Open Services for Lifecycle Collaboration
Core Specification Version 2.0

This Version
- http://open-services.net/bin/view/Main/OslcCoreSpecification

Latest Version
- http://open-services.net/bin/view/Main/OslcCoreSpecification

Authors
- Dave Johnson, Steve Speicher

Contributors
- The OSLC Core Specification Workgroup

Table of Contents

- Overview
  - Design considerations
  - Glossary of terms
- OSLC Defined Resources
  - OSLC Defined Resources
  - Unknown properties and content
  - Resource Operations
  - Resource Paging
  - Selective Property Values
  - Common Properties
- Service Provider Resources
  - Conceptual Model
- Creation Factories
  - Creating an OSLC Defined Resource
- Query Capabilities
  - Conceptual Model
  - Query Syntax
- Delegated User Interface Dialogs
  - Post Message and Window Name protocols
  - Resource Selection
  - Resource Creation
  - Dialog Resizing
- User Interface Previews
- Authentication
  - HTTP Basic Authentication
  - OAuth Authentication
  - Form Based Authentication
- Error Responses
  - Conceptual Model
- Specification Versioning
Overview

(this section is informative)

The Open Services for Lifecycle Collaboration (OSLC) initiative is creating a family of web services specifications for products, services and other tools that support all phases of the software and product lifecycle. The purpose of these specifications is to enable integration between products that support Application Life-cycle Management (ALM) and Product Life-cycle Management (PLM). Each part of the lifecycle or domain has its own group and specification, for example there are Change Management, Quality Management, Estimation & Measurement and more. Each of the domain specifications are built upon this core specification.

This OSLC Core Specification sets out the common features that every OSLC Service can be expected to support using terminology from the World Wide Web Consortium (W3C). New terminology that we introduce can be found in the glossary section below. This specification is mostly about OSLC Services, it specifies what OSLC Services MUST, SHOULD and MAY do. It also contains some required behaviors for OSLC clients and rules for OSLC domain specifications that extend this specification.

OSLC Services are accessible via a Service Provider Resource that describes the Services offered. Each Service can provide Creation Factories for resource creation, Query Capabilities for resource query and Delegated UI Dialogs to enable clients to create and select resources via a web UI. Query Capabilities and Creation Factories may offer Resource Shapes that describe the properties of resources managed by the service. This is illustrated in the diagram below. See the section below on Service Provider Resources for further discussion of these concepts.

Figure #1: OSLC Core Specification concepts and relationships
This specification establishes terminology and rules for defining resources in terms of the property names and value-types that are allowed and required. OSLC domain specifications are expected to use these rules and terminologies to describe their resources. See the OSLC Defined Resources section for more on this topic.

This specification also sets out rules for creating resource representations in RDF/XML, JSON, Atom and Turtle formats. OSLC domain specifications are expected to refer to these rules when specifying how their resources are to be represented. See the OSLC Defined Resource Representations section for the representation rules and examples of each format.

**About the version number.** We use the version number "2.0" even though there has never been an OSLC Core Version 1.0 specification. We do this because this OSLC Core specification was written after a series of version 1.0 domain specifications were finalized by OSLC workgroups. The version 2.0 domain specifications will all be based on this Core specification and to avoid confusion this specification will also be known as Version 2.0.

**About RDF.** The resource and property-based data model used in OSLC resources is based on the Resource Description Framework (RDF) and OSLC requires RDF representations, but OSLC uses a small number of RDF concepts and does not require implementers to provide an RDF triple-store or a SPARQL query-engine.

**Design considerations**

The core philosophies of OSLC are to build on the powerful and scalable architecture of the World Wide Web and to do the simplest possible things that will work.

**Build on the WWW.** OSLC builds on the architecture of the WWW and follows the REST architectural pattern. This means that OSLC Services provide a uniform HTTP interface, OSLC URIs are stable and opaque and, in simple terms, OSLC works like the web.

**Keep things simple.** Doing the simplest things that will possibly work means a couple of different things in regard to OSLC. It means starting with simple and existing concepts. For example, we model everything as resources with property values and do not stray from that model. Keeping things simple also means building on established and well-known specifications, but also carefully limiting the number of other specifications that we reference. This simplicity is intended to enable loose coupling and to make life easier for everybody: OSLC domain work groups, OSLC Service implementers and OSLC client developers.
Accommodate different schemas. Because of the breadth of the OSLC domains, spanning lifecycle and platforms, OSLC has to work for systems with very different data schemas or no schemas at all. Flexibility is needed, but some OSLC Services must be able to offer information about resource properties so that clients can learn which are allowed and required for resource creation, query and reporting.

Accommodate different representations. Different client platforms might require or at least prefer different representations. For example, in the browser a JSON or Atom format representation might work best. OSLC Services will all support RDF/XML and may support other formats including JSON, Atom and Turtle.

Align with the W3C Linked Data initiative. Instead of defining a new data model, OSLC’s resource and property-value approach is based on industry standard Resource Description Framework (RDF) data model. This model allows OSLC to keep things simple, build on the WWW and accommodate different schemas.

Glossary of terms

This is a guide to some of the terminology used in this document. The following definitions are standard W3C concepts. OSLC uses these concepts without modification – their definitions are summarized here for the convenience of the reader. See http://www.w3c.org.

- **Resource**: A network data object or service that can be identified by a URI. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, resolutions) or vary in other ways. (reference: HTTP)

- **Representation**: An HTTP payload entity, included in an HTTP response, that is subject to content negotiation. There may exist multiple representations associated with a particular HTTP response status. (reference: HTTP)

- **URI**: Uniform Resource Identifiers are simply formatted strings which identify – via name, location, or any other characteristic -- a resource (reference: URI Syntax)

Here are the OSLC specific terms used in this specification.

(These definitions are normative)

- **OSLC Domain**: an OSLC Domain is one ALM or PLM topic area such as Change Management, Requirements management or Automation. Each OSLC Domain will have its own OSLC specification that complies with this Core specification.

- **OSLC Service**: a set of capabilities that enable a web client to create, retrieve, update and delete resources managed by an ALM or PLM product or online service offering and associated with one OSLC Domain.

- **OSLC Service Provider**: a product or online service offering that provides an implementation of one or more OSLC Services, which may themselves implement different OSLC Domain specifications.

- **OSLC Resource**: a resource that is managed by an OSLC Service, may have properties and may link to other resources including those provided by other OSLC Services.

- **OSLC Defined Resource**: a resource that is defined by an OSLC specification, see OSLC Defined Resources below.

- **OSLC Defined Properties**: resource properties that are defined by an OSLC specification, defined by an OSLC Resource Shape or both.

- **OSLC Resource Shape**: defines the set of OSLC Properties expected in a resource for specific operations (i.e. creation, update or query) for each their value types, allowed values, cardinality and optionality. Examples of such operations include OSLC Creation Resource and Query Resource. Other examples might include applications that display data in tables.

- **OSLC Creation Factory**: A creation factory provides a URI used to create new resources via HTTP POST and may also provide Resource Shapes that describe the types of resources that may be created.

- **OSLC Query Capability**: A query capability provides a base URI for forming query resource URIs and, optionally, Resource Shapes that describe the property values that may be used in query expressions and returned in query results.
OSLC Response Info Resource: An OSLC Defined Resource that provides information about a paged resource representation, e.g. the next page in a multi-page query result representation.

Here are some industry terms that we use in this specification:

- **Application Lifecycle Management (ALM):** ALM is the marriage of business management to software engineering made possible by tools that facilitate and integrate requirements management, architecture, coding, testing, tracking, and release management ([http://en.wikipedia.org/wiki/Application_lifecycle_management](http://en.wikipedia.org/wiki/Application_lifecycle_management)).

- **Product Lifecycle Management (PLM):** In industry, product lifecycle management (PLM) is the process of managing the entire lifecycle of a product from its conception, through design and manufacture, to service and disposal ([http://en.wikipedia.org/wiki/Product_lifecycle_management](http://en.wikipedia.org/wiki/Product_lifecycle_management)).

