

Supplement for OSLC Workshop

Tracked Resource Set Provider for Bugzilla

Lab Exercises

Contents

LAB 0	GETTING ADDITIONAL SETUP	7
	0.1 SET UP THE LAB ENVIRONMENT	7
LAB 1	ENABLING TRACKED RESOURCE SET PROVIDER	8
	1.1 UNDERSTAND AND EXPLORE THE LYO TRS TOOLKIT	8
	1.2 CREATE A PRIMARY TRS SERVICE	10
	1.3 BUILDING TRACKED RESOURCE SET SERVICES	14
	1.4 SUMMARY	18
LAB 2	CONNECT TO JAZZ TEAM SERVER (OPTIONAL)	19
	2.1 ENHANCE OAUTH IMPLEMENTATION FOR INTRA-SERVER COMMUNICATION	19
	2.2 ENHANCE JAZZ ROOT SERVICES SUPPORT	22
	2.3 SUMMARY	24
APPENDIX A.	TROUBLESHOOTING	25
APPENDIX B.	SOURCES	26
APPENDIX C.	LEGAL	27
	1. NOTICES	27
	2. ABOUT THIS CONTENT	27

Overview

Objectives: In this lab, you'll go through steps to enable Bugzilla as Tracked Resource Set Provider. This will make Bugzilla capable of sending Change Logs.

Pre-requisites:

- *Some Java, JavaScript, Java Servlet, Java Server Pages (JSP) and HTML programming experience within an Eclipse IDE. Experience with JAX-RS Web Services and Resource Description Framework (RDF) a plus.*
- *Open Services for Lifecycle Collaboration (OSLC) and its technologies such as OSLC Resource, Resource Shape, and OSLC REST APIs. OSLC (<http://open-services.net/>) is a community that is working to standardize the way that software lifecycle tools can share data. As Tracked Resource Set (TRS) refers OSLC, you should have understanding of them.*
- *OAuth (<http://oauth.net/core/1.0/>). You should have basic understanding of OAuth because TRS client utilizes OAuth protocol to access TRS providers. Lab 6 in the Lyo OSLC Workshop provides how to add OAuth support to your adapter.*
- *(Optional) Completion of the Lyo OSLC Workshop (<http://wiki.eclipse.org/Lyo/OSLCWorkshop>). This document refers to the workshop a lot for concepts around OSLC and Eclipse Lyo OSLC4J. The completion of the workshop will help your understanding.*

Length: 1-2 hours depending on which optional labs you include

Introduction

This workshop will focus on the implementation of a Tracked Resource Set (TRS) provider. It is intended to provide a foundation in the skills required for developing Tracked Resource Set Provider implementations.

TRS is a protocol designed for servers to expose a set of resources in a way that allows clients to discover all additions, removals and modifications to the resource. A Client utilizes this protocol to keep the information about the tracked resources (i.e. OSLC resources) updated. As the TRS is based on OSLC, this workshop is designed as the supplement for the Lyo OSLC Workshop and depends on its OSLC provider/consumer functions. So you should have understanding of OSLC.

OSLC (<http://open-services.net/>) is a community that is working to standardize the way that software lifecycle tools can share data and provides specifications for that. It includes OSLC resource definition, resource shape, and REST APIs for API-based programmatic resource

access, and delegated UI and OSLC preview for Web-based UI integration. This workshop refers to the concept of RDF-based OSLC resource definition.

The example is based on a typical usage. Everyone's environment is unique, and the set of integration scenarios may need to be customized. Following these exercises will provide insight into how a common pattern can be used to implement a TRS provider for an existing tool. This has the advantage of providing additional capability to an existing tool without requesting vendor changes or locating tool source code. You will look at alternatives along the way but focus on this key pattern.

The objective of this workbook is to walk you an already-working implementation, so you can read and try this code by using a REST Client and Java debugger if you need. You will have access to the both source code of the Lyo OSLC Workshop project (OSLC4JBugzilla) and this TRS provider workbook project (org.eclipse.lyo.oslc4j.bugzilla.trs) so that you can see exactly what changes are made and to where. Also each Lab has the list of the Java and/or other type of files where the changes are made. These will help you to understand what needs to be implemented to enable an existing OSLC provider as a TRS provider.

