OSLC Core Version 3.0. Part 4: Delegated Dialogs

Committee Specification Draft 01

07 June 2016

Specification URIs

This version:

http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part4-delegated-dialogs/oslc-core-v3.0-csd01-part4-delegated-dialogs.html
http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part4-delegated-dialogs/oslc-core-v3.0-csd01-part4-delegated-dialogs.pdf

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/oslc-core-v3.0-part4-delegated-dialogs.html
http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/oslc-core-v3.0-part4-delegated-dialogs.pdf

Technical Committee:

OASIS OSLC Lifecycle Integration Core (OSLC Core) TC

Chairs:

Jim Amsden (jamsden@us.ibm.com), IBM

Martin Sarabura (msarabura@ptc.com), PTC

Editor:

Jim Amsden (jamsden@us.ibm.com), IBM

Additional artifacts:

This specification is one component of a Work Product that also includes:

- OSLC Core Version 3.0. Part 1: Overview. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part1-overview/oslc-core-v3.0-csd01-part1-overview.html
- OSLC Core Version 3.0. Part 2: Discovery. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part2-discovery/oslc-core-v3.0-csd01-part2-discovery.html
- OSLC Core Version 3.0. Part 3: Resource Preview. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part3-resource-preview/oslc-core-v3.0-csd01-part3-resource-preview.html
- OSLC Core Version 3.0. Part 4: Delegated Dialogs (this document). http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part4-delegated-dialogs/oslc-core-v3.0-csd01-part4-delegated-dialogs.html
- OSLC Core Version 3.0. Part 5: Attachments. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part5-attachments/oslc-core-v3.0-csd01-part5-attachments.html
- OSLC Core Version 3.0. Part 6: Resource Shape. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part6-resource-shape/oslc-core-v3.0-csd01-part6-resource-shape.html
- OSLC Core Version 3.0. Part 7: Vocabulary. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part7-core-vocabulary/oslc-core-v3.0-csd01-part7-core-vocabulary.html

Related work:

This specification is related to:

- OSLC Core Version 3.0: Link Guidance. Work in progress. Current draft: https://tools.oasis-open.org/version-control/svn/oslc-core/trunk/supporting-docs/link-guidance.html

Abstract:

Delegated dialogs allow one application to embed a creation or selection UI into another using HTML `iframe` elements and JavaScript code. The embedded dialog notifies the parent page of resize or submit events using HTML5 `postMessage`.

Status:

This document was last revised or approved by the OASIS OSLC Lifecycle Integration Core (OSLC Core) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/oslc-core#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list oslc-core-comments@lists.oasis-open.org, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/oslc-core/.

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

Citation format:

When referencing this specification the following citation format should be used:

[OSLC-Dialogs-3.0]
OSLC Core Version 3.0. Part 4: Delegated Dialogs. Edited by Jim Amsden. 07 June 2016. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part4-delegated-dialogs/oslc-core-v3.0-csd01-part4-delegated-dialogs.html. Latest version: http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/oslc-core-v3.0-part4-delegated-dialogs.html.

Notices

Table of Contents

1. Introduction

This section is non-normative.

OSLC specifications target specific integration scenarios. In some cases, allowing one application to delegate to a user interface defined in another application 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 another application. In this case, the web application asks the other server to provide a UI for resource creation, and the server 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 another application. In this case, the web application asks the other server to provide a UI for resource selection, and the server notifies the application when a resource or resources has been selected or if the selection was canceled.

Delegated dialogs support these two cases. They allow one application to embed a creation or selection UI into another using HTML `iframe` elements and JavaScript code.

Test Run #53 - Failed

Automated test failure. View log

Open Bug

Open Bug

* **Summary:**	Test Run #53 - Failed
Component:	Component 1
Version:	1.0
OS:	All
Platform:	All
Description:	