And finally, we use the following two terms to describe portions of the OSLC Core specification:

- **Normative.** Normative sections of this document set forth requirements that must be met to establish conformance with the OSLC Core specification; or providing recommendations or optional courses of action. This is done using use words from [RFC-2119](http://tools.ietf.org/html/rfc2119) (e.g. MUST, SHOULD, MAY, etc.). For recommended or optional features, conformance is not dependent on the fact of implementation, but, if implemented, that implementation is as prescribed in this specification. Contrast with informative.

- **Informative.** Informative text provides background or explanation. Contrast with Normative. It should be clear which sentences and paragraphs are informative from the context and the absence of RFC-2119 keywords. Sections that are purely informative will be marked as informative. Informative text that might be mistaken for normative will also be marked.

### OSLC Defined Resources

An OSLC Resource is a resource managed by an OSLC Service. An OSLC Resource is typically something like a Change Request, a Requirement or some other ALM or PLM artifact or record, but an OSLC Resource could also be a video or a presentation file. The resource’s storage medium is unconstrained by OSLC, e.g. it could be stored in a relational database, a flat-file on disk, a source code control system, or in any other way.

An OSLC Service can manage any type of resource; OSLC specifications only constrain an OSLC Service’s behavior with respect to resource types it manages that are defined by OSLC specifications (OSLC Defined Resources). OSLC Defined Resources may be specified in any OSLC specification, including this document. Resources are defined by the properties that are allowed and required in the resource.

### OSLC Defined Resources

OSLC uses a simple model of resources with property values intended to be consistent with the Resource Description Framework (RDF) data model ([reference: RDF Concepts](http://www.w3.org/TR/rdf-concepts/)). OSLC also builds upon the Extensible Markup Language (XML) namespace mechanism ([reference: XML Namespaces](http://www.w3.org/TR/REC-xml-names/)).

When specifying a resource or a property, OSLC Specifications define its type as a URI which can be decomposed into a namespace URI and a name. We abbreviate type URIs as Prefixed Names ([reference: Prefixed Names](http://www.w3.org/Submission/2002/ES/Prefixed Names/)), which are represented in XML as QNames. The namespace used for resources defined in this specification is defined as follows:

- **Namespace URI:** [http://open-services.net/ns/core#](http://open-services.net/ns/core#)
- **Default Prefix:** oslc

When defining an OSLC Resource type, OSLC Specifications **MUST** provide the following information:

- **Name (String):** name of the resource which **MUST** be valid as the Local Name part of a QName ([reference: XML Namespaces](http://www.w3.org/TR/REC-xml-names/)).
- **URI (URI):** The URI of the resource definition. Per the rules of Prefixed Names, this URI is formed by appending the Name to the end of the Namespace URI in the specification that defines the resource. For example, the resource named Service (defined below) gets the Type URI of [http://open-services.net/ns/core#Service](http://open-services.net/ns/ns/core#Service).
Once a resource type is defined, its allowed and required properties can be defined.

Regardless of any property definitions, providers and clients MAY impose implementation-specific limits on resources they accept. For example, they are not required to accept/create/store resources whose RDF triples contain objects with arbitrarily large literal values.

## Defining OSLC Properties

OSLC Specifications MAY provide a list of properties allowed and/or required for a particular domain and operation on an OSLC Defined Resource; if no operation is specified, then the list applies to all operations governed by that specification. Specifications that provide a list of properties and constraints on them SHOULD provide the following information for each property that they define.

- **Name**: name of the property which MUST be valid as the Local Name part of a QName (reference: XML Namespaces).
- **URI**: The URI that identifies the property. The URI is formed by appending the Name to the end of the Namespace URI associated with the property. For example, the resource named `oslc:ServiceProviderCatalog` (defined below in the Service Providers Section) defines a property named `domain` with the URI of `http://open-services.net/ns/core#domain`.
- **Description**: Description of the property.
- **Occurs**: value MUST be one of:
  - `http://open-services.net/ns/core#Exactly-one`
  - `http://open-services.net/ns/core#Zero-or-one`
  - `http://open-services.net/ns/core#Zero-or-many`
  - `http://open-services.net/ns/core#One-or-many`
- **Value-types**: A property MAY allow multiple value-types and a value MUST satisfy one or more of them. Each value-type MUST be a URI that corresponds to one of the following:
  - **Literal value-types**:
    - **Boolean**: a boolean type as specified by XSD Boolean (`http://www.w3.org/2001/XMLSchema#boolean`, reference: XSD Datatypes).
    - **DateTime**: a Date and Time type as specified by XSD dateTime (`http://www.w3.org/2001/XMLSchema#dateTime`, reference: XSD Datatypes).
    - **Decimal**: a decimal number type as specified by XSD Decimal (`http://www.w3.org/2001/XMLSchema#decimal`, reference: XSD Datatypes).
    - **Double**: a double floating-point number type as specified by XSD Double (`http://www.w3.org/2001/XMLSchema#double`, reference: XSD Datatypes).
    - **Float**: a floating-point number type as specified by XSD Float (`http://www.w3.org/2001/XMLSchema#float`, reference: XSD Datatypes).
    - **Integer**: an integer number type as specified by XSD Integer (`http://www.w3.org/2001/XMLSchema#integer`, reference: XSD Datatypes).
    - **String**: a string type as specified by XSD String (`http://www.w3.org/2001/XMLSchema#string`, reference: XSD Datatypes).
    - **XMLLiteral**: a Literal XML value (`http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral`).
  - **Resource value-types**:
    - **Resource**: value is a resource at a specified URI (i.e. a URI Reference) (`http://open-services.net/ns/core#Resource`).
    - **Local Resource**: value is an resource available only inside the resource being defined (i.e. a Blank Node) (`http://open-services.net/ns/core#LocalResource`).
    - **AnyResource**: value is either a Resource or Local Resource as defined above (`http://open-services.net/ns/core#AnyResource`).
  - **Representation**: for properties with a resource value-type, OSLC specifications should also specify how the resource will be represented. The options are `http://open-services.net/ns/core#Reference`, `http://open-services.net/ns/core#Inline` or `http://open-services.net/ns/core#Either`.
  - **Range**: for properties with a resource value-type, OSLC specifications should follow the best practices in Appendix C.
Guidance on Links and Relationships, which usually results in no restrictions on the range of possible resource types allowed, and an informative recommendation in the property description suggesting which resource types implementations should expect to find. This can be specified as a list of one or more resource types specified by URI reference; when no restrictions are required, use the string any. Clients SHOULD allow any resource type as the target of a link. Providers are strongly RECOMMENDED to behave reasonably for all resource types listed in a property’s description, and to degrade gracefully for others, as defined in Appendix C.

- **Read-only**: Boolean indication of whether or not clients are permitted to replace the property’s value after the resource has been created. Allowable values are: true, false, unspecified.
  - **True** indicates that providers **SHOULD NOT** permit clients to change the property’s value after the resource has been created.
  - **False** indicates that providers **MAY** permit clients to change the property’s value after the resource has been created.
  - **Unspecified** indicates that the domain specification leaves the choice up to provider implementations.

In the rest of this document we will define OSLC resources as described above. The below section titled OSLC Defined Resource Representations defines how OSLC resources are to be represented in RDF/XML, JSON and other formats.

OSLC Services that wish to provide the information above in a machine-readable format **MAY** use OSLC Resource Shapes, see Appendix A: Common Properties and Resources for more information.

**NOTE**: we do not mention Internationalization of strings here because we expect standard HTTP content-negotiation and representation (e.g. `xml:lang`) mechanisms to be used for such.

### Unknown properties and content

For OSLC Defined Resources, clients **SHOULD** assume that an OSLC Service will discard unknown property values. An OSLC Service **MAY** discard property values that are not part of the resource definition or Resource Shape known by the server. If a client needs verification that the requested update was accepted it **SHOULD** note the HTTP response header ETag value returned with the HTTP PUT, Immediately HTTP GET the resource back and compare the HTTP response header ETag value and contents with its expectations. The Service **SHOULD NOT** return an error code for unrecognized content. A Service **MUST** return an error code if recognized content is invalid.

The rule is different for clients. When doing an update, OSLC clients **MUST** preserve any unknown property-values and other content in OSLC Defined Resources.

