refHandle`. These parameters define where the entity should place data items it needs in the `clientParameters` field of the `markupParams` data structure when the URL is activated and a reference the Consumer MAY use to determine the target entity for an invocation. The value replacing `wsia:clientParameters` should follow the pattern of a URL querystring (name/value pairs separated by the '&' character).

This specification defines well-known property names that entities MAY declare in their published state. Entities making such a declaration MUST support all of these properties such that the Consumer does not need to parse the markup to write proper URLs. The defined properties names of this type are:

9.2.2.1 wsia:ActionTemplate

Activation of the URL will result in invoking `performInteraction()`. The Consumer SHOULD integrate placeholders for the tokens `wsia:navigationalState`, `wsia:entityMode` and `wsia>windowState` in its template. The Consumer MUST apply changes in mode or state before invoking `performInteraction()`.

9.2.2.2 wsia:SecureActionTemplate

Equivalent to `wsia:ActionTemplate` using secure communications.

9.2.2.3 wsia:RenderTemplate

Activation of the URL will result in invoking `getMarkup()`. The Consumer SHOULD integrate placeholders for the tokens `wsia:navigationalState`, `wsia:entityMode` and `wsia>windowState` in its template. The Consumer MUST apply changes in mode or state before invoking `getMarkup()`.

9.2.2.4 wsia:SecureRenderTemplate

Equivalent to `wsia:RenderTemplate` using secure communications.

9.2.2.5 wsia:ResourceTemplate

5 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. For the http protocol this maps to a “get” on the underlying resource. The Consumer SHOULD integrate placeholders for the token `wsia:url` to allow the entity to place the address of the URL.

9.2.2.6 wsia:SecureResourceTemplate

Equivalent to `wsia:ResourceTemplate` using secure communications.

10 9.2.2.7 wsia:DefaultTemplate

A template whose value is to be used as the default value for any template whose value is not supplied. Consumers setting just this template value SHOULD also set a separate default for the secure communications templates using `wsia:SecureDefaultTemplate` unless the default template already uses secure communications. The default value of this template is
15 “`wsia:rewrite?wsia:navigationalState={wsia:navigationalState}&wsia:urlType={wsia:urlType}&wsia:mode={wsia:mode}&wsia>windowState={wsia>windowState}&wsia:url={wsia:url}&wsia:token={wsia:token}&wsia:rewriteResource={wsia:rewriteResource}&wsia:refHandle={wsia:refHandle}&{wsia:clientParameters}`”. It should be noted that this default value provides the correct syntax for the Consumer to then process the “URL” using the rules from section 9.2.1.

20 9.2.2.8 wsia:SecureDefaultTemplate

This template provides a value that overrides the one supplied for `wsia:DefaultTemplate` for those templates whose names begin with “wsia:Secure”. The default value of this template is
25 “`wsia:rewrite?wsia:secureURL=true&wsia:navigationalState={wsia:navigationalState}&wsia:urlType={wsia:urlType}&wsia:mode={wsia:mode}&wsia>windowState={wsia>windowState}&wsia:url={wsia:url}&wsia:token={wsia:token}&wsia:rewriteResource={wsia:rewriteResource}`”.

9.2.2.9 wsia:NamespacePrefix

The Producer can use the content of this property as a prefix for tokens that need to be unique on the aggregated page from the Consumer.

9.2.3 BNF-like Description of URL formats

30 ConsumerURL = BeginToken NameValuePairs EndToken
BeginToken = ‘wsia:rewrite?’
EndToken = ‘/wsia:rewrite’
NameValuePairs = (NameValuePair (‘&’ NameValuePair)*
NameValuePair = (BaseNames ‘=’ Value) | UriTypePair | SecureUrlPair | RewriteResourcePair
35 | EntityNameValuePair
BaseNames = ‘wsia:navigationState’ | ‘wsia:mode’ | ‘wsia>windowState’ | ‘wsia:url’ | ‘wsia:token’
UriTypePair = UriTypeName ‘=’ (‘Action’ | ‘Render’ | ‘Resource’ | ‘NameSpace’)
SecureUrlPair = SecureUrlName ‘=’ BooleanValue
RewriteResourcePair = RewriteResourceName ‘=’ BooleanValue
40 UriTypeName = ‘wsia:urlType’

SecureUrlName = 'wsia:secureUrl'
RewriteResourceName = 'wsia:rewriteResource'
BooleanValue = ('true' | 'false')
Value = Text
5 EntityNameValuePair = Text '=' Text

ProducerURLTemplate = (Text* ReplacementToken)*
Text = <any character with '{', '}', '&' and '/wsia:rewrite' escaped>
ReplacementToken = '{ ParameterName }'
10 ParameterName = BaseNames | UriTypeName | SecureUrlName | RewriteResourceName |
'wsia:clientParameters' | 'wsia:refHandle'

9.3 Namespace Encoding

15 Aggregating multiple entities from different sources can potentially result in naming conflicts for various types of elements: named attributes, form fields, JavaScript functions and variables, etc. Such tokens must therefore be encoded to an entity-instance specific namespace [\[A301\]](#). The entity does this by prefixing the name of the resource by a namespace prefix. Similar to the case of URL rewriting, two options exist to obtain such a prefix.

