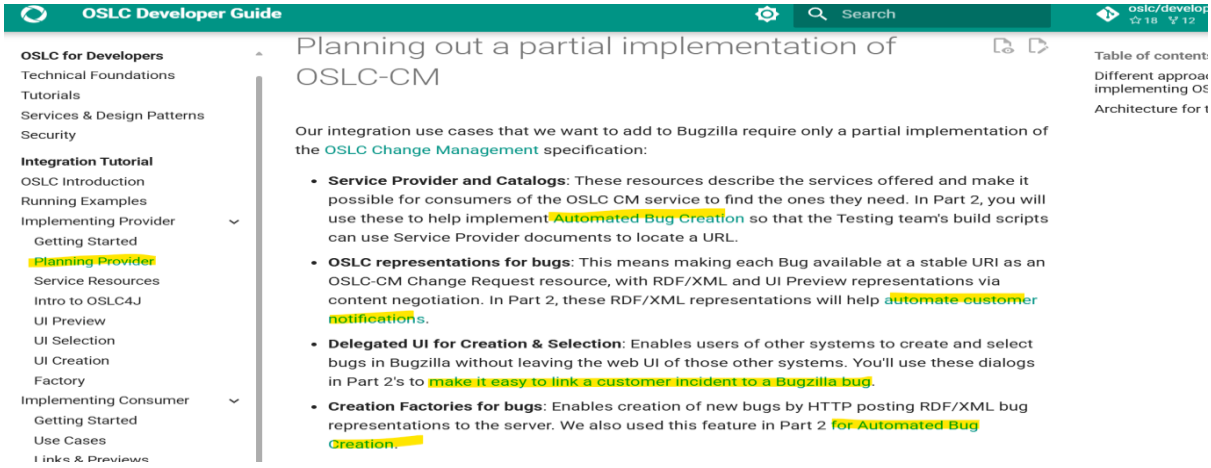
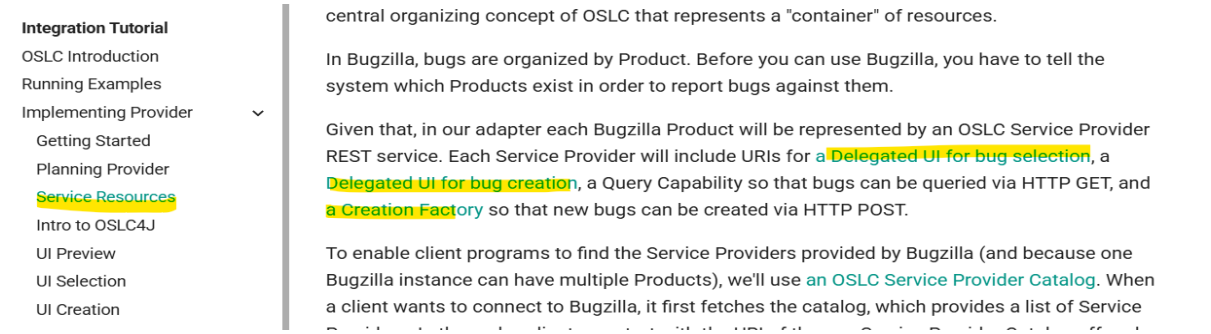
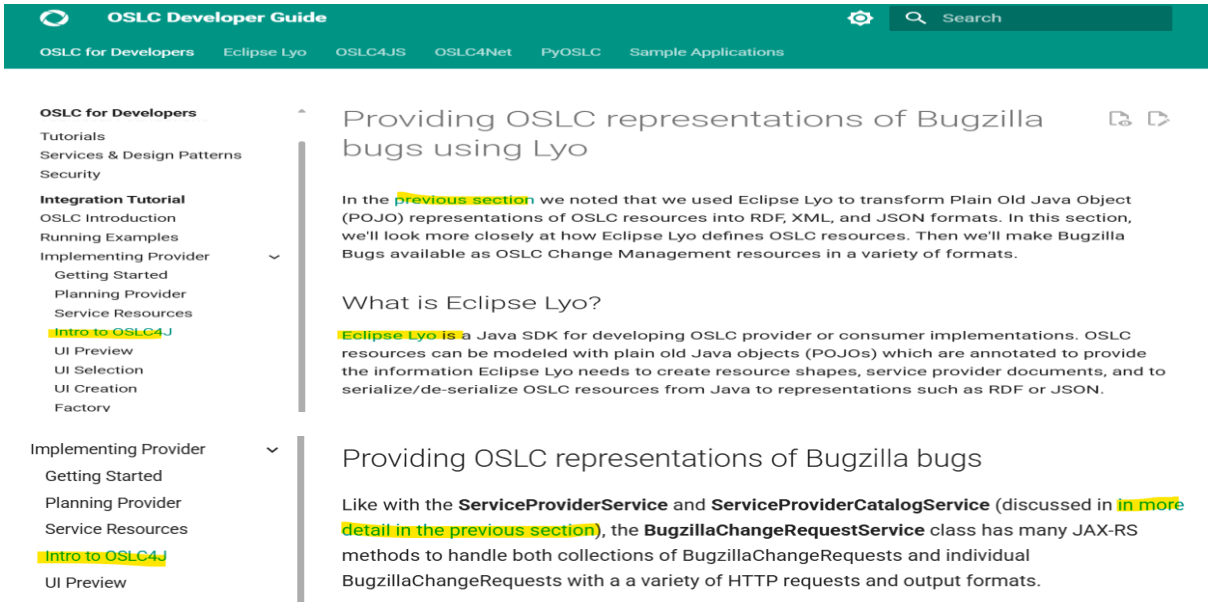