See following section on Resource Update.

### Resource Operations

OSLC Services use HTTP for create, retrieve, update and delete operations on resources. OSLC Services **MUST** comply with the HTTP specification (reference: HTTP).

#### Resource Creation

(this section is informative)

To create an OSLC Defined Resource, or any type of resource managed by an OSLC Service, an OSLC client HTTP POSTs a representation of that resource to a Creation URI. See the section on Creation Factories for more information.

#### Resource Removal

(this section is informative)

To delete an OSLC Defined Resource, or any type of resource managed by a service, a client performs an HTTP DELETE on the resource's URI.
Resource Update

To update an OSLC Resource in an OSLC Service, a client fetches a representation of that resource via HTTP GET. The client updates the representation and then uses HTTP PUT to send the new representation to the resource's URI.

Recall from OSLC Defined Resources: Unknown properties and content above that, when doing an update, OSLC clients must preserve any unknown property-values and other content in OSLC Defined Resources.

Because the update process involves first getting a resource, modifying it and then later putting it back to the server there is the possibility of a conflict, e.g. some other client may have have updated the resource since the GET. To mitigate this problem, OSLC Services SHOULD use the HTTP If-Match header:

- If the HTTP If-Match header is missing OSLC Services SHOULD return HTTP Bad Request (400) status code to indicate that the header is required.
- If the HTTP If-Match header is present OSLC Services MUST behave as described in the HTTP specification, returning an HTTP Precondition Failed (412) error to indicate that the header does not match.
- If the HTTP If-Match header is present and it matches, but there is some other problem or conflict with the update then OSLC Services MAY return an HTTP Conflict (409) to indicate that problem.

Note that section Error Responses below, we specify that when an error occurs and useful information can be provided to clients OSLC Services SHOULD return error information in the body of the response.

Resource Paging

OSLC Services MAY support a technique called Resource Paging to enable clients to retrieve resources one page at a time.

When a client requests a resource, the client can expect that the entire resource will be returned in the response, with all property values. This can be problematic because, in some cases, resources may be so large that a client might not want to retrieve the entire resource in one HTTP response.

One solution for response size-sensitive Clients is to check size before loading. Clients that do not wish to load large resources can use the HTTP HEAD method to determine the size of a resource and, according to the rules of HTTP the server's SHOULD include an HTTP Content-Length header that indicates the size of the resource as the "decimal number of OCTETs." If the size is too large, a client can choose not to retrieve the resource.

Another solution is to use Resource Paging; here's how it works. To get a paged version of a resource, a client adds the "key=value" pair oslc.paging=true to the query component of the resource URI and the server MAY respond by returning a representation that contains partial information about the resource; only a subset of the resource's property values.

When a page is returned and it is NOT the last page in the sequence, then it SHOULD include an oslc:ResponseInfo (defined below), which contains a property oslc:nextPage that links to a resource that represents the next page of property-values. Because paging is unstable (see below), by the time a client follows an oslc:nextPage link there may no longe be a next page, in this case the server MAY respond with an HTTP 404 Page Not Found status code.

A client can also request paging by adding the "key=value" pair oslc.pageSize to the query string component of the resource URI. By adding this, a client requests that the server respond with a specific number of property values. For example, oslc.pageSize=20 indicates to the server that the client would like 20 values per page. OSLC Services MAY ignore oslc.pageSize.

When Resource Paging is used, the values of a multi-valued property MAY be split across resource pages. Each property value MUST be represented in its entirety and not split across multiple partial resource pages.

Provider response-size limitations

When a client requests a resource that an OSLC Service considers to be too large to return in one response and the client has not indicated that it desires paging (via oslc.paging or oslc.pageSize), the OSLC Service MAY redirect the client to a representation that
The OSLC Service receives an HTTP GET request for a resource that exceeds size limits and URL does not include `oslc.paging` or an `oslc.pageSize` key/value pair.

- The OSLC Service returns an HTTP Status 302 redirect a URL that does include the key/values for paging, as follows:
  - If the client did not indicate paging, the new redirect URL **MUST** include the `oslc.paging` pair.
  - If the client indicated a page size, then the redirect URL **MUST** include the `oslc.pageSize` pair with a size value that is acceptable to the service.
- The client **MAY** choose to follow the redirect and receive a representation that contains partial information about the resource.

On receiving a resource representation, OSLC Clients **SHOULD** check for the presence of an `oslc:nextPage` value to determine if the representation contains partial information about the resource. If the value is present, then paging is in effect and the representation contains partial information about the resource.

### Unstable Paging

Because HTTP is a stateless protocol and OSLC Services manage resources that can change frequently, OSLC clients **SHOULD** assume that resources can change as they page through them using the `oslc:nextPage` mechanism.

### Stable Paging

Some OSLC Services might wish to guarantee stable paging, meaning that the chain of `oslc:nextPage` links in a resource represent a snapshot in time and will not change as the client pages through them. OSLC specifications that require stable paging **SHOULD** state this requirement and specify to which resources it applies.

Note that because stable paging implementations are based on server-side state, it is possible that such state will expire. Implementations **MAY** use the HTTP response code 410 (Expired) to indicate to clients that the next-page link they requested has expired.

### Response Information

Resource representations returned via Resource Paging **MUST** include a resource of type `oslc:ResponseInfo`, as defined in this section. A response info resource representation describes information about a paged HTTP response body in which it appears.

- **Name**: `ResponseInfo`
- **URI**: `http://open-services.net/ns/core#ResponseInfo`
<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>zero-or-one</td>
<td>True</td>
<td>XML</td>
<td>Literal</td>
<td>n/a</td>
<td>Title of the response</td>
</tr>
<tr>
<td>dcterms:description</td>
<td>zero-or-one</td>
<td>True</td>
<td>XML</td>
<td>Literal</td>
<td>n/a</td>
<td>Description of response</td>
</tr>
<tr>
<td>oslc:nextPage</td>
<td>zero-or-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>any</td>
<td>Link to next page of response</td>
</tr>
<tr>
<td>oslc:totalCount</td>
<td>zero-or-one</td>
<td>True</td>
<td>Integer</td>
<td>n/a</td>
<td>n/a</td>
<td>This optional property indicates the total number of results across all pages, its value should be non-negative. In the context of a query resource, this value SHOULD be the total number of results, i.e. the number of resources that match the query. In the context of other resources, the value SHOULD be the total number of property values (i.e. RDF triples) of the resource. Unless Stable Paging is in effect, the total count MAY vary as a client retrieves subsequent pages.</td>
</tr>
</tbody>
</table>

The subject resource URI of the oslc:ResponseInfo resource representation will be the HTTP request URI, or URI from subsequent redirects. The response representation may also include properties from subject resources different from the one identified by the request URI.

Here’s an example, using the OSLC Core RDF/XML representation guidance, that illustrates how the oslc:ResponseInfo resource representation is included in addition to the blog entry resource representation.

**Example: Resource Paging, partial response with response info resource representation**

**Example URI:** http://example.com/blogs/entry/1?oslc.paging=true&pageno=2

```xml
<rdf:RDF
  xmlns:oslc_blog="http://open-services.net/ns/bogus/blogs#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:dcterms="http://purl.org/dc/terms/">

  <oslc_blog:Entry
    rdf:about="http://example.com/blogs/entry/1">
    <!-- partial property values of of the blog entry -->
  </oslc_blog:Entry>

  <oslc:ResponseInfo rdf:about="http://example.com/blogs/entry/1?oslc.paging=true&pageno=2">
    <oslc:nextPage rdf:resource="http://example.com/blogs/entry/1?oslc.paging=true&pageno=3" />
  </oslc:ResponseInfo>

</rdf:RDF>
```

Refer to the OSLC Defined Resource Representation guidance for an explanation of how the response info resource may be represented in RDF/XML.

**Selective Property Values**

OSLC Services MAY support a technique called Selective Properties to enable clients to retrieve only selected property values.
By adding the key=value pair `oslc.properties`, specified below, to a resource URI, a client can request a new resource with a subset of the original resource's values. Here's how the selective properties values `oslc.properties` and `oslc.prefix` work.