9.3.1 Consumer Rewriting

20 The entity uses the static, predefined method (section 9.2.1) to denote tokens needing 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 is done using the same method as URL rewriting and is described in that section. It is expected that the length this method adds to names will make it unwieldy to content authors, but the expectation that tooling/runtime support can alleviate most of this burden led to reusing this
25 singular technique for the Consumer parsing/rewriting the markup.

9.3.2 Producer Writing

30 The entity uses a namespace provided by the Consumer to prefix these tokens in its markup. The Consumer ensures this prefix is unique for the page aggregation, so the Consumer is not required to process the markup. The Consumer supplies the prefix the entity needs to use via the `NamespacePrefix` field in the `Templates` structure.

9.3.3 General issues

35 If namespace encoding is used for form parameters or other data the entity receives as in a **performInteraction()** or **getMarkup()** invocation, then the Consumer is responsible for stripping the namespace prefix from the parameter names before passing them to the Producer. This means the entity logic can be agnostic regarding namespace issues except when encoding parameters in the markup.

40 As a rule of thumb, named parameters that are processed by the Consumer will ALWAYS consist of a namespace prefix, named parameters that are processed by the Producer will NEVER consist of a namespace prefix.

9.4 Markup Fragment Rules

5 Because the Consumer aggregates the markup fragments produced by entities 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 entity. So in order to be aggregatable the entity must ensure that the markup conforms to the following general guidelines [A300][A302].

10 The disallowed tags listed below are those tags that impact other entities 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.

9.4.1 HTML

9.4.1.1 Disallowed Tags

15 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 entity, an entity MUST NOT use the following tags:

base
body
frame
frameset
head
html
title

9.4.1.2 Other Tags

20 There are some tags that are specifically prohibited by the HTML specs from existing outside the <head> of the document. However, for some of these tags browser implementations may vary. For example: current versions of Internet Explorer and Netscape Navigator both support the style tag anywhere within the document.

It is up to the entity developer to decide when using such tags is appropriate. Here is a list of tags that fit this description:

25

Link
Meta
Style

9.4.2 XHTML

9.4.2.1 Disallowed Tags

Base
Body
head
html
title

9.4.2.2 Other Tags

link
meta
style

9.4.3 XHTML Basic

5 9.4.3.1 Disallowed Tags

base
body
head
html
title

9.4.3.2 Other Tags

link
meta
style

9.5 CSS Style Definitions

10 One of the goals of an aggregated page is a common look-and-feel across the entities contained on that page. This not only affects the decorations around the entities, but also their content. Using a common CSS style sheet for all entities and defining a set of standard styles provide this common look-and-feel without requiring the entities to generate Consumer specific markup. Entities SHOULD use these style definitions in order to participate in a uniform display of their content by various Consumers.

15

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

9.5.1 Links (Anchor)

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

9.5.2 Fonts

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

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

If an entity author wants a certain font type to be larger or smaller, they should indicate this using a relative size.

Example1: <div class="wsia-font" style="font-size:larger">Important information</div>

- 10 Example1: <div class="wsia-font-dim" style="font-size:80%">Small and dim</div>

9.5.3 Messages

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

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

9.5.4 Sections

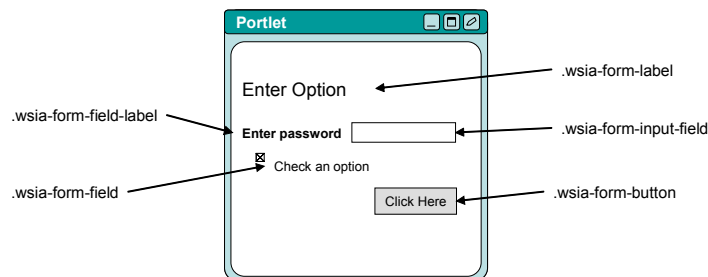
Section style definitions affect the rendering of markup sections such as table, div and span (alignment, borders, background color, etc) as well as their text attributes.