OSLC Developer Guide
Search
OSLC for Developers
Technical Foundations
Tutorials
Services & Design Patterns
Security
Integration Tutorial
OSLC Introduction
Running Examples
Implementing Provider
Getting Started
Planning Provider
Service Resources
Intro to OSLC4J
UI Preview
UI Selection
UI Creation
Factory
Implementing Consumer
Getting Started
Use Cases
Planning out a partial implementation of OSLC-CM
Table of content
Our integration use cases that we want to add to Bugzilla require only a partial implementation of the OSLC Change Management specification:
Service Provider and Catalogs: These resources describe the services offered and make it possible for consumers of the OSLC CM service to find the ones they need. In Part 2, you will use these to help implement Automated Bug Creation so that the Testing team's build scripts can use Service Provider documents to locate a URL.
OSLC representations for bugs: This means making each Bug available at a stable URI as an OSLC-CM Change Request resource, with RDF/XML and UI Preview representations via content negotiation. In Part 2, these RDF/XML representations will help automate customer notifications.
Delegated UI for Creation & Selection: Enables users of other systems to create and select bugs in Bugzilla without leaving the web UI of those other systems. You'll use these dialogs in Part 2's to make it easy to link a customer incident to a Bugzilla bug.
Creation Factories for bugs: Enables creation of new bugs by HTTP posting RDF/XML bug representations to the server. We also used this feature in Part 2 for Automated Bug Creation.
In our case, building an adapter makes the most sense.
Architecture for the adapter
Back to top
Download the OSLC4J Bugzilla adapter. We'll be exploring the adapter instead of writing one from scratch.
The OSLC4J Bugzilla adapter is a RESTful web application built on Java EE with JAX-RS. It has the following additional dependencies:
OSLC4J SDK part of Eclipse Lyo, OSLC4J is a Java toolkit that simplifies building OSLC applications
J2Bugzilla: Java wrapper classes for Bugzilla's XML-RPC based web services interface
In addition, it uses the following helper classes (in the utils directory):
BugzillaHttpClient: helper classes for doing HTTP GET requests against a Bugzilla server
HttpUtils: helper classes for working with HTTP requests and responses
StringUtils: helper classes for dealing with strings
XmlUtils: helper classes for XML processing


Integration Tutorial
OSLC Introduction
Running Examples
Implementing Provider
Getting Started
Planning Provider
Service Resources
Intro to OSLC4J
UI Preview
UI Selection
UI Creation
central organizing concept of OSLC that represents a "container" of resources.
In Bugzilla, bugs are organized by Product. Before you can use Bugzilla, you have to tell the system which Products exist in order to report bugs against them.
Given that, in our adapter each Bugzilla Product will be represented by an OSLC Service Provider REST service. Each Service Provider will include URIs for a Delegated UI for bug selection, a Delegated UI for bug creation, a Query Capability so that bugs can be queried via HTTP GET, and a Creation Factory so that new bugs can be created via HTTP POST.
To enable client programs to find the Service Providers provided by Bugzilla (and because one Bugzilla instance can have multiple Products), we'll use an OSLC Service Provider Catalog. When a client wants to connect to Bugzilla, it first fetches the catalog, which provides a list of Service


OSLC Developer Guide
Search
OSLC for Developers
Eclipse Lyo
OSLC4JS
OSLC4Net
PyOSLC
Sample Applications
OSLC for Developers
Tutorials
Services & Design Patterns
Security
Integration Tutorial
OSLC Introduction
Running Examples
Implementing Provider
Getting Started
Planning Provider
Service Resources
Intro to OSLC4J
UI Preview
UI Selection
UI Creation
Factory
Providing OSLC representations of Bugzilla bugs using Lyo
In the previous section we noted that we used Eclipse Lyo to transform Plain Old Java Object (POJO) representations of OSLC resources into RDF, XML, and JSON formats. In this section, we'll look more closely at how Eclipse Lyo defines OSLC resources. Then we'll make Bugzilla Bugs available as OSLC Change Management resources in a variety of formats.
What is Eclipse Lyo?
Eclipse Lyo is a Java SDK for developing OSLC provider or consumer implementations. OSLC resources can be modeled with plain old Java objects (POJOs) which are annotated to provide the information Eclipse Lyo needs to create resource shapes, service provider documents, and to serialize/de-serialize OSLC resources from Java to representations such as RDF or JSON.
Implementing Provider
Getting Started
Planning Provider
Service Resources
Intro to OSLC4J
UI Preview
Providing OSLC representations of Bugzilla bugs
Like with the ServiceProviderService and ServiceProviderCatalogService (discussed in in more detail in the previous section), the BugzillaChangeRequestService class has many JAX-RS methods to handle both collections of BugzillaChangeRequests and individual BugzillaChangeRequests with a a variety of HTTP requests and output formats.