### oslc.properties

The `oslc.properties` key=value pair lets you specify the set of properties to be included in the response. Both immediate and nested properties may be specified. A nested property is a property that belongs to the resource referenced by another property. Nested properties are enclosed in brace brackets, and this nesting may be done recursively, i.e. a nested property may contain other nested properties.

For example, suppose we have a bug report resource at the following URL:

```
http://example.com/bugs/4242
```

Suppose this bug resource has properties such as `dcterms:title`, `dcterms:description`, and `dcterms:creator`, and that `dcterms:creator` refers to a person resource that has properties such as `foaf:givenName` and `foaf:familyName`. Suppose you want a representation of the bug report that includes its `dcterms:title` and the `foaf:givenName` and `foaf:familyName` of the person referred to by its `dcterms:creator`. The following URL illustrates the use of the `oslc.properties` query value to include those properties:

```
http://example.com/bugs/4242?oslc.properties=dcterms:title,dcterms:creator{foaf:givenName,foaf:familyName}
```

### Syntax

The `oslc.properties` pair is defined by the `oslc_properties` term in the following BNF grammar:

```
oslc_properties ::= "oslc.properties=" properties
properties ::= property ("," property)*
property ::= identifier | wildcard | nested_prop
nested_prop ::= (identifier | wildcard) "{" properties "}"
wildcard ::= "*"
identifier ::= PrefixedName
PrefixedName ::= /* see "SPARQL Query Lanaguage for RDF", http://www.w3.org/TR/rdf-sparql-query/#rPrefixedName */
```

### oslc.prefix

In our examples of `oslc.properties`, property names include a URI prefix, i.e. `dcterms:` or `foaf:`. An OSLC Service SHOULD predefine URI prefixes for its properties. Here we assume that OSLC has predefined the Dublin Core (`dcterms:`) and Friend of a Friend (`foaf:`) prefixes. However, OSLC resources SHOULD also be open to new content, which means that new properties may not have predefined URI prefixes. We therefore need a way to define new URI prefixes in resource requests. The `oslc.prefix` value lets you specify URI prefixes used in property names. For example, suppose the `foaf:` prefix was not predefined. The following URL illustrates the use of the `oslc.prefix` value to define it:

```
http://example.com/bugs/4242?oslc.prefix=foaf=http://xmlns.com/foaf/0.1/&oslc.properties=foaf:lastName,...
```

### Syntax

The syntax of the `oslc.prefix` is defined by the `oslc_prefix` term in the following BNF grammar:
Common Properties

OSLC domains specifications are strongly encouraged to use the common properties approved by the OSLC Core Workgroup (See OSLC Core Spec Appendix A) rather than defining new properties.

Service Provider Resources

OSLC Services are accessible via a Service Provider Resource that describes each service, which domain specifications the service implements as well as the creation, query and delegated UI capabilities of each service.

Additionally, a provider may offer a Service Provider Catalog that lists related Service Providers.

Conceptual Model

The conceptual model of Service Provider Catalog and Service Provider resources is simple. They are both resources with property values. The values allowed and required in each type of resource are defined below.

The diagram below illustrates the Service Provider Catalog and Service Provider concepts and relationships. As you can see there are two Resources defined: Service Provider Catalog and Service Provider. There are also a set of Local In-Line Resources that are used inside the Resources to define namespaces, OAuth configurations, contributors as well as services and their capabilities.

Figure #2: Service Provider concepts and relationships
Next, we will formally define the Service Provider Catalog and Service Provider resources.

**Resource: Service Provider Catalog**

An OSLC implementation that offers one or more Service Provider resources (see below), MAY also provide Service Provider Catalog to enable OSLC clients to find Service Providers offered. These catalogs may contain other nested catalogs as well as service providers.

- **Name:** ServiceProviderCatalog
- **URI:** http://open-services.net/ns/core#ServiceProviderCatalog
<table>
<thead>
<tr>
<th>Prefix</th>
<th>Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>zero-or-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td>Title of the service provider catalog</td>
</tr>
<tr>
<td>dcterms:description</td>
<td>zero-or-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td></td>
<td>Description of the service provider catalog</td>
</tr>
<tr>
<td>dcterms:publisher</td>
<td>zero-or-one</td>
<td>True</td>
<td>Local Resource</td>
<td>Inline</td>
<td>oslc:Publisher</td>
<td></td>
<td>Describes the software product that provides the implementation.</td>
</tr>
<tr>
<td>oslc:domain</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td></td>
<td>Namespace URI of the specification that is implemented by this service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>In most cases this namespace URI will be for an OSLC domain, but other</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>URIs MAY be used.</td>
</tr>
<tr>
<td>oslc:serviceProvider</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Either</td>
<td>oslc:ServiceProvider</td>
<td></td>
<td>A service offered by the service provider.</td>
</tr>
<tr>
<td>oslc:serviceProviderCatalog</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Either</td>
<td>oslc:ServiceProviderCatalog</td>
<td></td>
<td>Additional service provider catalog.</td>
</tr>
<tr>
<td>oslc:oauthConfiguration</td>
<td>zero-or-many</td>
<td>True</td>
<td>Local Resource</td>
<td>Inline</td>
<td>oslc:OAuthConfiguration</td>
<td></td>
<td>Defines the three OAuth URIs required for a client to act as an OAuth</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>consumer.</td>
</tr>
</tbody>
</table>

**Resource: Service Provider**

A Service Provider describes a set of services offered by an OSLC implementation.

- **Name**: ServiceProvider
- **URI**: http://open-services.net/ns/core#ServiceProvider
<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>zero-or-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td>Title of the service provider</td>
</tr>
<tr>
<td>dcterms:description</td>
<td>zero-or-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td>Description of the service provider</td>
</tr>
<tr>
<td>dcterms:publisher</td>
<td>zero-or-one</td>
<td>True</td>
<td>Local Resource</td>
<td>Inline</td>
<td>oslc:Publisher</td>
<td>Describes the software product that provides the implementation.</td>
</tr>
<tr>
<td>oslc:service</td>
<td>one-or-many</td>
<td>True</td>
<td>Local Resource</td>
<td>Inline</td>
<td>oslc:Service</td>
<td>Describes a service offered by the service provider.</td>
</tr>
<tr>
<td>oslc:details</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>any</td>
<td>A URL that may be used to retrieve a web page to determine additional details about the service provider.</td>
</tr>
<tr>
<td>oslc:prefixDefinition</td>
<td>zero-or-many</td>
<td>True</td>
<td>Local Resource</td>
<td>Inline</td>
<td>oslc:PrefixDefinition</td>
<td>Defines a namespace prefix for use in JSON representations and in forming OSLC Query Syntax strings.</td>
</tr>
<tr>
<td>oslc:oauthConfiguration</td>
<td>zero-or-one</td>
<td>True</td>
<td>Local Resource</td>
<td>Inline</td>
<td>oslc:OAuthConfiguration</td>
<td>Defines the three OAuth URIs required for a client to act as an OAuth consumer.</td>
</tr>
</tbody>
</table>

**Resource: Service**

A Service describes the specific services offered by an implementation of an OSLC specification.

- **Name**: Service
- **URI**: http://open-services.net/ns/core#Service
<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:domain</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>Namespace URI of the specification that is implemented by this service. In most cases this namespace URI will be for an OSLC domain, but other URIs MAY be used.</td>
</tr>
<tr>
<td>oslc:creationFactory</td>
<td>zero-or-many</td>
<td>True</td>
<td>Local</td>
<td>n/a</td>
<td>oslc:CreationFactory</td>
<td>Enables clients to create new resources</td>
</tr>
<tr>
<td>oslc:queryCapability</td>
<td>zero-or-many</td>
<td>True</td>
<td>Local</td>
<td>n/a</td>
<td>oslc:QueryCapability</td>
<td>Enables clients query across a collection of resources</td>
</tr>
<tr>
<td>oslc:selectionDialog</td>
<td>zero-or-many</td>
<td>True</td>
<td>Local</td>
<td>n/a</td>
<td>oslc:Dialog</td>
<td>Enables clients to select a resource via UI</td>
</tr>
<tr>
<td>oslc:creationDialog</td>
<td>zero-or-many</td>
<td>True</td>
<td>Local</td>
<td>n/a</td>
<td>oslc:Dialog</td>
<td>Enables clients to create a resource via UI</td>
</tr>
<tr>
<td>oslc:usage</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Resource</td>
<td>n/a</td>
<td>An identifier URI for the domain specified usage of this service. If a service provider has multiple services, it may designate the primary or default one that should be used with a property value of <a href="http://open-services.net/ns/core#default">http://open-services.net/ns/core#default</a></td>
</tr>
</tbody>
</table>

**Resource: Creation Factory**

A Creation Factory describes a creation factory, capable of creating new resources via HTTP POST.