Submit Bug   Cancel
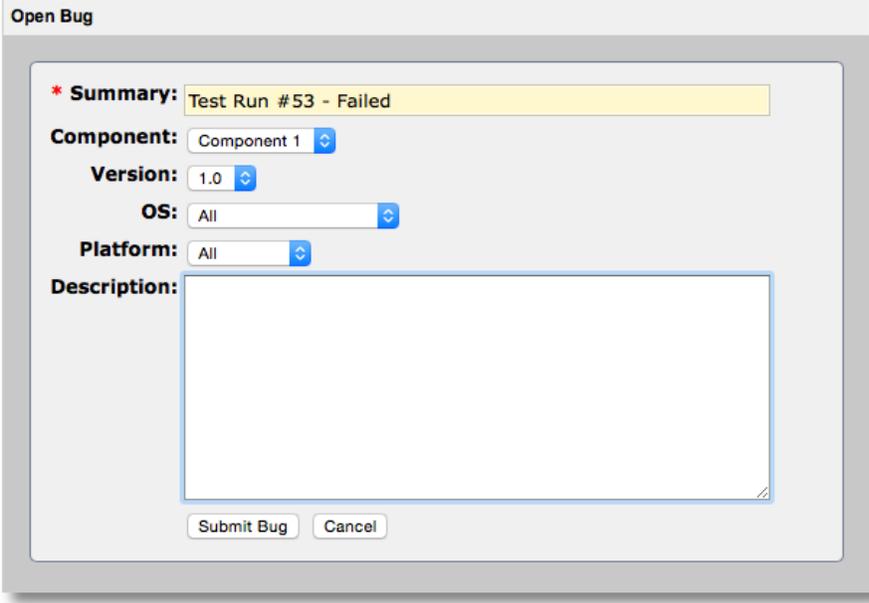
Fig. 1 Creation Dialog

Fig. 1 Creation Dialog depicts what a defect creation dialog might look like inside a quality management tool, displayed when the user clicks the *Open Bug* button. The dialog content itself comes from the change management tool, running on another server. The dialog displays seamlessly inside the quality management application, however, while retaining the change management tool's look and feel and capabilities.

Delegated dialogs provide a simple way for an end user to create or select resources from another application, thus eliminating the need to rebuild the other application's dialog and its application logic.

1.1 Terminology

Terminology uses and extends the terminology and capabilities of OSLC Core Overview [OSLCCore3], W3C Linked Data Platform [LDP], W3C's Architecture of the World Wide Web [WEBARCH], and Hyper-text Transfer Protocol [HTTP11].

The following terms are used in discussions of dialogs:

Dialog

A web page for creating or selecting resources to be embedded inside another application.

Dialog Descriptor

A resource that describes information about a dialog such as its title and dimensions. In RDF representations, the dialog descriptor has RDF type `oslc:Dialog`. See 5. Resource Constraints.

Prefill

A method of setting initial field values in a creation dialog.

1.2 References

1.2.1 Normative references

[CSS21]

Bert Bos; Tantek Çelik; Ian Hickson; Håkon Wium Lie et al. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. 7 June 2011. W3C Recommendation. URL: http://www.w3.org/TR/CSS2

[HTTP11]

R. Fielding, Ed.; J. Reschke, Ed.. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7230

[LDP]

Steve Speicher; John Arwe; Ashok Malhotra. *Linked Data Platform 1.0*. 26 February 2015. W3C Recommendation. URL: http://www.w3.org/TR/ldp/

[OSLCCore2]

S. Speicher; D. Johnson. *OSLC Core 2.0*. Finalized. URL: http://open-services.net/bin/view/Main/OslcCoreSpecification

[OSLCCore3]

Jim Amsden; S. Padgett; S. Speicher. *OSLC Core 3.0*. Working Draft. URL: http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part1-overview/oslc-core-v3.0-csd01-part1-overview.html

[OSLCCoreVocab]

Jim Amsden; S. Padgett; S. Speicher. *OSLC Core Vocabulary*. Working Draft. URL: http://docs.oasis-open.org/oslc-core/oslc-core/v3.0/csd01/part7-core-vocabulary/oslc-core-v3.0-csd01-part7-core-vocabulary.html

[OSLCPreview30]

Jim Amsden; S. Padgett; S. Speicher. *OSLC Resource Preview 3.0*. Working Draft. URL: https://tools.oasis-open.org/version-control/svn/oslc-core/trunk/specs/resource-preview.html

[RFC2119]

S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC3986]

T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. January 2005. Internet Standard. URL: https://tools.ietf.org/html/rfc3986

[RFC4627]

D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)*. July 2006. Informational. URL: https://tools.ietf.org/html/rfc4627

[RFC5988]

M. Nottingham. *Web Linking*. October 2010. Proposed Standard. URL: https://tools.ietf.org/html/rfc5988

[RFC7240]

J. Snell. *Prefer Header for HTTP*. June 2014. Proposed Standard. URL: https://tools.ietf.org/html/rfc7240

[webmessaging]

Ian Hickson. *HTML5 Web Messaging*. 19 May 2015. W3C Recommendation. URL: http://www.w3.org/TR/webmessaging/

1.2.2 Informative references

[WEBARCH]

Ian Jacobs; Norman Walsh. *Architecture of the World Wide Web, Volume One*. 15 December 2004. W3C Recommendation. URL: http://www.w3.org/TR/webarch/

1.3 Typographical Conventions and Use of RFC Terms

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this specification are to be interpreted as described in [RFC2119].