The labs in this workshop make extensive use of OSLC4J and Tracked Resource Set (TRS) SDK from the Eclipse Lyo project and the projects in the Lyo OSLC Workshop. OSLC4J is a Java SDK for developing OSLC integrations. For more information, see:

<http://wiki.eclipse.org/Lyo/LyoOSLC4J>

<http://wiki.eclipse.org/Lyo/OSLCWorkshop>

<http://wiki.eclipse.org/Lyo/TRSToolkit>

Also, the TRS specification is available at:

<http://open-services.net/wiki/core/TrackedResourceSet-2.0/>

Scenario

In addition to the scenarios documented in the Lyo OSLC Workshop, Nina is looking at the way to track Bugs in her local replica, she will try to implement a TRS for Bugzilla so that she can keep her replica updated with the Bugs.

Labs:

Completing all labs might require more than the time allotted for this workshop.

Lab 1: Enabling Tracked Resource Set Provider – this lab provides an introduction to defining OSLC resources for TRS and REST services using OSLC4J.

Lab 2: Connect to Jazz Team Server (optional) – this lab provides steps needed for integration with TRS clients.

Topics not covered:

There are a number of topics that will not be covered due to time constraints of this workshop. They are important concepts and are critical in supporting some scenarios. Some of the concepts **excluded** are: In addition to what is stated in the Lyo OSLC Workshop;

1 “Delete” Change Event:

Currently, Bugs in Bugzilla can be deleted when the container Product or Component is deleted. However, once being deleted, Bugzilla deletes the associated history too so that it can't tell us the deleted Bug as well as its history anymore.






In order to support the “Delete” Change event, Tracked Resource Set Provider needs to keep the history of the deleted Bugs. For example, there are various areas of Bugzilla that an extension can hook into, which allow the extension to run code during that point in Bugzilla's execution. If we hook “object_before_delete”, you can keep the all history data of Bug just before it is going to be deleted, so that Tracked Resource Set Provider can record that event anyway. However the actual implementation is beyond the scope of this workshop.

- 2 Concurrent access to the Resources: A clever TRS client will spawn several threads to access TRS and the referred OSLC Resources. So that TRS provider needs to be “concurrent access safe”. We achieve that in this workshop by serializing the concurrent requests using mutual exclusion (Java's “synchronized” block).

Icons

The following symbols appear in this document at places where additional guidance is available.

Icon	Purpose	Explanation
	Important!	This symbol calls attention to a particular step or command. For example, it might alert you to type a command carefully because it is case sensitive.
	Information	This symbol indicates information that might not be necessary to complete a step, but is helpful or good to know.
	Trouble-shooting	This symbol indicates that you can fix a specific problem by completing the associated troubleshooting information.

Lab 0 Getting additional setup

Objectives:

Perform the necessary steps to get the environment set up and ready for development and testing.

0.1 Set up the lab environment

Please see the Eclipse Lyo project Wiki for setup instructions:

<http://wiki.eclipse.org/Lyo/BuildTRS4JBugzilla>

Lab 1 Enabling Tracked Resource Set Provider

Objectives:

- Understand the concept of Tracked Resource Set (TRS) provider
- Explore OSLC resources for Tracked Resource Set (TRS) in Lyo TRS toolkit
- Explore using JAX-RS to provide OSLC REST services

Description:

TRS is a protocol designed for servers to expose a set of resources in a way that allows clients to discover all additions, removals and modifications to the resources. A TRS client utilizes this protocol to keep the information about the tracked resources (i.e. OSLC resources) updated. The specification, OSLC Tracked Resource Set Specification Version 2.0, can be found at:

<http://open-services.net/wiki/core/TrackedResourceSet-2.0/>

OSLC (<http://open-services.net/>) is a community that is working to standardize the way that software lifecycle tools can share data and provides specifications for that. It includes OSLC resource definition, resource shape, and REST APIs for API-based programmatic resource access, and delegated UI and OSLC preview for Web-based UI integration. This workshop refers to the concept of RDF-based OSLC resource definition. The TRS is based on the OSLC resource concepts and built top of it.

TRS Provider provides a set of links to all resources available at a particular point of time (base resource) and a list of change log about OSLC resource CUD (creation, update, deletion) operation (change log resource).