- **Name**: CreationFactory
- **URI**: http://open-services.net/ns/core#CreationFactory
<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>exactly-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td>Title string that could be used for display</td>
</tr>
<tr>
<td>oslc:label</td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Very short label for use in menu items</td>
</tr>
<tr>
<td>oslc:creation</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>To create a new resource via the factory, post it to this URI</td>
</tr>
<tr>
<td>oslc:resourceShape</td>
<td>zero-or-one</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td>oslc:ResourceShape</td>
<td>A Creation Factory <strong>MAY</strong> provide Resource Shapes that describe shapes of resources that may be created.</td>
</tr>
<tr>
<td>oslc:resourceType</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>The expected resource type URI of the resource that will be created using this creation factory. These would be the URIs found in the result resource's <em>rdf:type</em> property.</td>
</tr>
<tr>
<td>oslc:usage</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>An identifier URI for the domain specified usage of this creation factory. If a service provides multiple creation factories, it may designate the primary or default one that should be used with a property value of <a href="http://open-services.net/ns/core#default">http://open-services.net/ns/core#default</a></td>
</tr>
</tbody>
</table>

**Resource: Query Capability**

A Query Capability describes a query capability, capable of querying resources via HTTP GET or POST.

- **Name**: QueryCapability
- **URI**: [http://open-services.net/ns/core#QueryCapability](http://open-services.net/ns/core#QueryCapability)

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>exactly-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td>Title string that could be used for display</td>
</tr>
<tr>
<td>oslc:label</td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Very short label for use in menu items</td>
</tr>
<tr>
<td>oslc:queryBase</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>The base URI to use for queries. Queries are invoked via HTTP GET on a query URI formed by appending a key=value pair to the base URI, as described in Query Capabilities section.</td>
</tr>
<tr>
<td>oslc:resourceShape</td>
<td>zero-or-one</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>The Query Capability <strong>SHOULD</strong> provide a Resource Shape that describes the query base URI.</td>
</tr>
<tr>
<td>oslc:resourceType</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>The expected resource type URI that will be returned with this query capability. These would be the URIs found in the result resource's <em>rdf:type</em> property.</td>
</tr>
<tr>
<td>oslc:usage</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource Reference</td>
<td>n/a</td>
<td></td>
<td>An identifier URI for the domain specified usage of this query capability. If a service provides multiple query capabilities, it may designate the primary or default one that should be used with a property value of <a href="http://open-services.net/ns/core#default">http://open-services.net/ns/core#default</a></td>
</tr>
</tbody>
</table>
**Resource: Dialog**

A Dialog describes a delegated user interface (UI) which can be used to allow a user to interactively create a new resource or pick a resource.

- **Name:** Dialog
- **URI:** http://open-services.net/ns/core#Dialog

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>exactly-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td>Title string that could be used for display</td>
</tr>
<tr>
<td>oslc:label</td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Very short label for use in menu items</td>
</tr>
<tr>
<td>oslc:dialog</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>The URI of the dialog</td>
</tr>
<tr>
<td>oslc:hintWidth</td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Values <strong>MUST</strong> be expressed in <strong>relative length units</strong> as defined in the W3C Cascading Style Sheets Specification (CSS 2.1) Em and ex units are interpreted relative to the default system font (at 100% size).</td>
</tr>
<tr>
<td>oslc:hintHeight</td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Values <strong>MUST</strong> be expressed in <strong>relative length units</strong> as defined in the W3C Cascading Style Sheets Specification (CSS 2.1) Em and ex units are interpreted relative to the default system font (at 100% size).</td>
</tr>
<tr>
<td>oslc:resourceType</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>The expected resource type URI for the resources that will be returned when using this dialog. These would be the URIs found in the result resource’s <code>rdf:type</code> property.</td>
</tr>
<tr>
<td>oslc:usage</td>
<td>zero-or-many</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>An identifier URI for the domain specified usage of this dialog. If a service provides multiple selection or creation dialogs, it may designate the primary or default one that should be used with a property value of <a href="http://open-services/ns/core#default">http://open-services/ns/core#default</a></td>
</tr>
</tbody>
</table>

**Resource: Publisher**

A Publisher identifies and describes the software product that provides the OSLC implementation.

- **Name:** Publisher
- **URI:** http://open-services.net/ns/core#Publisher
<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcterms:title</td>
<td>exactly-one</td>
<td>True</td>
<td>XMLLiteral</td>
<td>n/a</td>
<td>n/a</td>
<td>Title string that could be used for display</td>
</tr>
<tr>
<td>oslc:label</td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Very short label for use in menu items</td>
</tr>
<tr>
<td>dcterms:identifier</td>
<td>exactly-one</td>
<td>unspecified</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>A URN that uniquely identifies the implementation</td>
</tr>
<tr>
<td>oslc:icon</td>
<td>zero-or-one</td>
<td>True</td>
<td>Resource</td>
<td>reference</td>
<td>n/a</td>
<td>URL to an icon file that represents the provider. This icon should be a favicon format and 16x16 pixels in size</td>
</tr>
</tbody>
</table>

**Resource: Prefix Definition**

Service Providers **MUST** provide a Prefix Definition for each prefix supported by the service. Each Prefix Definition defines a namespace prefix that clients **MAY** use in forming OSLC Query Syntax strings.

- **Name**: PrefixDefinition
- **URI**: http://open-services.net/ns/core#PrefixDefinition

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:prefix</td>
<td>exactly-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Namespace prefix to be used for this namespace</td>
</tr>
<tr>
<td>oslc:prefixBase</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>The base URI of the namespace</td>
</tr>
</tbody>
</table>

**Resource: OAuth Configuration**

Service Providers that support OAuth Authentication **SHOULD** provide a way for clients to automatically discover the three OAuth URIs necessary to act as an OAuth Consumer.

- **Name**: OAuthConfiguration
- **URI**: http://open-services.net/ns/core#OAuthConfiguration

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:oauthRequestTokenURI</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>URI for obtaining OAuth request token</td>
</tr>
<tr>
<td>oslc:authorizationURI</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>URI for obtaining OAuth authorization</td>
</tr>
<tr>
<td>oslc:oauthAccessTokenURI</td>
<td>exactly-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>URI for obtaining OAuth access token</td>
</tr>
</tbody>
</table>

The next sections cover the Creation Factory and Query Capability in more detail.

**Creation Factories**

An OSLC Service can provide one or more creation factory to enable the creation of new resources. A creation factory provides a Creation URI used to create new resources via HTTP POST and may also provide Resource Shapes that describe the types of
Creating an OSLC Defined Resource

To create an OSLC Defined Resource, an OSLC Client first forms an representation of that resource including the desired and required property values. A client can learn what properties are allowed in a new OSLC Defined Resource via the OSLC specification that defines or, in some cases via a Resource Shape resource. Next the client uses HTTP POST to post that representation to a Creation URI.

- The response to a successful HTTP POST of a representation to a Creation Resource URI MAY include a representation of the newly created resource.
- The resource returned MAY contain changes to properties made by the server or new properties added by the server.

Query Capabilities

An OSLC Service may provide one or more Query Capabilities to enable query of resources. A Query Capability provides a base URI for forming Query Resource URIs and MAY provide Resource Shapes that describe the property values that may be expected in the resources that are queryable via the query capability. Thus, Query Capabilities provide a way to discover the resources managed by an OSLC Service.