2. Discovering Dialogs

This section is non-normative.

Clients can discover dialogs in three ways:

- 2.1 Discovering Dialogs Using the Link Header
- 2.2 Discovering Dialogs Using the Prefer Header
- 2.3 Discovering Dialogs Using the Service resource

2.1 Discovering Dialogs Using the Link Header

This section is non-normative.

LDP containers [LDP] advertise their support for dialogs using the HTTP `Link` header [RFC5988]. Each dialog type, creation or selection, has its own link relation.

Dialog Type	Link Relation
Creation	`http://open-services.net/ns/core#creationDialog`
Selection	`http://open-services.net/ns/core#selectionDialog`

The client discovers the dialogs by making an HTTP OPTIONS request on the container.

EXAMPLE 1: An OPTIONS Request on the Container

```
OPTIONS /bugs/ HTTP/1.1
Host: example.com
```

The server response contains a `Link` header with URLs to the dialog descriptors.

EXAMPLE 2: Response with Dialog Link Headers

```
HTTP/1.1 204 No Content
Date: Thu, 12 Jun 2014 18:26:59 GMT
Allow: GET,POST,OPTIONS,HEAD
Accept-Post: text/turtle,application/ld+json
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>; rel="type",
      <http://example.com/dialogs/createBug>; rel="http://open-services.net/ns/core#creationDialog",
      <http://example.com/dialogs/selectBug>; rel="http://open-services.net/ns/core#selectionDialog"
```

The client then requests the dialog descriptor which contains an `oslc:dialog` property whose value is a link to the HTML dialog to be displayed.

EXAMPLE 3

```
GET /dialogs/selectBug HTTP/1.1
Host: example.com
```

EXAMPLE 4

```
HTTP/1.1 200 OK
Content-Type: text/turtle
Transfer-Encoding: chunked

@prefix oslc:   <http://open-services.net/ns/core#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<>      a                   oslc:Dialog ;
        oslc:dialog         <http://example.com/dialogs/selectBug/form> ;
        oslc:hintHeight     "600px" ;
        oslc:hintWidth      "400px" ;
        oslc:label          "Select Bug" ;
        oslc:resourceType   <http://open-services.net/ns/cm#Bug> ;
        dcterms:title       "Select Bug from Product Z" .
```

2.2 Discovering Dialogs Using the Prefer Header

This section is non-normative.

Clients can also use the HTTP `Prefer` header to find the dialogs for a container. The client makes an HTTP GET request on the resource URI using the `return=representation` preference [RFC7240] and `include` parameter [LDP] value `http://open-services.net/ns/core#PreferDialog`. The server responds with the dialog descriptors in the response body. Clients should also use `include` parameter value `http://www.w3.org/ns/ldp#PreferMinimalContainer` so that the response doesn't include unnecessary data.

The following example shows a container that supports creation and selection dialogs:

EXAMPLE 5

```
GET /bugs/ HTTP/1.1
Host: example.com
Accept: text/turtle
Prefer: return=representation;
        include="http://open-services.net/ns/core#PreferDialog http://www.w3.org/ns/ldp#PreferMinimalContainer"
```