A **Client** utilizes a Tracked Resource Set for such as keeping its own local replica of Resources in the Tracked Resource Set. The client retrieves that which OSLC resources in the replica are required to be updated, obtains the content of the resources using REST API (HTTP GET) for the OSLC resources, and then update the replica with the new contents. The following items are the brief summary:

- TRS provides the fact that resources are created, changed or modified
- TRS only provides references (links) to OSLC resources. Content to be replicated by a client are obtained via OSLC resource REST API
 - This implies that you need to implement OSLC resource (at least HTTP GET) before start implementing TRS
- Different from history or typical notification event, a TRS resource does not include what property was changed, what was the previous value, or the new value

1.1 Understand and Explore the Lyo TRS Toolkit

TRS consists of several OSLC resources (TRS resources) to describe information for TRS:

- 1 Tracked Resource Set

- 2 Base
- 3 Change Log
- 4 Change Event

For the Bugzilla adapter, you have implemented OSLC4J-annotated Java classes for Bugzilla OSLC Change Request resource, which is an OSLC resource, in Lab 1 in the Lyo OSLC Workshop. OSLC4J framework gives RDF and JSON serialization of the annotated Java classes.

Lyo TRS Toolkit provides OSLC4J-annotated Java beans for TRS resources. To learn the toolkit, see:

<http://wiki.eclipse.org/Lyo/TRSToolkit>

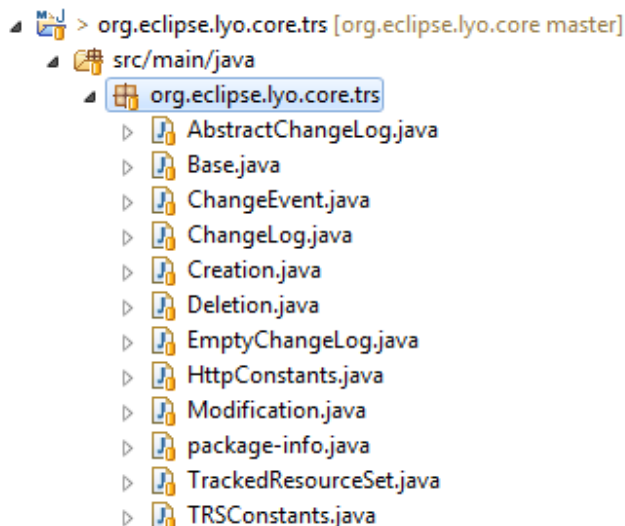
In this section, you will explore the TRS resource definitions provided by TRS toolkit so that you can get familiar with using them.

1.1.1 Changed Resources

This section doesn't involve any code changes. Just explore the Lyo TRS toolkit.

1.1.2 Steps

1. Find **org.eclipse.lyo.core.trs** package in org.eclipse.lyo.core.trs project. The package contains Java beans for TRS resources. Notice that each OSLC resource in the TRS specification has a corresponding Java bean in the package. Each bean class has Javadoc document in it and you can use it for the detail.



- ___2. Find **TrackedResourceSet.java** class in the `org.eclipse.lyo.core.trs` package and browse it and the javadoc to get familiar with the contents. This class represents Track Resource Set Resource. You can find there are properties (pairs of getter and setter methods) which are correspondent to Tracked Resource Set resource defined in the TRS specification. The significant properties and the annotations are:

```
/**
 * @return the base
 */
@OslcName(TRS_TERM_BASE)
@OslcDescription("An enumeration of the Resources in the Resource Set.")
@OslcPropertyDefinition(TRS_BASE)
@OslcTitle("Base")
public URI getBase() {
    return base;
}

/**
 * @return the changeLog
 */
@OslcName(TRS_TERM_CHANGE_LOG)
@OslcDescription("A Change Log providing a time series of incremental adjustments to the Resource Set.")
@OslcPropertyDefinition(TRS_CHANGE_LOG)
@OslcTitle("Change Log")
public AbstractChangeLog getChangeLog() {
    return changeLog;
}
```

- ___3. Find **Base.java** class in `org.eclipse.lyo.core.trs` package and browse it and the javadoc to get familiar with the contents. This class represents Base Resource. There are properties: `members`, `cutoffEvent`, and `nextPage`.