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

9.5.5 Forms

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

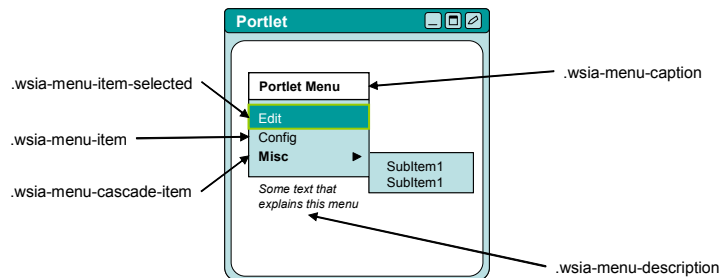
Style	Description
.wsia-form-label	Text used for the descriptive label of the whole form (not the labels for fields).
.wsia-form-input-field	Text of the user-input in an input field.
.wsia-form-button	Text on a button
.wsia-icon-label	Text that appears beside a context dependent action icon.
.wsia-dlg-icon-label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



9.5.6 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
.wsia-menu	General menu settings such as background color, margins, etc
.wsia-menu-item	Normal, unselected menu item.
.wsia-menu-item-selected	Selected menu item.
.wsia-menu-item-hover	Normal, unselected menu item when the mouse hovers over it.
.wsia-menu-item-hover-selected	Selected menu item when the mouse hovers over it.
.wsia-menu-cascade-item	Normal, unselected menu item that has sub-menus.
.wsia-menu-cascade-item-selected	Selected sub-menu item that has sub-menus.
.wsia-menu-description	Descriptive text for the menu (e.g. in a help context below the menu)
.wsia-menu-caption	Menu caption



5

10 User Information

This specification provides a mechanism for entities to use End-User information as a means for personalizing behavior to the current user [A600][A606]. A standard set of user attributes have been derived from P3P User Data and are defined in Section 11. Extensibility is supported with Consumer's able to indicate to the Producer during registration what set of user profile extensions it supports and an entity's metadata declaring what user profile items it uses. The following table maps the nested profile structures to profileNames:

10

Profile Name	Field Name	Structure 1	Structure 2
name/prefix	prefix	name	
name/given	given	name	
name/family	family	name	

name/middle	middle	name	
name/suffix	suffix	name	
name/nickName	nickName	name	
birthDate	birthDate		
gender	gender		
employerInfo/employer	employer	employerInfo	
employerInfo/department	department	employerInfo	
employerInfo/jobTitle	jobTitle	employerInfo	
homeInfo/address/name	name	homeInfo	address
homeInfo/address/street	street	homeInfo	address
homeInfo/address/city	city	homeInfo	address
homeInfo/address/stateprov	stateprov	homeInfo	address
homeInfo/address/country	country	homeInfo	address
homeInfo/address/org	org	homeInfo	address
homeInfo/telephone	telephone	homeInfo	
homeInfo/email	email	homeInfo	
homeInfo/online	online	homeInfo	
workInfo/address/name	name	workInfo	address
workInfo/address/street	street	workInfo	address
workInfo/address/city	city	workInfo	address
workInfo/address/stateprov	stateprov	workInfo	address
workInfo/address/country	country	workInfo	address
workInfo/address/org	org	workInfo	address
workInfo/telephone	telephone	workInfo	
workInfo/email	email	workInfo	
workInfo/online	online	workInfo	

Entities that need access to user information MUST declare in its [metadata](#) the specific user profile fields the entity needs using the name specified above.

10.1 Passing User Information

- 5 User information is passed to the Producer when a Consumer invokes operations beyond registration and **initEnvironment()**. A Consumer SHOULD provide the specific fields the entity declared it needed unless the information is not available or is restricted by policy (e.g. privacy policy).

10.2 User Identity

5 Mechanisms that support federation of user identity between web services system 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 to these standards be the means to passing the required information.

10 It is anticipated that some entities will interact with one or more back-end applications that require a user identity for the End-User. If user identity known by the back-end application is not the same as that authenticated by the Consumer, the entity SHOULD require the End-User to provide the necessary information (preferably using secure transport) for use with the back-end application via markup interactions.

11 Data Structures

15 It is often necessary to pass data to operations. Wherever possible typed data object are defined as the transport mechanism for this data [\[A504\]\[A505\]](#). Extensibility elements are also provided for vendor or application specific data extensions. While many of these data structures have been described in the sections describing the operations in which they are referenced here (with hyperlinks back to those explanations) in order to have all the structures defined in one place. This non-normative section uses an IDL like syntax for describing these structures for the ease of the reader. The normative definitions for these structures is defined 20 by the WSDL referenced in section 15.

Extensibility of all the data structures is defined using the schema syntax for including arbitrary content from other namespaces.