EXAMPLE 6

```
HTTP/1.1 200 OK
Content-Type: text/turtle
ETag: "_87e52ce291112"
Link: <http://www.w3.org/ns/ldp#BasicContainer>; rel="type",
      <http://www.w3.org/ns/ldp#Resource>;    rel="type"
Accept-Post: text/turtle, application/ld+json
Allow: POST,GET,OPTIONS,HEAD
Preference-Applied: return=representation
Vary: Accept,Prefer
Transfer-Encoding: chunked

@prefix ex:     <http://example.com/vocab#> .
@prefix oslc:  <http://open-services.net/ns/core#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix ldp:    <http://www.w3.org/ns/ldp#> .

<http://example.com/bugs/>
        a                   ldp:BasicContainer ;
        oslc:creationDialog  <http://example.com/dialogs/createBug> ;
        oslc:selectionDialog <http://example.com/dialogs/selectBug> ;
        dcterms:title       "Bugs Records for Product Z" .

<http://example.com/dialogs/createBug>
        a                   oslc:Dialog ;
        oslc:dialog         <http://example.com/dialogs/createBug/form> ;
```

```
        oslc:hintHeight    "600px" ;
        oslc:hintWidth     "400px" ;
        oslc:label         "New Bug" ;
        oslc:resourceType  <http://open-services.net/ns/cm#Bug> ;
        dcterms:title      "Report Bug (Product Z)" .

<http://example.com/dialogs/selectBug>
        a                  oslc:Dialog ;
        oslc:dialog        <http://example.com/dialogs/selectBug/form> ;
        oslc:hintHeight    "600px" ;
        oslc:hintWidth     "400px" ;
        oslc:label         "Select Bug" ;
        oslc:resourceType  <http://open-services.net/ns/cm#Bug> ;
        dcterms:title      "Select Bug (Product Z)" .
```

2.3 Discovering Dialogs Using the Service resource

This section is non-normative.

Servers may also advertise their support for dialogs using the `oslc:creationDialog` and `oslc:selectionDialog` properties of the Service discovery resource. Each dialog type, creation or selection, has its own property:

Dialog Type	Link Relation
Creation	`http://open-services.net/ns/core#creationDialog`
Selection	`http://open-services.net/ns/core#selectionDialog`

The client discovers the dialogs by making a GET request on the Service resource and accessing these properties.

EXAMPLE 7: An OPTIONS Request on the Container

```
GET /serviceproviders/bugs/services.xml HTTP/1.1
Host: example.com
Accept: application/rdf+xml
```

The server response contains a Service resource which contains properties with URLs to the dialog descriptors.

EXAMPLE 8: Service Resource

```
HTTP/1.1 200 OK
Date: Thu, 12 Jun 2014 18:26:59 GMT
Content-Type: application/rdf+xml

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:oslc="http://open-services.net/ns/core#">
<oslc:ServiceProvider rdf:about="https://example.com/serviceproviders/bugs/services.xml">
<oslc:creationDialog>
  <oslc:Dialog>
    <dcterms:title rdf:parseType="Literal">New Bug</dcterms:title>
    <oslc:label>Bug Change Request</oslc:label>
    <oslc:usage rdf:resource="http://open-services.net/ns/cm#requirementsChangeRequest"/>
    <oslc:resourceType rdf:resource="http://open-services.net/ns/cm#ChangeRequest"/>
    <oslc:dialog rdf:resource="http://example.com/dialogs/createBug/form"/>
    <oslc:hintWidth>680px</oslc:hintWidth>
    <oslc:hintHeight>505px</oslc:hintHeight>
  </oslc:Dialog>
</oslc:creationDialog>
<oslc:selectionDialog>
  <oslc:Dialog>
    <dcterms:title rdf:parseType="Literal">Select Bug</dcterms:title>
    <oslc:label>Bug</oslc:label>
    <oslc:usage rdf:resource="http://open-services.net/ns/core#default"/>
    <oslc:resourceType rdf:resource="http://open-services.net/ns/cm#ChangeRequest"/>
    <oslc:dialog rdf:resource="http://example.com/dialogs/selectBug/form"/>
    <oslc:hintWidth>550px</oslc:hintWidth>
    <oslc:hintHeight>460px</oslc:hintHeight>
  </oslc:Dialog>
</oslc:selectionDialog>
</oslc:ServiceProvider>
</rdf:RDF>
```

The client then requests the dialog to display using the discovered URI in the `oslc:dialog` property in order to display the desired form.

EXAMPLE 9

```
GET /dialogs/selectBug/form HTTP/1.1
Host: example.com
```

3. Using Dialogs

This section is non-normative.

Servers can offer two kinds of dialogs, selection dialogs and creation dialogs. Clients use selection dialogs when they want a user to

pick a resource from another application. They use creation dialogs when they want a user to create a new resource in another application.

A client can open the dialog in a new browser window, or it can embed the dialog in another page by creating an `iframe` element and setting the `src` attribute to the URI of the dialog to be included.