- ___4. Find **ChangeLog.java** class in `org.eclipse.lyo.oslc4j.core.trs` package and browse it and the javadoc to get familiar with the contents. There are `changes` and `previous` properties.

```
@OslcPropertyDefinition(Constants.TRS_NAMESPACE + "changes")
```

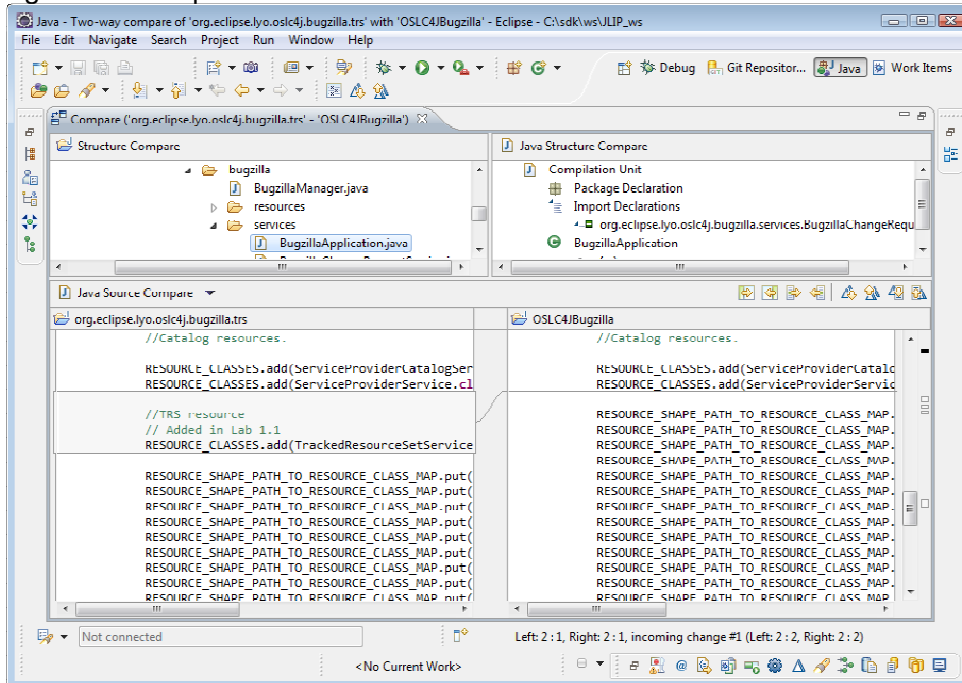
1.2 Create a primary TRS Service

We will start with implementing an outline of required elements for TRS. We will create a new JAX-RS service class.

1.2.1 Changed Resources

Changed resources can be found by comparing the original OSLC4JBugzilla project and the `org.eclipse.lyo.oslc4j.bugzilla.trs` project using Eclipse.

- ___a. Select the both projects in Project Explorer view
- ___b. Select “Compare With” > “Each Other” from the context menu
- ___c. In “Structure Compare” pane, files that have changes are listed and you can show file difference by double-clicking on a file name. You may ignore whitespace by clicking on “Ignore Whitespace” button.



The comparison contains all the changes to enable TRS in OSLC4JBugzilla. The resources changed particularly for this section are the following:

- ✓ **org.eclipse.lyo.oslc4j.bugzilla.services.TrackedResourceSetService.java**: New (only `getTrackedResourceSetText()` method)
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.services.BugzillaApplication.java**: Modified as explained below. (added `//TRS resource` and `RESOURCE_CLASSES.add(TrackedResourceSetService.class);` lines).

1.2.2 Steps

- ___1. You will create a TRS service as a JAX-RS service class. The class is an entry point to TRS REST service. In this step, the service is implemented as a stub. It will be implemented in the next section. The child steps show the class and required configuration to register the class to JAX-RS.
 - ___a. Open **TrackedResourceSetService.java**. The class is the JAX-RS service class created for TRS in the `org.eclipse.lyo.oslc4j.bugzilla.trs`. You'll have added `@Path("/trs")` annotation to the class. The value `"/trs"` is the TRS URI.