In a Query Capability, the base URI, as defined by the oslc:queryBase property, is itself a resource managed by the service and it acts as the starting subject resource for the queries based on it. Since the list may contain hundreds of thousands of members, queries are used to filter the list for members that satisfy certain conditions, e.g. the bugs that have high priority and were created this week.

Conceptual Model

To perform a query an OSLC client first creates a URI by starting with a Query Capability's base URI as a base and adding a URI Query String to express the query criteria. The OSLC client then uses HTTP GET to request a Query Resource representation of the query results. The Query Resource representation will contain property values about the query and a collection of resources that match the query criteria.

HTTP GET Queries

To perform an HTTP GET query, an OSLC client starts with the base URI as defined by the oslc:queryBase property of a Query Capability, and appends to it query parameters in a syntax supported by the service. The resulting URI is the query URI. The OSLC client sends an HTTP GET request to the query URI, optionally specifying the preferred content media type for the query response in the HTTP Accept header. OSLC Services MUST support query responses in RDF/XML format (media type application/rdf+xml) and MAY support other formats. OSLC Services SHOULD support the Query Syntax defined in this specification, but MAY support other syntaxes.

Query Syntax

A query URI can be formed by adding a query string to the end of the Query Capability's base URI (or by sending the query string in the request body when using HTTP POST). The syntax used to express the query criteria in that string is specified by each OSLC domain specification.

The OSLC Core Spec Query Specification document defines a standard set of OSLC query parameters that other OSLC domain specifications MAY use to query resources.
Query Specification errors

If there is an error in the specification of the query, whether the query is specified by key=value pairs in the HTTP GET URL or key=value pairs in the body of an HTTP POST, then the provider **MUST** respond with an error. The error response should be an HTTP 400 Bad Request error and an explanation of the error in the OSLC Error Response format (see Error Responses below).

Delegated User Interface Dialogs

OSLC specifications target specific integration scenarios. In some cases, allowing one product to delegate to a user interface defined in another product is a more effective way to support a use case than an HTTP interface that can only be accessed programmatically. There are two cases where this is especially true:

- **Resource creation**: when a user of a web application needs to create a new resource in an OSLC Service Provider. In this case the web application asks the service provider to provide a UI for resource creation and the provider notifies the application when the creation has been completed or canceled by the user.

- **Resource selection**: when a user of a web application needs to pick a resource managed by a OSLC Service Provider. In this case the web application asks the service provider to provide a UI for resource selection and the provider notifies the application when a resource or resources has been selected or if the selection was canceled.

To support these two cases, below we define OSLC Delegated User Interface (UI) Dialogs. Delegated UI Dialogs are a technique where one provider can embed a creation or selection UI into another using a combination of an HTML `<iframe>` and JavaScript code. The diagram below illustrates how delegated UI dialogs work in a scenario where Provider A wants to allow a user to select or create a resource managed by Provider B.

**Figure #3: Delegated UI Dialog interactions**

Next, the details of the Delegated UI Dialog protocol.

**Terminology**

The following terms are used in discussions of Delegated UI Dialogs:

- **UI Consumer** - a web application that is embedding a Delegated UI Dialog from an OSLC Service Provider. This consumer could be a web page, with the Delegated UI Dialog loaded into an iframe or a native application, e.g. an IDE like Eclipse, that
is embedding a web browser component.

- **UI Provider** - an OSLC Service provider that offers one or more Delegated UI Dialogs. These dialogs will be specified in the provider's Service Provider resource.

The next sections explain how Delegated UI Dialogs work.

### Post Message and Window Name protocols

To support the widest range of web browsers, we define two different protocols for communicating the information about the user's action from the UI Provider and back to the UI Consumer. These are the Post Message and Window Name protocols described below.

In both the Post Message and Window Name protocols, the way that a UI Consumer includes a Delegated UI Dialog in an HTML page is to create an iframe and specify the src as the URI of the Delegated UI Dialog to be included. The UI Consumer indicates the protocol to be used by appending one of the two fragment identifiers below to the URI:

- #oslc-core-postMessage-1.0 - Indicates that the Post Message protocol is to be used
- #oslc-core-windowName-1.0 - Indicates that the Window Name protocol is to be used

The JavaScript code example below shows how a UI Provider can determine which protocol is in use:

```javascript
if (window.location.hash == '#oslc-core-windowName-1.0') {
    // Window Name protocol in use
} else if (window.location.hash == '#oslc-core-postMessage-1.0') {
    // Post Message protocol in use
}
```

### iframe Creation Considerations

Regardless of the protocol in effect, it is recommended that UI Consumers follow the below iframe creation guidelines to provide a more seamless user experience:

- Embed the iframe within a div element, with height and width set based on the relative length values specified in the Service Resource that declares the Delegated UI Dialog.
- Set the iframe border size to '0'
- Set the iframe scrolling to 'auto'

Next, the details for each of the two protocols.

### Post Message Protocol

The Post Message protocol relies on the HTML5 function `window.postMessage()` (reference: HTML5), available in the latest or pending releases of most browsers. UI Consumers must anticipate other, unrelated uses of postMessage(), and should ignore messages not conforming to this protocol.

Typically, the embedded page will be loaded in a window inside another window, such as a iframe inside some surrounding webpage. In such cases, `postMessage()` must be called on that parent window. But in a native application, an outer page is not needed and the embedded page may be shown directly in the browser's "root" window. When the embedded page has no parent window, it must call `postMessage()` on its own window.

Here are the responsibilities of the UI Consumer and UI Provider in Post Message protocol.

### The UI Consumer's responsibilities
1. Include the Delegated UI Dialog via iframe (i.e setting iframe src to the URI of the Delegated UI Dialog) or via an embedded browser. Append the fragment identifier #oslc-core-postMessage-1.0 to the URL to indicate that Post Message is the desired protocol.
2. Add a 'message' listener to the outer window to receive messages from the Delegated UI Dialog.
3. Listen for window 'message' events, ignoring events from other sources or not prefixed with "oslc-response:"
4. When message from Delegated UI Dialog indicates user completed action, free resources and handle action.

The UI Provider's responsibilities

1. Provide Delegated UI Dialog, an HTML page that provides a user interface for resource creation or selection.
2. Allow the user to perform resource creation or selection.
3. Once the user has created, selected or canceled, send notification using `postMessage()` to the page's parent window, passed in `event.data` string, that is prefixed with "oslc-response:" See below for the two possible response formats, one for resource selection and one for creation.
4. If the page is not parented, then the message is posted to the page's own window. The page must ignore this message to itself.

The below JavaScript code example shows how a UI Provider page would send a response using `postMessage()` and taking into account the fact that some pages are not parented.

```
function respondWithPostMessage(/*string*/ response) {
    (window.parent | window).postMessage("oslc-response:" + response, "*");
}
```

Now, the Window Name protocol.

**Window Name Protocol**

The Window Name protocol uses the HTML DOM `window.name` property to communicate the response (reference: Window Object) from the UI Provider to the UI Consumer. This special property of window maintains its value even as the window navigates to a different origin, but the iframe's `window.name` can only be read when the accessing window shares the same origin. For this to happen, when the embedded page is finished it must set the `window.name` and also change the `window.location` to a page with the same origin as the outer frame. This not only allows the UI Consumer to access the result, but also fires an event telling the UI Consumer when to do so. This return location is passed to the embedded page using the `window.name` property.

Here are the responsibilities of the UI Consumer and UI Provider in Window Name protocol.

The UI Consumer's responsibilities

1. Include the Delegated UI Dialog via iframe (i.e setting iframe src to the URI of the Delegated UI Dialog) or via an embedded browser. Append the fragment identifier #oslc-core-windowName-1.0 to the URL to indicate that Window Name is the desired protocol.
2. On the iframe, set the frame's `window.name` to indicate the Return URL.
3. On the iframe, listen for 'onload' events
4. When an 'onload' event occurs an the frame's location is equal to the Return URL then read the response from the `window.name`.