EXAMPLE 10: Display a Dialog

```
var iframe =
      document.createElement("iframe");
   iframe.style.border = 0;
   iframe.style.width = "600px";  // or preferred dialog width
   iframe.style.height = "400px"; // or preferred dialog height
   iframe.src = "http://example.com/dialogs/createBug/form";
   document.getElementById("dialogContainer").appendChild(iframe);
```

Clients might choose to place dialogs inside page elements styled to look like a dialog window.

Fig. 2 Dialog iframe

The `iframe` itself, outlined in Fig. 2 Dialog iframe, contains the dialog content from the server.

Dialogs send results back to the client using the HTML5 function `Window.postMessage` [webmessaging]. `postMessage` is called on `window.opener` if set. Otherwise, `postMessage` is called on `window.parent`.

When embedded in an `iframe`, dialogs may also request to be resized dynamically. These requests are the same as specified in [OSLCPreview30] and also use `postMessage`. Clients can distinguish between results and resize messages from the message prefix. Messages have the following prefixes:

Message Prefix	Meaning
`oslc-response:`	The user has created or selected resources or canceled the dialog.
`oslc-resize:`	The dialog is asking to be resized.

A stringified JSON object will be concatenated to the message prefix. For dialog responses, the JSON is described by Appendix A. Dialog Results JSON. It is a JSON object with an `oslc:results` array of results. Each result is a JSON object with an `rdf:resource` property whose value is the resource URI. The result may also have an `oslc:label` that can be useful for client to display a label for the delegated dialog.

EXAMPLE 11: Dialog Results Message

```
oslc-response:{"oslc:results":
  [{ "oslc:label": "bug 123: server crash", "rdf:resource": "http://example.com/bug123" }]}
```

An empty array indicates the dialog was canceled.

For resize requests, the JSON is an object with an `oslc:hintHeight` property, an `oslc:hintWidth` property, or both.

EXAMPLE 12: Dyanmic Resize Message

```
oslc-resize:{"oslc:hintHeight": "277px", "oslc:hintWidth": "400px"}
```

3.1 The Client's Responsibilities

This section is non-normative.

1. Open the dialog in a new window using the `Window.open()` method or embed the dialog in an `iframe`, setting `iframe src` to the URI of the dialog.
2. Add a `message` listener to receive messages from the dialog.
3. Listen for `message` events, ignoring unrecognized events from other sources or those not prefixed with `oslc-response:`
4. When message from the dialog indicates a completed action, free resources and handle the action.

EXAMPLE 13: Listen for Dialog Results

```
window.addEventListener("message", function(event) {
    // Make sure the message originated from the same origin as the dialog.
    if (event.origin !== "http://example.com") {
        return;
    }

    // Make sure the message starts with oslc-response:
    var message = event.data;
    if (message.indexOf("oslc-response:") !== 0) {
        return;
    }

    // Handle each result.
    var response = JSON.parse(message.substr("oslc-response:".length));
    var results = response["oslc:results"];
    for (var i = 0; i < results.length; i++) {
        var label = results[i]["oslc:label"];
        var uri = results[i]["rdf:resource"];
        // Handle resource...
    }

    // Remove the dialog from the page.
    var dialogContainer = document.getElementById('dialogContainer');
    dialogContainer.removeChild(dialogContainer.firstChild);
}, false);
```

3.2 The Server's Responsibilities

This section is non-normative.

1. Provide the dialog, an HTML page for resource creation or selection.
2. Allow the user to perform resource creation or selection.
3. Once the user has created or selected a resource or canceled the dialog, send notification using `postMessage` to the page's opener or parent window prefixed with `"oslc-response:"`.

The following JavaScript code example shows how a dialog would send a response using `postMessage`, taking into account pages that the dialog might be in its own window or an `iframe`.

EXAMPLE 14

```
function respondWithSelection(label, uri) {
    var response = {
        "oslc:results": [ {
            "oslc:label": label,
            "rdf:resource": uri
        } ]
    };
    (window.opener || window.parent).postMessage("oslc-response:" + JSON.stringify(response), "*");
}
```

3.3 Prefill

This section is non-normative.

Servers may support setting the initial values in a dialog. A client can test if a dialog supports prefill by making an HTTP OPTIONS request to the dialog descriptor URI. Prefill can be used with creation dialogs to provide a template of initial values. Clients can use prefill with selection dialogs to inform servers about the users desired content in the selection dialog.