25 Optional parameters are marked with [O], required parameters with [R]

11.1 Handle Types

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:

- 30
- Handles may be stored in databases and may used for indexing
 - The Consumer will likely embed handles in client URLs
 - Comparison of handles should be efficient

35 The maximum handle size is restricted to 255 bytes. The Consumer MAY ignore any character in a handle that falls out of this range.

Handle **extends** String

11.2 Property Type

Properties are used to ... [RT: Intro blurbs of this type would be useful for each type]

5	Property	
	[R] String	name
	[O] String	type
	[O] String	value

Members:

- `name`: Name of the property, must not be null
- `type`: Definition of the Property's datatype (default is "xsd:string"). This data field SHOULD only be specified in the metadata that describes a property to the Consumer and not on any invocation that sets properties.
- `value`: String representation of the property's value. The interpreter is responsible for serializing and deserializing the correct value type.

11.3 PropertyDescription Type

15 This data structure is defined in section 6.1.10.

11.4 RegistrationData Type

This data structure is defined in section 5.1.1.

11.5 RegistrationCore Type

This data structure is defined in section 4.1.3.

20 11.6 RegistrationContext Type

This data structure is defined in section 4.1.4.

11.7 UserContext Type

This data structure is defined in section 4.1.2.

11.8 ServiceDescription Type

25 This data structure is defined in section 4.1.1.

11.9 EntityDescription Type

This data structure is defined in section 6.1.2.

11.10 EntityContext Type

This data structure is defined in section 6.1.5.

30 11.11 EntityResponse Type

This data structure is defined in section 6.1.6.

11.12 MarkupParams Type

This data structure is defined in section 7.1.3.

11.13 MarkupResponse Type

This data structure is defined in section 7.1.4.

11.14 InteractionParams Type

This data structure is defined in section 6.1.7.

5 11.15 InteractionResponse Type

This data structure is defined in section 6.1.8.

11.16 CacheControl Type

This data structure is defined in section 7.1.2.

11.17 Templates Type

10 This data structure is defined in section 7.1.5.

11.18 ClientData Type

The Interaction and Markup data structure types carry common information using these types.

15	ClientData
	[R] String userAgent
	[R] String deviceInfo
	[O] Object any

Members:

- `userAgent`: String identifying the UserAgent of the End-User.
- `deviceInfo`: Type of device rendering the markup to the End-User.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

11.19 User Profile Type

25	UserProfile
	[O] UserName name
	[O] Date birthdate
	[O] String gender
	[O] EmployerInfo employer
	[O] LocationInfo homeInfo
30	[O] LocationInfo workInfo
	[O] Object any

Members:

- `name`: A structure containing the various fields for the End-User's name.
- `birthdate`: The End-User's birthdate.
- `gender`: The End-User's gender ('M' = male, 'F' = female).
- `employer`: 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.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

11.20 Username Type

5	UserName		
	[O] String	prefix	
	[O] String	given	
	[O] String	family	
	[O] String	middle	
10	[O] String	suffix	
	[O] String	nickName	
	[O] Object	any	

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

11.21 EmployerInfo Type

25	Employerinfo		
	[O] String	employer	
	[O] String	department	
	[O] String	jobTitle	
	[O] Object	any	

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

11.22 LocationInfo Type

40	LocationInfo		
	[O] Address	address	
	[O] String[]	telephone	
	[O] String[]	email	
	[O] String[]	online	
	[O] Object	any	

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.
- 5 • `online`: An array of URIs for the End-User (usually web-sites).
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

11.23 Address Type

10	Address	
	[O] String	name
	[O] String[]	street
	[O] String	city
	[O] String	stateprov
15	[O] String	country
	[O] String	org
	[O] Object	any

Members:

- `name`: The name to which items should be addressed.
- 20 • `street`: The street portion of the address. This may involve multiple lines of an address.
- `city`: The city portion of the address.
- `stateprov`: The state or providence portion of the address.
- `country`: The country portion of the address.
- `org`: Any organization needing to be specified in the address.
- 25 • `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

11.24 LocaleData Type

30	LocaleData	
	[O] String	locale
	[O] String	description
	[O] String	shortTitle
	[O] String	title
	[O] String[]	keywords
35	[O] Object	any

Members:

- `locale`: The locale for this descriptive response.¹
- `description`: Descriptions of the entity. This SHOULD be displayed on selection dialogs, etc.
- 40 • `shortTitle`: A short title for the entity.