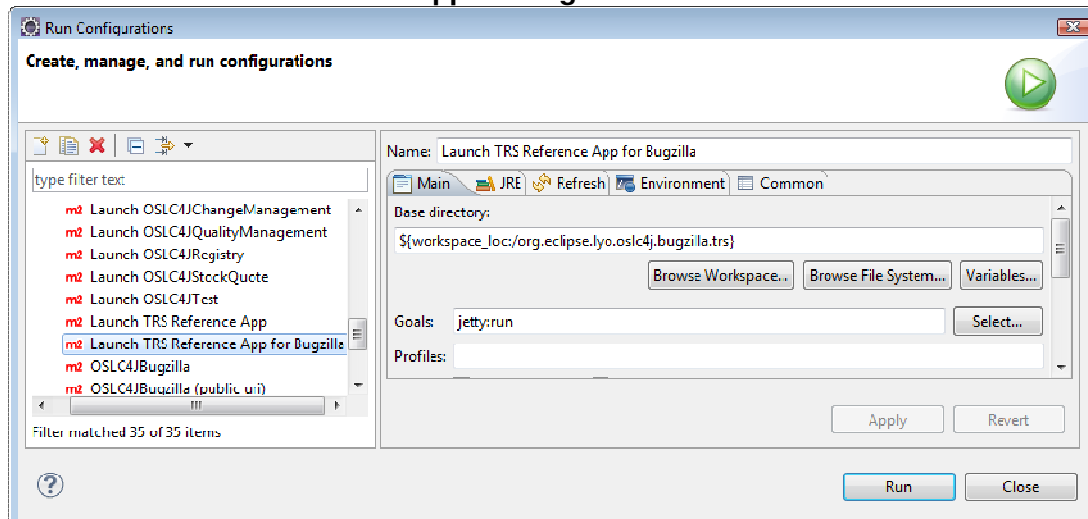
__b. Also, you'll have added a service method that responds to REST requests for the TRS resources. Take a look at the **getTrackedResourceSetText()** function. The function is mapped to the path **/trs** and invoked by HTTP GET method due to **@GET** annotation. The method has **@Produces** annotation, too. It indicates that when HTTP request is invoked with **Accept** header whose value is "text/html" or "text/plain", this method is selected.

__c. The service class needs to be registered to a JAX-RS registry. Open **BugzillaApplication.java** in **org.eclipse.lyo.oslc4j.bugzilla.services** package. The class is a JAX-RS services registry in **org.eclipse.lyo.oslc4j.bugzilla.trs**. There is a line for registering the **TrackedResourceSetService** to JAX-RS.

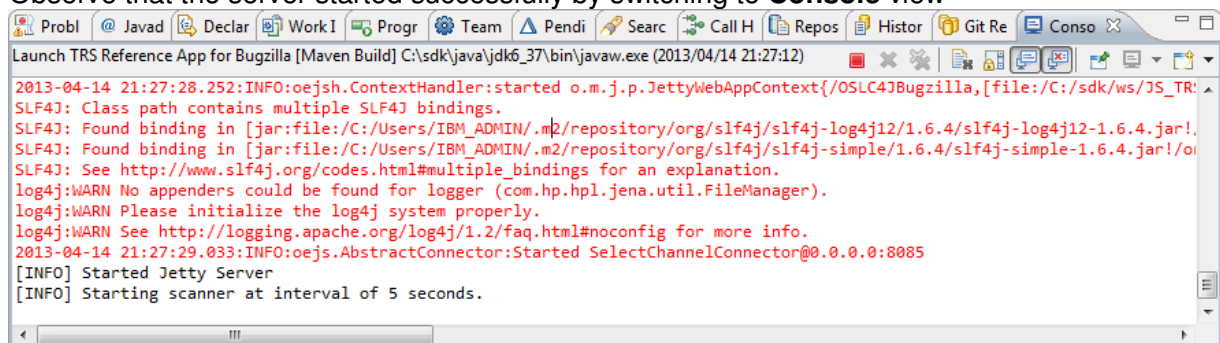
```
//TRS resource
RESOURCE_CLASSES.add(TrackedResourceSetService.class);
```

__2. Test the current changes.

__a. Select Run->Run Configurations menu item and then expand the Maven Build section and select **Launch TRS Reference App for Bugzilla** and click the **Run** button