The following Javascript code illustrates the protocol. The code for the `destroyFrame()`, `handleMessage()` and `displayFrame()` methods are not provided in this example, but should be obvious to a JavaScript developer. The UI Consumer must provide these methods.
The UI Provider's responsibilities

As soon as the embedded page has loaded, perform the following:

1. Provide Delegated UI Dialog, an HTML page that provides a user interface for resource creation or selection.
2. Read the Return URL from the window.name variable
3. Allow user to perform resource creation or selection.
4. Once user has created, selected or canceled, communicate the user's response by setting the window.name variable to the response. See below for the two possible response formats, one for resource selection and one for creation.
5. Indicate that user has responded by setting the window.location to the Return URL specified by the UI Consumer.

The JavaScript example below shows a UI Provider notifying its UI Consumer after a user has responded.

```javascript
function respondWithWindowName(/*string*/ response) {
    // Step #2: read the return URL
    var returnURL = window.name;

    // Step #4: send the response via the window.name variable
    window.name = response;

    // Step #5: indicate that user has responded
    window.location = returnURL;
}
```

**Resource Selection**

Resource Selection can be used when a UI Consumer wants to allow a user to pick a resource that is managed by an OSLC Service. Using either the Post Message or Window Name protocols defined above, the UI Consumer uses an iframe to embed a selection dialog that is provided by the service, then awaits notification that the user has selected a resource.

To enable Resource Selection, an OSLC Service **MUST** provide in its Service Resource a value for the `oslc:selectionDialog` property. The property value will include a `oslc:dialogURI` property that indicates the URI of the selection dialog.

Regardless of how the response from the UI Provider is conveyed to the UI Consumer, the response **SHOULD** be formatted as follows:

- **Name:** results
- **URI:** http://open-services.net/ns/core#results

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rdf:resource</code></td>
<td>zero-or-one</td>
<td>True</td>
<td>Resource</td>
<td>Reference</td>
<td>n/a</td>
<td>URI of the resource selected or created</td>
</tr>
<tr>
<td><code>oslc:label</code></td>
<td>zero-or-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Short label describing the resource selected</td>
</tr>
</tbody>
</table>

An empty array indicates that the resource selector has been canceled

An example Resource Selection response:

```json
{
    "oslc:results": [{
        "oslc:label": "Bug 123: Server crash",
        "rdf:resource": "http://example.com/bug123"
    }, {
        "oslc:label": "Bug 456: Client hangs on startup",
        "rdf:resource": "http://example.com/bug456"
    }
}
```
Resource Creation

Resource Creation can be used when a UI Consumer wants to allow a user to create a new resource that is managed by an OSLC Service. Using either the Post Message or Window Name protocols defined above, the UI Consumer uses an iframe to embed a creation dialog that is provided by the service, then awaits notification that the user has created a resource.

To enable Resource Creation, an OSLC Service **MUST** provide in its Service Resource a value for the `oslc:creationDialog` property. The property value will include a `oslc:dialogURI` property that indicates the URI of the creation dialog.

Regardless of how the response from the UI Provider is conveyed to the UI Consumer, the response **SHOULD** be formatted as defined by `oslc:results`

Example:

```
{
    "oslc:results" : [ {
        "oslc:label": "Bug 123: Server crash",
        "rdf:resource": "http://example.com/bug123"
    }, {
        "oslc:label": "Bug 456: Client hangs on startup",
        "rdf:resource": "http://example.com/bug456"
    }
}
```

Prefilling Creation Dialogs

Service providers **MAY** support receiving a POST request whose content body is a resource representation to the Creation Dialog URI to retrieve a URI that represents the embedded page to be used. Service providers **MUST** respond with a response status of 201 (Created) with the response header `Location` whose value is the URI to request the newly created form. After some elapsed time, service providers **MAY** respond with a 404 (Not Found), 410 (Gone) or 3xx (Redirect) to an HTTP GET request for these URIs.

Dialog Resizing

Delegated UI dialogs receive their initial size (dimensions) based on the `oslc:hintWidth` and `oslc:hintHeight` properties described in `oslc:Dialog` resource description. There are cases where UI Provider recognizes that the initial size of a Delegated UI dialog is not sufficient and needs a way to ask the UI Consumer to resize the dialog. In this section we specify a mechanism that enables dialog resizing, but only when Post Message Protocol is used.

Consumers **MAY** honor a dialog's ability to dynamically resize. Those that do (a) **MUST** use Post Message Protocol, (b) **MUST** use the `oslc:resize` value instead of any static width or height, and and (c) **MUST** register a handler to receive dialog resize messages sent by the dialog Provider and adjust the size of the dialog accordingly.

Since a dialog is allowed to resize itself any number of times, the Consumer **MUST** keep a handler registered and react appropriately each time it received a dialog resize message from that dialog.

UI Providers **SHOULD NOT** request sizes larger than 95% of the current `viewport`, to avoid covering the entire viewport with the dialog.

Here are the responsibilities of the UI Consumer and UI Provider in dialog resizing.

**The UI Consumer's responsibilities**
1. Include the Delegated UI Dialog via iframe (i.e setting iframe src to the URI of the Delegated UI Dialog) or via an embedded browser.
2. Add a 'message' listener to the outer window to receive messages from the Delegated UI Dialog.
3. Listen for window 'message' events, ignoring events from other sources or not prefixed with "oslc-resize:". Multiple resize 'message' events may be sent while the dialog is visible.
4. When message from Delegated UI Dialog indicates user completed action, free resources and handle action.

**The UI Provider's responsibilities**

1. Provide Delegated UI Dialog, an HTML page that provides a user interface for resource creation or selection.
2. Allow the user to perform resource creation or selection.
3. Once the Provider needs to resize the dialog, send notification using `postMessage()` to the page's parent window, passed in `event.data` string, that is prefixed with "oslc-resize:". Multiple resize messages may be sent. See below for the response format.
4. If the page is not parented, then the message is posted to the page's own window. The page must ignore this message to itself.

The below JavaScript code example shows how a UI Provider page would send a response using `postMessage()` and taking into account the fact that some pages are not parented.

```javascript
function respondWithPostMessage(/*string*/ resize_response) {
    (window.parent | window).postMessage("oslc-resize:" + resize_response, "+" );
}
```

Regardless of how the response from the UI Provider is conveyed to the UI Consumer, the response SHOULD be formatted as follows:

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:hintHeight</td>
<td>exactly-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>New dialog height size. Height size MUST be expressed in <strong>relative length units</strong> as defined in the <a href="https://www.w3.org/TR/css21/">W3C Cascading Style Sheets Specification (CSS 2.1)</a> Em and ex units are interpreted relative to the default system font (at 100% size)</td>
</tr>
<tr>
<td>oslc:hintWidth</td>
<td>exactly-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>New dialog width size. Width size MUST be expressed in <strong>relative length units</strong> as defined in the <a href="https://www.w3.org/TR/css21/">W3C Cascading Style Sheets Specification (CSS 2.1)</a> Em and ex units are interpreted relative to the default system font (at 100% size)</td>
</tr>
</tbody>
</table>

An example Dialog Resize response with new height of '600px' and a width of '400px':

```json
{
    "oslc:hintHeight" : "600px",
    "oslc:hintWidth" : "400px"
}
```

That brings us to the end of the Delegated UI section. Next up, another UI related topic.

**User Interface Previews**

OSLC providers MAY support a technique known as User Interface (UI) Preview, that can be used to show a user in-context.
Authentication

OSLC Services use standard web protocols for authentication. OSLC Services can use HTTP Basic Authentication, OAuth or both.

HTTP Basic Authentication

OSLC Services MAY protect resources with HTTP Basic Authentication. OSLC Services that use HTTP Basic Authentication SHOULD do so only via SSL.

OAuth Authentication

OSLC Services MAY protect resources with OAuth Authentication.

Form Based Authentication

OSLC Services MAY use other authentication mechanisms, including those common described as Form Based Authentication. OSLC Services that choose to use other authentication mechanisms are responsible for specifying how those mechanisms work.

Error Responses

OSLC Services use the standard mechanisms of HTTP to report status and error codes to clients. When an error occurs and useful information can be provided to clients OSLC Services SHOULD return error information in the body of the response.