¹ <http://lcweb.loc.gov/standards/iso639-2/langcodes.html>

- `titles`: Title for the entity.
- `keywords`: Array of keywords describing the entity which can be used for search, etc.
- `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

5 11.25 EntityType Type

EntityType	
[R] Handle	entityHandle
[R] LocaleData[]	locales
[R] String[]	markupTypes
[O] String[]	markupLocales
[O] URL	wsdlURL
[O] String[]	modes
[O] String[]	windowStates
[O] String[]	producerRoles
[O] String[]	userProfileItems
[O] Integer	needSecureCommunications
[O] boolean	userContextStoredInSession
[O] boolean	templatesStoredInSession
[O] boolean	hasUserSpecificState
[O] boolean	doesUrlTemplateProcessing
[O] Object	any

Members:

- `entityHandle`: The handle by which Consumers MAY refer to this offered entity.
- `locales`: A array of locale specific data.
- `markupTypes`: The different markup languages supported by the entity, e.g. *HTML*, *XHTML*, *WML*, *VoiceXML*, *cHTML*, ...
- `markupLocales`: The list of locales for which this entity is willing to generate markup.
- `wsdlURL`: The URL for the WSDL description of this entity.
- `modes`: The modes that are supported by the entity (e.g. *view*, *edit*, *config*, *help*, *design*, *preview*). Mode constants are defined elsewhere in this section. In addition the portlet can define its own modes.
- `windowStates`: The windowStates that are supported by the entity (e.g. *minimized*, *normal*, *maximized*, ...). These constants are is defined elsewhere in this section.
- `producerRoles`: Array of localized names for the roles the entity can manage. Note: This support MAY be provided by the Producer service on behalf of the entity. The entity can freely define any role it wants, however there exists a set of predefined roles in this section. [\[R416\]](#)
- `userProfileItems`: An array of xpath-like strings that enumerate what portions of the `UserContext` structure the entity needs to provide full functionality. For the fields this specification defines, the named profile items an entity uses MUST all come from the "Profile Name" column of the table found in section 10.
- `needSecureCommunications` Flag that indicates whether this entity requires secure communications on all hops from the End-User to the entity for either its markup or user interactions. Possible values are: 0 = secure client communications not needed (default value), 1 = secure client communications needed by some markup or user interactions, and 2 = secure client communications required for all markup and user interactions.

- 5 • `userContextStoredInSession`: A flag indicating the entity 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 communications be used when the information is passed. Not requiring this of all invocations can result in a significant performance difference. The default value of this flag is 'false'.
- 10 • `templatesStoredInSession`: A flag indicating the entity will store any supplied `templates` in the current session. Setting this flag to 'true' allows the Consumer to optimize when the `templates` structure is included on **getMarkup()** invocations. Since the data since of the `templates` structure can get quite large, not requiring it to be passed on all **getMarkup()** invocations can result in a significant performance difference. The default value of this flag is 'false'.
- 15 • `hasUserSpecificState`: A flag indicating the entity will store persistent state specific to each End-User. For entities setting this flag to true, Consumers MAY choose to clone the entity when it is placed on an aggregated page rather than waiting for the fault message processing described in section 7.3.1. The default value of this flag is 'false'.
- 20 • `doesUrlTemplateProcessing`: A flag indicating the entity will process any `templates` supplied so as to correctly write URLs in its markup. For entities setting this flag to true, Consumers MAY choose to provide the URL writing templates and thereby reduce the number of time the Consumer parses markup in order to rewrite URLs. The default value of this flag is 'false'.
- 25 • `any`: A schema declaration that implementations MAY choose to extend this structure provided those extensions come from a different namespace.

11.26 RoleDescription Type

RoleDescription	
[R] String	name
[R] String	description
[O] Object	any

- Members:**
- 35 `name` The name for this role.
 - `description` A free form description of the role. Expected use of this field is display at the Consumer to someone who will provide a mapping to Consumer supported roles.
 - `extensions` An array of properties for use when implementations choose to extend this structure.

12 Producer Roles

Administrator	This role typically grants the highest level of access to the functionality of an entity.
User	This role is typically associated with End-Users who may personalize some set of properties for an entity.
Guest	This role is typically associated with End-Users who may view an entity on a page but not modify any of it's properties or settings.