Observe that the server started successfully by switching to **Console** view



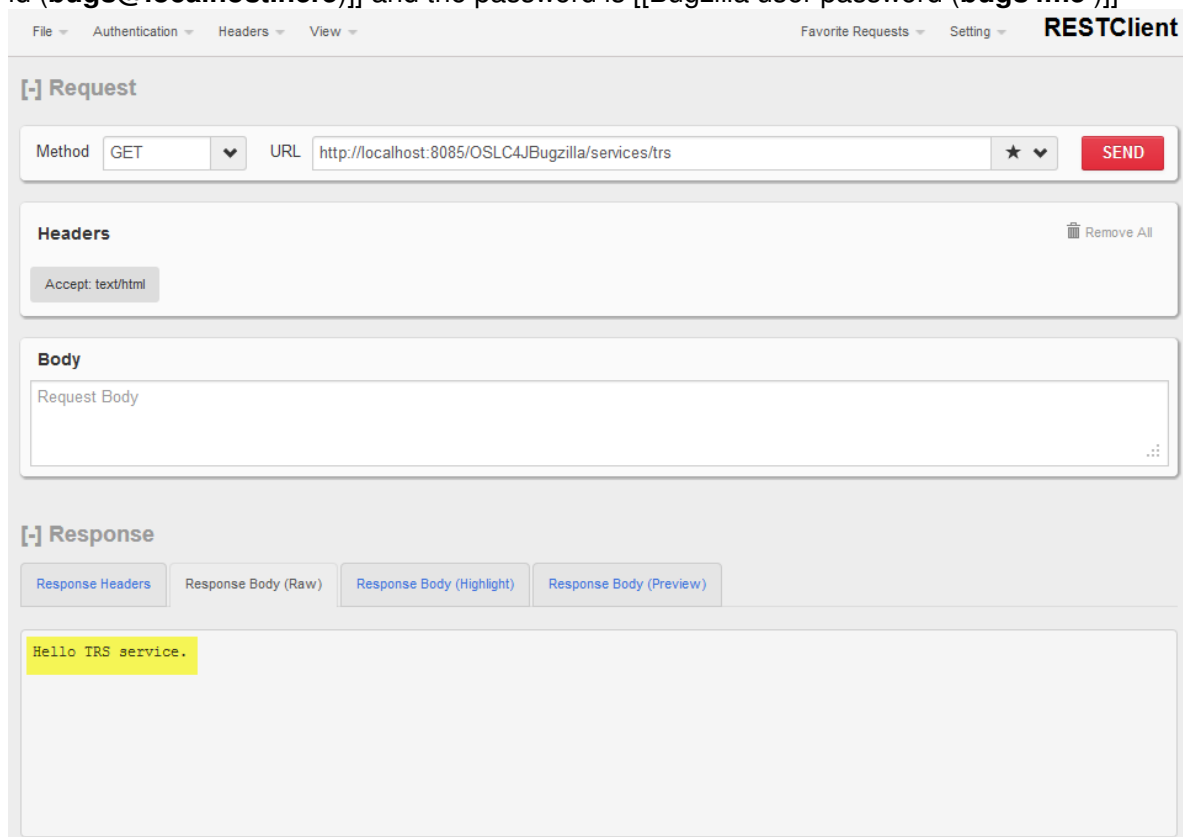
__b. Launch REST Client from Firefox by selecting Tools->REST Client

For this you'll use an add-in of Firefox called **REST Client** (<https://addons.mozilla.org/en-us/firefox/addon/restclient/>) to give you control over the requests being sent to your adapter

__c. Enter the **Request URL** to be <http://localhost:8085/OSLC4JBugzilla/services/trs> (Note: "localhost" is the host name where Tracked Resource Set Provider runs. This host name is also specified in org.eclipse.lyo.oslc4j.bugzilla.trs/src/main/resources/bugz.properties.)

__d. Click **Add Request Header** and add a new header with the name **Accept** and the value **text/html**

__e. Execute the HTTP GET method by clicking the **GET** action. You should see a page that only contains "Hello TRS service." in text. If you are prompted for a password, the user id is [[Bugzilla user id (**bugs@localhost.here**)]] and the password is [[Bugzilla user password (**bugs4me**)]]



REST Client

The look-and-feel of the REST Client in the screenshot above is different from the one in "OSLC Enable Your Tool in a Day" workshop. It's due to the version and you can continue to use the existing REST Client.