Intro to OSLC4J

UI Preview

UI Selection

In the previous section we explored how the **BugzillaChangeRequestService** class handles requests for collections of **BugzillaChangeRequests** or individual **BugzillaChangeRequests**.

UI Preview
UI Selection
UI Creation
Factory
Implementing Consumer
Getting Started
Use Cases
Links & Previews
Delegated UI
Notify Customers
Automatic Bugs

We'll put this dialog to use in the NinaCRM application later on.

We now have the ability to use user interface delegation as a way to provide a simple way for consumer applications to both create and select bugs. We've also exposed this capability from our service provider resource definition.

Next, we'll explore how to make it easier for other applications to create new bugs programmatically.

Next: Part 1.7, Factory

June 14, 2025 May 23, 2019

OSLC for Developers
Tutorials
Services & Design Patterns
Security
Integration Tutorial
OSLC Introduction
Running Examples
Implementing Provider
Getting Started
Planning Provider
Service Resources
Intro to OSLC4J
UI Preview
UI Selection
UI Creation
Factory

Providing a creation factory

With OSLC you can allow people to create bugs via Delegated UI; however, like all UI approaches, an actual human user must be involved. What if you want to support automated bug creation; for example, enabling a build server to automatically create a bug whenever there is a test or a build failure?

To allow clients to create new bugs automatically, you need to support an OSLC Creation Factory as described in the OSLC Core specification.

Adding a method to the adapter to create BugzillaChangeRequests via HTTP POST

Recall that when we created a delegated UI for creating new bugs, we wrote code in the **BugzillaManager** class to use the **j2bugzilla** API for creation of bugs; we'll re-use the `createBug()` method for automated bug creation via POST.

Implementing Consumer
Getting Started
Use Cases
Links & Previews
Delegated UI
Notify Customers
Automatic Bugs

5. The OSLC provider returns HTML that you can show to the user.

We explored the OSLC Provider side of this in more detail earlier in this tutorial.

Example XML for a UI preview

Here's an example of the XML that an OSLC Provider will return when you request the UI Preview representation of a resource:

Implementing Consumer
Getting Started
Use Cases
Links & Previews
Delegated UI
Notify Customers
Automatic Bugs

Starting the NinaCRM sample application and the Bugzilla adapter.

Open http://localhost:8181/ninacrm/ in a web browser. You'll see a sample incident:

Sample incident #676 in the NinaCRM sample application

At the bottom, find the **Related Defects** heading. This is where we show links to related bugs in Bugzilla; the HTML is a simple unordered list:

Implementing Consumer
Getting Started
Use Cases
Links & Previews
Delegated UI
Notify Customers
Automatic Bugs

Showing UI Previews via Dojo Tooltip Widgets

Throughout this, we'll be using the Dojo JavaScript toolkit to smooth out browser differences and build UI components like buttons and tooltips.

Open the file `index.jsp` in `/src/main/webapp/` and search for `dojo.addOnLoad(addPreviewMouseOverHandlers)`.

Here, when the page is done loading we use the `dojo.query()` method to get all the links on the

Implementing Consumer
Getting Started
Use Cases
Links & Previews
Delegated UI
Notify Customers
Automatic Bugs

Earlier in this tutorial, we walked through an implementation of Delegated UIs for OSLC4JBugzilla, both for selecting bugs and creating new bugs. In addition to providing the UI and handling the results, the OSLC4JBugzilla adapter (or any other OSLC provider application) announces in its Service Provider Documents the URL location and recommended size of the UI.

The application that wants to use the Delegated UI (the OSLC Consumer) creates an `<iframe>` for the Delegated UI so that the user can interact with it. The Consumer application must also listen to the `<iframe>` do something with the results of the user's actions.

UI Creation
Factory
Implementing Consumer
Getting Started
Use Cases
Links & Previews
Delegated UI
Notify Customers
Automatic Bugs

work with any OSLC Provider.

As we noted when we implemented Service Providers and Catalogs, one of the cores of OSLC is that clients should not have to hard-code any URLs other than a Service Provider Catalog. Clients should be able to parse the Catalog and navigate from the Catalog to the Service Providers; the Service Providers will then expose the available OSLC services.

If you'd like to follow along with a real Service Provider Catalog or Service Provider, see the "Viewing the machine-readable formats of a Service Provider Catalog" section near the bottom of this section.