13 Constants

Type	Value	Description
Mode	VIEW_MODE	Entity is expected to render markup reflecting its current state.
Mode	EDIT_MODE	Entity is expected to render markup useful for End-User personalization.
Mode	CONFIG_MODE	Entity is expected to render markup useful for setting its configuration.
Mode	HELP_MODE	Entity is expected to render markup useful for helping an End-User understand the entity's operation.
Mode	PREVIEW_MODE	Entity is expected to render markup representative of its configuration as might be useful to someone testing a page layout.
Mode	PREVIOUS_MODE	Entity requesting the Consumer to return to the previous mode.
Window state	VIEW_NORMAL	The entity is sharing space with other entities and should restrict its consumption of space accordingly.
Window state	VIEW_MINIMIZED	The entity is sharing space with other entities and should restrict its consumption of space accordingly.
Window state	VIEW_MAXIMIZED	The entity is being offered significantly more than the normal share of the space available to entities on the Consumer's aggregated page.
Window state	VIEW_DETACHED	The entity's markup will be detached in a separate window (could be just a virtual window) from any other entity's markup.
Window state	VIEW_PREVIOUS	Entity requesting the Consumer to return to the previous window state.
Cache Hint	CACHE_USER	The returned markup depends on the <code>userContext</code> such that any change to the <code>userContext</code> MUST invalidate any Consumer caching of the markup.
Cache Hint	CACHE_REGISTRATION	The returned markup depends on the <code>registrationContext</code> such that any change to the <code>registrationContext</code> MUST invalidate any Consumer caching of the markup.
Cache Hint	CACHE_MARKUP_PARAMS	The returned markup depends on the <code>markupParams</code> such that any change to the <code>markupParams</code> MUST invalidate any Consumer caching of the markup.

14 Fault Messages

In addition to generic fault messages that may be generated by the web service stacks of the Consumer and/or Producer, a variety of messages specific to this protocol are defined.

[RT: This will need significant work ... likely over the next 2-4 weeks. SOAP v1.2 defines a fault message format of the form:

```
<Fault>
  <Code>
    <Value/>
    <Subcode>
      <Value/>
      <!--Optional nesting of another Subcode element -->
      <Subcode>
    </Code>
    <Reason/>
    <!--optional Node and Role elements -->
    <Detail/>
  </Fault>
]
```

15 WSDL Interface Definition

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

<http://www.oasis-open.org/committees/wsrp/wsd/v1/WSRP-v1-Interfaces.wsdl> - Contains the messages and portType definitions for this specification.

25 <http://www.oasis-open.org/committees/wsrp/wsd/v1/WSRP-v1-Bindings.wsdl> - Contains 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.
- 30 3. WSRP.v1.Registration.PortType: Only Producers supporting in-band registration of Consumers need expose this portType.
- 35 4. WSRP.v1.EntityManagement.PortType: Producers supporting the entity management interface expose this portType. If this portType is not exposed, the entities of the service are not configurable by Consumers.

16 References

16.1 Normative

[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.

5 16.2 Non-Normative

[J2EE] <http://java.sun.com/j2ee/>

[.Net] <http://www.microsoft.com/net/>

[WSDL] <http://www.w3.org/TR/wsdl>

[SOAP] <http://www.w3.org/TR/SOAP/>

10 [UDDI] <http://www.uddi.org/specification.html>

[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->

15 [open.org/committees/wsrp/documents/wsrp_wp_09_22_2002.pdf](http://www.oasis-open.org/committees/wsrp/documents/wsrp_wp_09_22_2002.pdf), 22
September, 2002.

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

Appendix A. Glossary

<p>Access</p>	<ol style="list-style-type: none"> 1. To interact with a system entity in order to manipulate, use, gain knowledge of, and/or obtain a representation of some or all of a system entity's resources. [1] 2. in the system domain, what an End user does to a Web site using a browser, or what a Client does to a Web service; 3. in the business domain, what a Consumer does to a Web service or a Web site hosted by a Producer;
<p>Access Control</p>	<p>Protection of resources against unauthorized access; a process by which use of resources is regulated according to a security policy and is permitted by only authorized system entities according to that policy.</p>
<p>Access Rights</p>	<p>A description of the type of authorized interactions a subject can have with a resource. Examples include read, write, execute, add, modify, and delete.</p>
<p>Account</p>	<p>The set of attributes that together define a user's access to a given service. Each service may define a unique set of attributes to define an account. An account defines user or system access to a resource or service.</p> <p>A means of supporting a hierarchy of adaptations or properties related to portlet invocation for the consuming portal</p>
<p>Action</p>	<p>A notification that your state has changed.</p>
<p>Administrator</p>	<p>A person who installs or maintains a system (for example, a SAML-based security system) or who uses it to manage system entities, users, and/or content (as opposed to application purposes; see also End User). An administrator is typically affiliated with a particular administrative domain and may be affiliated with more than one administrative domain.</p>
<p>Anonymity</p>	<p>The quality or state of being anonymous, which is the condition of having a name or identity that is unknown or concealed. [1]</p>
<p>Attribute Also see 'Service Attribute'</p>	<p>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.</p>
<p>Authentication</p>	<p>To confirm a system entity's asserted principal identity with a specified, or understood, level of confidence.</p>
<p>Authorization</p>	<p>The process of determining, by evaluating applicable access control information, whether a subject is allowed to have the specified types of access to a particular resource. Usually, authorization is in the context of authentication. Once a subject is authenticated, it may be authorized to perform different types of access.</p>