1.3 Building Tracked Resource Set services

In the section 1.1, you have learned that the OSLC4J TRS toolkit provides Java beans for the TRS resources and OSLC4J provides serialization of the resources. In the section 1.2, you have used JAX-RS for the service entry point. In this section, you'll combine them and build a TRS service for the Bugzilla adapter.

1.3.1 Changed Resources

All the changed resources for this section are the following:

- ✓ **org.eclipse.lyo.oslc4j.bugzilla.services.TrackedResourceSetService.java**: Modified (added five methods `getTrackedResourceSet()`, `getBase()`, `getBasePage()`, `getChangeLog()`, `getChangeLogPage()`, and import statements.)
- ✓ **org.eclipse.lyo.orls4j.bugzilla.trs** package: Added.
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.BugzillaManager.java**: Modified (added two methods named `getBugHistoryById()` and an import statement.)
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.resources.BugzillaChangeRequest.java**: Modified (modified `fromBug()` method and import statements for “see also” support.)
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.services.BugzillaChangeRequestService.java**: Modified (adapted to `BugzillaChangeRequest#fromBug()` method signature change.)
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.servlet.ServiceProviderFactory.java**: Modified. (added line `new PrefixDefinition(Constants.TRS_NAMESPACE_PREFIX, new URI(Constants.TRS_NAMESPACE))`).

1.3.2 Steps

1. Find **TrackedResourceSetService.java** class in `org.eclipse.lyo.oslc4j.bugzilla.services` package. Please find the **getTrackedResourceSet()** method. The method has **@GET** annotation so when TRS client invokes the path “/trs” this method is invoked. In the class, there is a static method invoked in the middle:

```
ChangeBugzillaHistories.buildBaseResourcesAndChangeLogs(httpServletRequest).
AbstractChangeLog changeLog =
(AbstractChangeLog)ChangeBugzillaHistories.getChangeLog("1", httpServletRequest);
if (changeLog == null) {
    changeLog = new EmptyChangeLog();
}
result.setChangeLog(changeLog);
return result;
```

The invocation of the static method builds the TRS Base resource and TRS Change Log at the moment. The built resources are kept in the **CreateBugzillaHistories** class. The built resources will be referred to later. See the **getBasePage()** method:

```
public Base getBasePage(@PathParam("page") String pagenum ) {  
    return ChangeBugzillaHistories.getBaseResource(pagenum, httpServletRequest);  
}
```

This method returns the TRS Base resource kept in the **CreateBugzillaHistories** class. Please see also **getChangeLogPage()** method:

```
public ChangeLog getChangeLogPage(@PathParam("page") String pagenum ) {  
    return ChangeBugzillaHistories.getChangeLog(pagenum, httpServletRequest);  
}
```

This method also returns the TRS ChangeLog resource kept in the **CreateBugzillaHistories** class.

In summary, the TRS resources are kept in the **CreateBugzillaHistories** class. They are built or refreshed by the **getTrackedResourceSet()** method, which is triggered by HTTP GET requests to **/trs**. Subsequent requests are delegated to the **CreateBugzillaHistories** class and the TRS resources kept in the class are returned.

- __2. Find the **getBase()** method. This method is mapped to **/trs/base** path due to the **@Path** annotation. Requests to this method are redirected to the URL that indicates the first page of the base resource. The base resource can be split into several pages for performance. The **getBasePage()** method is for the page and it returns the page from the **CreateBugzillaHistories** class. See TRS specification for the paging:

<http://open-services.net/wiki/core/TrackedResourceSet-2.0/#Base-Resources>

- __3. Find the **getChangeLog()** method. This method is mapped to **/trs/changeLog** path due to **@Path** annotation. Requests to this method are redirected to the URL that indicates the first page of the change log. The change log can also be split into several pages for performance. The **getChangeLogPage()** method is for the page and it returns the page from the **CreateBugzillaHistories** class.

- __4. Find the **buildBaseResourcesAnChangeLogsInternal()** method in the **CreateBugzillaHistories** class. It's the method for building the TRS resources.

__a. Around the line 325, you see **histories** array variable and in the several lines of the code, obtain Bug histories from Bugzilla and put them into the variable.