EXAMPLE 15

```
OPTIONS /dialogs/createBug HTTP/1.1
Host: example.com
```

If the server supports prefill for this dialog, it includes POST among the methods in the `Allow` response header.

EXAMPLE 16

```
HTTP/1.1 204 No Content
Allow: GET,POST,HEAD,OPTIONS
```

To prefill content, clients POST the resource representation with the desired initial values to the delegated dialog descriptor URI.

```
POST /dialogs/createBug HTTP/1.1
Host: example.com
Content-Type: text/turtle
Transfer-Encoding: chunked

@prefix oslc_cm: <http://open-services.net/ns/cm#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

<>
    a oslc_cm:Bug ;
    dcterms:title "Build 23 failed" ;
    oslc_cm:severity <http://example.com/enums#S1> .
```

The content of the request depends on the type of dialog and the server. Servers may describe constraints on POST content using resource shapes [OSLCCoreVocab]. The dialog descriptor uses property `oslc:resourceShape` for the shape constraints.

The server's response contains a `Location` header with the prefilled dialog's URI. The client uses this as the `iframe src`.

```
HTTP/1.1 201 Created
Location: http://example.com/dialogs/createBug/form/2zFy45
```

The client then uses the dialog URI from the `Location` like any other dialog. The dialog URI from a prefill request is often temporary. After a reasonable time, the server might respond with 404 Not Found or 410 Gone.

3.4 Security Issue: Clickjacking (UI redress attack)

This section is non-normative.

Clickjacking is a vulnerability that can occur when a malicious (or compromised) web application embeds an OSLC delegated dialog in the application in such a way that the user is tricked into using their authenticated session with the dialog, believing that they are performing an operation other than the one indicated on the delegated dialog. The clickjacking application embeds it own web page in a way that means the dialog is not clearly visible to the user, either almost invisible, or hidden behind other components). When the user attempts to click on the visible components, the browser interprets this as a click on the obscured delegated dialog, and the components within it. The malicious web page encourages the user to click one of the visible components, which the user might believe will not have side-effects (such as a link saying "If you are not redirected within 2 seconds, click here"), but places that component over a component in the delegated dialog that the malicious page wishes the user to click unknowingly. The user attempts to click on the visible component, but the browser interprets that as a click on the hidden component, which will perform some action that the user is authorised to do (but the malicious web page is not) but that the user did not intend to perform.

This vulnerability requires that the user loads a malicious web page in their browser. This can happen in a number of ways, including:

- The user is tricked into loading the page directly, believing it to be trustworthy.
- A trusted page is compromised and "infected" to perform this attack.
- An OSLC server administrator is tricked into configuring their server to trust a malicious server. Their server then loads the malicious page as a delegated dialog or rich UI preview, which then embeds the page under attack in a further iframe.

There are always two pages involved in this attack; the "UI consumer" page (which has the iframe embedded within it), and the "UI provider" page (which gets loaded within the iframe). Consider the UI provider's server the "server under attack".

To protect against this attack:

1. When requesting the URL for the delegated dialog (from the ServiceProvider) or the UI preview (from the compact resource), the server under attack must require requests for the ServiceProvider to be authenticated. This authentication must be for the end-user, not for the client app itself. That is, access to the dialog URIs should be protected, as well as access to the dialogs themselves.
2. When returning a URL for the dialog or preview, the server under attack must return a URL that is specific to the authenticated user, and that cannot be guessed by merely knowing the user's username or ID.
3. When the server under attack receives a request for the dialog or preview URL itself (the request that is generated by the iframe to load the page to display within it), the server under attack must check that the URL that was requested is one that was generated for the same user that is requesting this page (i.e. the user that is identified by a session ID in the cookies sent with the request). This makes sure that the client can't get a URL for the dialog from one user that it is authenticated for, and use that to perform a clickjacking attack on another user that has not authorized that client.
4. If the users (for the URL and the cookie) do not match, then the server can either return with an HTTP error, or include an `X-Frame-Options` header with value SAMEORIGIN or DENY.

This is only required on dialogs or resource previews that contain controls that can be invoked with one or more clicks, and where those controls have side effects on the server. That is, if there are no controls on a preview, then there is no target for the malicious page to tick the user to click on.

By following these steps the server under attack knows that only clients that are able to authenticate against the server (e.g. for OAuth authentication they must have a valid client ID and - if required by the interaction pattern they're using - they must know the client secret), and that have also been authorized by the user to whom the the dialog is displayed, can display that dialog.