Browser	A system entity that is used by an end user to access a Web site. A browser provides a run-time environment for distributed application components on the client's device.
Client	a system entity (not a business entity) that accesses a Web service. Contrast with Browser and Customer.
Company	Any organizational entity
Consumer	A business entity that accesses a Web service or a Web site. Contrast with End user and Customer A business entity creating Consumer Applications
Consumer Application	A web application that uses one or more WSIA Web Services
Credential	Data that is transferred to establish a claimed principal identity. [4]
Customer	A business entity that purchases goods or services
End-User	1. A natural person who makes use of resources for application purposes (as opposed to system management purposes; see Administrator, User). [4] 2. A person who uses a device specific Browser to access a Web site
Event	A notification that some state in the system (that you are interested in) has changed
Host (verb)	to run an application on an execution platform, which typically consists of hardware and software
Fragment	A piece of markup that is not part of a full document - part of aggregate - not binary, but not necessarily XML - generally a markup language - can aggregate a bunch of fragments
Identity	The unique identifier for a person, organization, resource, or service.
Login, Logon, Sign-On	The process whereby a user presents credentials to an authentication authority, establishes a simple session, and optionally establishes a rich session.
Logout, Logoff, Sign-Off	The process of presenting credentials to an authentication authority, establishing a simple session, and optionally establishing a rich session.
Party	Refers to any person who interacts with the system and/or the network the system is managing.
Portal Application	Component that is the controlling application and is responsible for aggregating portlet content and displaying the portal page
Portal Modes	View, edit, help config, is under debate

Portal Page	Complete document rendered by a portal
Portlet	Component that generates fragment
Portlet Application	The equivalent of the WAR file
Portlet Class	Implementation of portlet as a Java class (compiled code)
Portlet Container	Environment where portlets run (lifecycle, security)
Portlet Content	What the portlet renders without controls that decorate it (fragment that the portlet creates)
Portlet Instance	Portlet object with given user configuration; essentially the handle
Portlet Object	Instance of portlet class (no defined portal state)
Portlet Window	Portlet has a set of controls that decorate it
Portlet Window Instance	Instantiation of a portlet on a page in a portlet window
Principal	A system entity whose identity can be authenticated.
Producer	A business entity that hosts a Web service or a Web site One or more WSIA web services A business entity creating, publishing and supporting WSIA Web Services
Provider	A business entity that sells access to or use of Web services
Pull	To actively request information from a system entity.
Push	To provide information to a system entity that did not actively request it.
Role	The combination of access rights available to a particular actor.
Service	1. A specific type of resource that is not physically obtained by a user, but is accessed periodically by the user. [4] 2. See Web Service
Service Attribute	Characteristics or qualifiers of a service – which describe details like type of encoding, network address, mailbox size for email, storage space for backup, and so on.
Service Offer	The unique combination of service attributes and service options that is provisioned to an identity
Service Option	The choices available within a service – which could be custom configured by the service provider as opposed to a service attribute which is inherent to the service. For example, a Gold Option and a Silver Option – which have to be part of the provisioning data.
Service Provider	The organizational entity that provides the service
Session	A lasting interaction between system entities, often involving a user, typified by the maintenance of some state of the interaction for the duration of the interaction.

Site	<p>An informal term for an administrative domain in geographical or DNS name sense. It may refer to a particular geographical or topological portion of an administrative domain, or it may encompass multiple administrative domains, as may be the case at an ASP site.</p> <p>one portal-specific example of an administrative domain, user group, etc.</p>
System / System Entity	<p>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.</p>
Time-Out	<p>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".</p>
Uniform Resource Locator (URL)	<p>Defined as "a compact string representation for a resource available via the Internet." URLs are a subset of URI.</p>
User	<p>A natural person who makes use of a system and its resources for any purpose. See also administrator, end user.</p> <p>A natural person who makes use of a system and its resources for any purpose. See also end user.</p>
Username/User Identity	<p>The unique identity for a user with a system</p>
Web Service	<p>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.</p>
WSIA Web Service	<p>A SOAP-compliant Web Service that adheres to one or more WSIA interfaces.</p>
Web Site	<p>A hosted application that can be accessed by an End user using a browser</p>
Window States	<p>Max, min, normal, detached</p>
WSIA Interface	<p>A programmatic interface defined by the WSIA committee to support the creation of Web Services that encapsulate and integrate user-facing interactive applications.</p>
WSRP Service	<p>Presentation oriented, interactive web services that can be aggregated by consuming applications</p> <p>- WSRP services can be published, found, and bound in a standard manner, describing themselves with standardized metadata</p>
XML (Extensible Markup Language)	<p>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]</p>
XML Namespace	<p>A collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names. An XML</p>

	namespace is often associated with an XML schema. For example, SAML defines two schemas, and each has a unique XML namespace.
--	---