OSLC Services SHOULD use the Error resource defined below as the basis for forming error responses. OSLC Services SHOULD return an Error resource using the same representation requested by the client via the HTTP Accept header.

Conceptual Model

The following OSLC Defined Resource can be used as the basis for forming an error response.

Resource: Error

- Name: Error
- URI: http://open-services.net/ns/core#Error

<table>
<thead>
<tr>
<th>Prefixed Name</th>
<th>Occurs</th>
<th>Read-only</th>
<th>Value-type</th>
<th>Representation</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oslc:statusCode</td>
<td>exactly-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>The HTTP status code reported with the error.</td>
</tr>
<tr>
<td>oslc:message</td>
<td>exactly-one</td>
<td>True</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>An informative message describing the error that occurred.</td>
</tr>
<tr>
<td>oslc:extendedError</td>
<td>zero-or-one</td>
<td>True</td>
<td>Either</td>
<td>oslc:ExtendedError</td>
<td>n/a</td>
<td>Extended error information</td>
</tr>
</tbody>
</table>

Resource: Extended Error
Specification Versioning

One of the goals of the OSLC initiative is to mitigate or eliminate the need for lock-step version upgrades, where clients or services target one version of a specification and break when new versions are introduced -- requiring all services to be upgraded simultaneously.

In this section we specify a version header that will enable old "Version 1" OSLC clients to continue to work and share the same resource URIs as used by clients that specifically target the Core. And we establish rules that will enable clients to continue to work as new versions of specifications are introduced.

Supporting pre-Core clients

We anticipate that the OSLC Core and domain specifications will each be versioned independently and each version will be assigned a version number, but we would like to avoid exposing version numbers in OSLC implementations. There is one use case that requires version information to be exposed. We must ensure that old OSLC "Version 1" clients continue to work.

To enable OSLC Service specifications to evolve without breaking existing clients, we introduce an HTTP Header, OSLC-Core-Version set to the Core specification version number "2.0". We expect clients that target the Core to send this HTTP header.

- If the OSLC-Core-Version header is present and set to a version that the service can support, then the service MUST return a representation that is complies with the specified version.
- If the OSLC-Core-Version header is present and indicates a specification version that the service cannot support, the service SHOULD respond with what it determines is the most compatible version that it can return.
- If the OSLC-Core-Version header is not present then the OSLC Service SHOULD respond by returning a resource that conforms to the earliest or most compatible (as determined by the implementation) specification version's representation. Services that never offered an OSLC Version 1 interface can ignore this restriction.
- When returning an OSLC Defined Resource, OSLC Services MUST return the OSLC-Core-Version header set to the Core specification with which the representation complies.

Rules for new versions of OSLC specifications

When specifying a new version of an OSLC specification the rule is this:

A new version of an OSLC specification is not allowed to introduce changes that will break old clients.
Here are some guidelines for OSLC workgroups defining new specifications or upgrades to existing ones:

- If you believe that you need a property but cannot agree on the value-type, then this is a strong indication that you should not attempt to standardize on the property. Once you decide on a value-type you are stuck with it forever. Wait until you have the scenarios or implementation experience needed to agree on type.
- When introducing a new capability in a new version of a specification, e.g. a creation factory, query capability or delegated UI dialog; one that works differently than those specified in the Core spec or older versions of your own specification, you should create a new resource type to represent the service. This will enable old clients to continue to work against old services and new clients to work with your new capabilities.
- When defining resources, do not remove, change the meaning or the value-type of any properties that you defined in earlier versions of the specification. You can add new properties but not change those that already exist.
- It is possible to relax restrictions on clients, because relaxing restrictions should not break clients. But it is not possible to relax restrictions on services, because clients expect to find the required fields and if they are missing, clients will break.
- Before defining a new property within your OSLC domain's namespace consult the list of common properties in OSLC Core Spec Appendix A to see if using a common property would be more appropriate.

Migrating to the Core Specification

Most of the first OSLC specifications were developed before this Core specification existed and do not implement versioning as described above and so must use some other mechanism to migrate to the OSLC Core v1.0 specification.

OSLC implementations that wish to continue to support old pre-Core OSLC or OSLC v1.0 specifications can do so by keeping the old implementation in place and adding the new OSLC Core v1 implementation with different service provider, query capability and creation factory URIs.

OSLC Defined Resource Representations

This section specifies what resource representations are required for OSLC resources, some requirements for providing representations and some rational for the requirement that OSLC Services provide RDF/XML representations.

OSLC resource representations come in many forms and are subject to standard HTTP mechanisms for content negotiation.

OSLC domain specifications (1) SHOULD require the representations needed for the specific scenarios that they are addressing and (2) SHOULD recognize that different representations are appropriate for different purposes. For example, browser oriented scenarios might be best addressed by JSON or Atom format representations. For these reasons, OSLC Services MAY provide and accept standard or emerging standard formats such as XML, JSON, HTML, Turtle and the Atom Syndication Format.

OSLC domain specifications are also expected to follow common practices and conventions that are in concert with existing industry standards and offer consistency across domains. All of the OSLC specifications are built upon the standard RDF data model, allowing OSLC to align with the W3C's Linked Data initiative. In addition, all OSLC specifications have adopted the convention to illustrate RDF/XML representations and will typically require RDF/XML representations to enable consistency across OSLC implementations. For those reasons, OSLC Services SHOULD provide and accept RDF/XML representations for each OSLC resource.

Though the OSLC Core workgroup does provide guidance on how to form RDF/XML representations using a subset of RDF/XML (reference: Appendix B - Representation Guidance and Examples), OSLC clients SHOULD NOT assume any specific form of RDF/XML. It is RECOMMENDED that OSLC Services also provide an HTML representation for each resource.

Use standard content-types

Note that existing standard content-types are used, e.g. application/rdf+xml and application/json, in this document and no new content-types are introduced (except for the one introduced in the UI Preview specification). Those writing OSLC specifications are strongly encouraged to follow this pattern -- use standard and existing content-types and avoid inventing new content-types for existing formats.
In past OSLC specifications we defined a specific RDF/XML format for each resource defined and gave each its own content-type. This implied to consumers that each resource had a different format when in reality they were all standard RDF/XML. Using different content-types makes it more difficult to write generic tools, crawlers and other services that work across all data.

**Order of property values insignificant**

This specification defines how OSLC property values are to be represented in a variety of formats. Except in the case of a sorted Query Response, the ordering of property values is insignificant. OSLC clients and service providers **MUST** not place any significance on the ordering of property values in representations.

**Use Absolute URIs**

OSLC representations **MUST** use absolute URIs in all cases except XML representations, where the `xml:base` attribute may be used to allow relative URIs to be resolved to absolute form (reference: XML Base).

Before a resource representation that uses `xml:base` is posted to an OSLC Service for creation, it may include relative URIs that cannot be resolved until the OSLC Service has received, created and assigned a URI to the new resource.

**Appendix A: Common Properties and Resources**

See separate page **OSLC Core Spec Appendix A**

**Appendix B: Representation Guidance and Examples**

See separate page **OSLC Core Spec Appendix B**

**Appendix C: Guidance on Links and Relationships**

See separate page **OSLC Core Spec Appendix C**

**Appendix D: References**

These are the specifications referenced by the OSLC Core Specification.

- **BNF** Backus-Naur Form
- **CSS 2.1** - Cascading Style Sheets Level 2 Revision 2 v2.1
- **Dublin Core 1.1** - Dublin Core Metadata Element Set, Version 1.1
- **FOAF** - Friend of a Friend (FOAF) v0.98
- **HTML5 Posting Message** - W3C HTML5 postMessage
- **HTTP 1.1** - Hyper-text Transfer Protocol (HTTP/1.1)
- **OAuth 1.0a** - RFC5849 - The OAuth 1.0 Protocol
- **RDF/XML Syntax** - RDF / XML Syntax Specification (Revised)
- **Turtle** - http://www.w3.org/TeamSubmission/turtle/
- **URI Syntax** - URI Generic Syntax
- **Window Object** - Window Object 1.0 Window.name
- **XML Namespaces** - Namespaces in XML 1.0.(Third Edition)
- **XML Base** - XML Base (Second Edition)