There are still some potential attack vectors:

- The server administrator could authorize a malicious client app.
- The user could authorize a malicious app to authenticate with the server under attack on their behalf.

But the exposure has been reduced as the attack cannot occur simply by loading a malicious web page in the user's browser. See the OWASP Clickjacking Defense Cheat Sheet for further information.

4. Implementation Conformance

4.1 Discovery

4.1.1 In responses to successful HTTP requests for an LDP container that has a selection dialog, server **MUST** include a `Link` header [RFC5988] where

- The context URI is the effective request URI,
- The link relation is `http://open-services.net/ns/core#selectionDialog`, and
- The target URI is the URI of the selection dialog descriptor.

> EXAMPLE 19
>
> ```
> Link: <http://example.com/dialogs/selectBug>; rel="http://open-services.net/ns/core#selectionDialog"
> ```

4.1.2 In responses to successful HTTP requests for an LDP container that has a creation dialog, server **MUST** include a `Link` header [RFC5988] where

- The context URI is the effective request URI,
- The link relation is `http://open-services.net/ns/core#creationDialog`, and
- The target URI is the URI of the creation dialog descriptor.

> EXAMPLE 20
>
> ```
> Link: <http://example.com/dialogs/createBug>; rel="http://open-services.net/ns/core#creationDialog"
> ```

4.1.3 Clients **MAY** request that the dialog descriptors for an LDP container are returned inline using the `Prefer` request header [RFC7240] with:

- Preference `return`, value `representation`
- Parameter `include` [LDP], value `http://open-services.net/ns/core#PreferDialog`.

> EXAMPLE 21: Example Prefer Header
>
> ```
> Prefer: return=representation; include="http://open-services.net/ns/core#PreferDialog"
> ```

4.1.4 Servers **MUST** honor a client's request to inline dialog descriptors if the target resource has any dialog descriptors and the request is successful.

4.1.5 Servers **MUST** express the `oslc:hintWidth` and `oslc:hintHeight` properties of an `oslc:Dialog` in length units as specified in [CSS21].

4.1.6 In responses to HTTP GET requests targeting resources that have dialogs, servers **SHOULD** either include a `Vary` response header with at least `Accept` and `Prefer` field values or a `Cache-Control` header value `no-store`.

4.1.7 Servers **MAY** also provide delegated dialog discovery using the ServiceProvider and Service resource and the `oslc:creationDialog` and `oslc:selectionDialog` properties.

4.2 Display

4.2.1 When embedding a dialog inside another web page, the client **MUST** use an `iframe` element and set the its `src` attribute to the URI of the dialog.

4.3 Messaging

4.3.1 Servers **MUST** support the `Window.postMessage` method [webmessaging] for dialog responses.

4.3.2 Servers **MAY** support the Window Name Protocol defined in [OSLCCore2] for backward compatibility with OSLC 2.0 clients that use that protocol.

4.3.3 Servers **MAY** support other protocols for dialog responses not specified in this document, which clients request by appending a fragment identifier [RFC3986] describing the protocol to the end of the dialog URI.

4.3.4 Servers **MUST** use `postMessage` for dialog responses if a client has not added a fragment identifier to the dialog URI, or the

fragment identifier is `#oslc-core-postMessage-1.0`.

4.3.5 Messages describing dialog results **MUST** start with the characters `oslc-response:` followed by JSON as specified in Appendix A. Dialog Results JSON.

EXAMPLE 22

```
oslc-response:{"oslc:results":
  [{ "oslc:label": "bug 123: server crash", "rdf:resource": "http://example.com/bug123" }]}
```

4.3.6 Servers **MAY** include additional server-specific properties in the results JSON.

4.3.7 If the user cancels a dialog, the dialog **MUST** still send a message with an empty `oslc:results` array.

EXAMPLE 23

```
oslc-response:{"oslc:results": []}
```

4.3.8 Calls to `postMessage` from within a dialog **MUST** be made on `window.opener` unless `null` or `undefined`.

4.3.9 If `window.opener` is `null` or `undefined`, calls to `postMessage` **MUST** be made on `window.parent`.

EXAMPLE 24

```
(window.opener || window.parent).postMessage("oslc-response:" + response, "*");
```

4.3.10 Clients handling `message` events from dialogs **MUST** verify that the event `origin` is the same as the dialog.

For example, if the dialog is from example.com,

EXAMPLE 25

```
function handleMessage(event) {
    if (event.origin !== "http://example.com") {
        return;
    }

    // Otherwise, process message...
}
```

4.3.11 Dialogs **MAY** support dynamic resizing as specified in [OSLCPreview30].

4.3.12 Clients **MUST** anticipate other, unrelated uses of `postMessage` and ignore unrecognized messages not conforming to the protocol.