Appendix B. Acknowledgments

The following individuals were members of the WSIA committee during the development of this specification:

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

50

The following individuals were members of the WSRP committee during the development of this specification:

- Alejandro Abdelnur, Sun
- Olin Atkinson, Novell
- 5 • Sasha Aickin, Plumtree
- Jeff Broberg, Novell
- Chris Brown, Novell 45
- Mark Cassidy, Netegrity
- Richard Cieply, IBM
- 10 • Dave Clegg, Sybase
- Ugo Corda, SeeBeyond
- William Cox, BEA 50
- Michael C. Daconta, McDonald Bradley
- Ron Daniel Jr., Interwoven
- 15 • Brian Dirking, Stellent
- Angel Luis Diaz, IBM
- Gino Filicetti, Bowstreet 55
- Adrian Fletcher, BEA
- Michael Freedman, Oracle
- 20 • Tim Granshaw, SAP Portals
- Mike Hillerman, Peoplesoft
- Scott Huddleston, Divine
- Timothy N. Jones, CrossWeave
- Andre Kramer, Citrix
- 25 • Andreas Kuehne
- Aditi Karandika, France Telecom
- Alan Kropp, Epicentric
- Jon Klein, Reed-Elisvier
- Andreas Kuehne
- 30 • Carsten Leue, IBM
- Susan Levine, Peoplesoft
- Eric van Lydegraf, Kinzan
- Khurram Mahmood, Peoplesoft
- Lothar Merk, IBM
- 35 • Madoka Mitsuoka, Fujitsu
- Takao Mohri, Fujitsu
- Adam Nolen, Reed-Elisvier
- Petr Palas, Moravia IT
- Gregory Pavlik, HP
- 40 • Peter J Quintas, Divine
- Nigel Ratcliffe, Factiva
- Eilon Reshef, WebCollage
- Mark Rosenberg, Tibco
- Joe Rudnicki, U.S. Department Of the Navy
- Thomas Schaeck, IBM (chair)
- Robert Serr, Divine
- Davanum Srinivas, Computer Associates
- Andrew Sweet, Perficient
- David Taieb, IBM 50
- Yossi Tamari, SAP Portals
- Rich Thompson, IBM
- Srinivas Vadhri, CommerceOne
- Stephen A. White, SeeBeyond
- 55 • Charles Wiecha, IBM

Appendix C. Revision History

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 F2F discussion/decisions Rewrite Intro Propose caching support Propose Properties support

Appendix D. Notices

59 OASIS takes no position regarding the validity or scope of any intellectual property or other
60 rights that might be claimed to pertain to the implementation or use of the technology described
61 in this document or the extent to which any license under such rights might or might not be
62 available; neither does it represent that it has made any effort to identify any such rights.
63 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
64 at the OASIS website. Copies of claims of rights made available for publication and any
65 assurances of licenses to be made available, or the result of an attempt made to obtain a
66 general license or permission for the use of such proprietary rights by implementors or users of
67 this specification, can be obtained from the OASIS Executive Director.

68 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
69 applications, or other proprietary rights which may cover technology that may be required to
70 implement this specification. Please address the information to the OASIS Executive Director.

71 Copyright © The Organization for the Advancement of Structured Information Standards
72 [OASIS] 2001. All Rights Reserved.

73 This document and translations of it may be copied and furnished to others, and derivative
74 works that comment on or otherwise explain it or assist in its implementation may be prepared,
75 copied, published and distributed, in whole or in part, without restriction of any kind, provided
76 that the above copyright notice and this paragraph are included on all such copies and
77 derivative works. However, this document itself does not be modified in any way, such as by
78 removing the copyright notice or references to OASIS, except as needed for the purpose of
79 developing OASIS specifications, in which case the procedures for copyrights defined in the
80 OASIS Intellectual Property Rights document must be followed, or as required to translate it
81 into languages other than English.

82 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
83 successors or assigns.

84 This document and the information contained herein is provided on an "AS IS" basis and
85 OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT
86 LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT
87 INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR
88 FITNESS FOR A PARTICULAR PURPOSE.