4.4 Prefill

4.4.1 Servers **MAY** support prefill for creation and/or selection dialogs. Client provided prefill information, either in a request entity body or query parameters is intended to inform servers of possible initial values for creation dialogs, or the preferred content of selection dialogs. Clients **SHOULD NOT** assume any specific content or URL format.

4.4.2 Servers **MAY** allow clients to prefill dialogs by accepting HTTP POST requests where the Request-URI is the dialog descriptor and the entity body is an entity describing the initial values.

4.4.3 Servers that support prefill **MUST** include the value **POST** in the set of `Allow` header values returned in response to HTTP OPTIONS requests for the dialog descriptor URI.

EXAMPLE 26

```
Allow: GET,POST,HEAD,OPTIONS
```

4.4.4 Servers **MAY** describe prefill constraints by adding an `oslc:resourceShape` property to the dialog descriptor whose value is a resource shape URI.

4.4.5 Servers **MUST NOT** reject prefill requests on creation dialogs solely because they are missing required fields for the resource. Users can fill in the missing values in the dialog.

4.4.6 On successful prefill requests, servers **MUST** respond with status code 201 (Created) and a `Location` header whose value is the URI of the prefilled dialog.

4.4.7 After some elapsed time, servers **MAY** respond with a 404 (Not Found) or 410 (Gone) to an HTTP GET request for a prefilled

dialog URI.

4.4.8 Servers **MAY** support prefill by adding initial values as query parameters to the delegated dialog URI. Clients **SHOULD NOT** assume any specific URL format, however.

5. Resource Constraints

This document applies the following constraints to the Dialog [OSLCCoreVocab] vocabulary term.

- **Name:** `core#Dialog`
- **URI:** `http://open-services.net/ns/core#Dialog`
- **Summary:** Describes information about a dialog such as its title and dimensions.

core#Dialog Properties

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
`dcterms:title`	Exactly-one	true	XMLLiteral	N/A	Unspecified	Title string that could be used for display.
`oslc:dialog`	Exactly-one	true	string	N/A	Unspecified	The URI of the dialog.
`oslc:hintHeight`	Zero-or-one	true	string	N/A	Unspecified	Recommended height of the dialog. Values are expressed using length units as specified in [CSS21].
`oslc:hintWidth`	Zero-or-one	true	string	N/A	Unspecified	Recommended width of the dialog. Values are expressed using length units as specified in [CSS21].
`oslc:label`	Zero-or-one	true	string	N/A	Unspecified	Very short label for use in menu items.
`oslc:resourceShape`	Zero-or-many	true	Resource	Reference	`oslc:ResourceShape`	Describes constraints on dialog prefill requests.
`oslc:resourceType`	Zero-or-many	true	Resource	Reference	`rdfs:Class`	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 `rdf:type` property.
`oslc:usage`	Zero-or-many	true	Resource	Reference	Unspecified	An identifier URI for the domain specified usage of this dialog. If a resource has multiple uses, it may designate the primary or default one that should be used with a property value of oslc:default.

Appendix A. Dialog Results JSON

Dialog results are represented as JSON [RFC4627]. The top-level JSON object has an `oslc:results` property.

Property	Type	Occurs	Description
`oslc:results`	JSON Array	Exactly-one	An array of results, each result a JSON object. An empty array means the user canceled the dialog or didn't select a resource.

Each result is a JSON object with the following properties.

Property	Type	Occurs	Description
`rdf:resource`	String	Exactly-one	URI of the resource selected or created
`oslc:label`	String	Zero-or-one	Short label describing the resource selected

Here is an example response that contains two resources.

```
EXAMPLE 27

{
    "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"
        }
    ]
}
```

When sent as a message in calls to `postMessage`, the JSON string is prefixed with the characters `oslc-response:`.

Appendix B. Change History

This section is non-normative.

Revision	Date	Editor	Changes Made
1	07/06/2016	Jim Amsden	CSD was approved and published.