__b. Around the line 351, you see **changeLog** variable, it's the change log to be built.

__c. Around 359, there is a **for** loop for the **histories**. The TRS resources are built in the loop. You can find TRS change event creation

```
ChangeEvent ce = historyData.getType() == HistoryData.CREATED ?  
    new Creation(changedUri, uri, changeOrder) :  
    new Modification(changedUri, uri, changeOrder);
```

__d. You can also find the change log resource creation `changeLog = new ChangeLog();` The change log pages are stored into **changeLogs** static variable.

__e. You can also find the base resource creation base = **new** Base(); The base pages are stored into **baseResources** static variable.



The number of resources in each page is statically specified in the ChangeBugzillaHistories class.

BASE_PAGELIMIT (3 by default) is for the number of members in a Base page.

CHANGELOG_PAGELIMIT (3 by default) is for the number of resources in a ChangeLog page.



As for now, the number of Bugs TRS provider can extract is defined by the following properties in `org.eclipse.lyo.oslc4j.bugzilla.trsrc/main/resources/bugz.properties`.

max_number_of_products : The max number of Products TRS Provider will access.

max_number_of_bugs : The max number of Bugs in a Product TRS provider will access to generate their Change Logs.

start_date_year, start_date_month and start_date_day: . TRS Provider will access Bugs which were modified after this specified date. The time zone of this date is same as your PC setting.

__5. Time for the final tests of TRS

__a. Start OSLC4JBugzilla server if not already started. Run > Run Configuration > Maven Build > `org.eclipse.lyo.oslc4j.bugzilla.trsrc`

__b. In Firefox, go to Tools->REST Client

__c. Click Add Request Header and add a header with the name Accept and a value of either `application/rdf+xml`,

__d. Use the url <http://localhost:8085/OSLC4JBugzilla/services/trs/> (Note: "localhost" is the host name where Tracked Resource Set Provider runs. This host name is also specified in `org.eclipse.lyo.oslc4j.bugzilla.trsrc/main/resources/bugz.properties`.)

e. You will see Change Logs as well as the url to the base resources. For example,

[+] Response

Response Headers Response Body (Raw) Response Body (Highlight) Response Body (Preview)

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <rdf:RDF
3.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.   xmlns:oslc="http://open-services.net/ns/core#"
5.   xmlns:dcterms="http://purl.org/dc/terms/"
6.   xmlns:trs="http://jazz.net/ns/trs#"
7.   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
8.   <rdf:Description rdf:nodeID="A0">
9.     <rdf:rest rdf:nodeID="A1"/>
10.    <rdf:first rdf:resource="urn:urn-3:cm1.example.com:2012-09-26T01:08:57Z:47"/>
11.  </rdf:Description>
12.  <rdf:Description rdf:about="urn:urn-3:cm1.example.com:2012-09-26T01:08:57Z:47">
13.    <trs:order>47</trs:order>
14.    <trs:changed rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/2/changeRequests/5"/>
15.    <rdf:type rdf:resource="http://jazz.net/ns/trs#Modification"/>
16.  </rdf:Description>
17.  <rdf:Description rdf:about="http://localhost:8085/OSLC4JBugzilla/services/trs">
18.    <trs:base rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/trs/base"/>
19.    <trs:changeLog rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/trs/changeLog"/>
20.    <rdf:type rdf:resource="http://jazz.net/ns/trs#TrackedResourceSet"/>
21.  </rdf:Description>
22.  <rdf:Description rdf:about="urn:urn-3:cm1.example.com:2012-09-26T01:08:57Z:48">
23.    <trs:order>48</trs:order>
24.    <trs:changed rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/2/changeRequests/5"/>
25.    <rdf:type rdf:resource="http://jazz.net/ns/trs#Modification"/>
```

f. In the response, find "<trs:base>" tag

```
17. <rdf:Description rdf:about="http://localhost:8085/OSLC4JBugzilla/services/trs">
18.   <trs:base rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/trs/base"/>
19.   <trs:changeLog rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/trs/changeLog"/>
20.   <rdf:type rdf:resource="http://jazz.net/ns/trs#TrackedResourceSet"/>
21. </rdf:Description>
```

g. Use the that url (For example, <http://localhost:8085/OSLC4JBugzilla/services/trs/base>)

[+] Response

Response Headers Response Body (Raw) Response Body (Highlight) Response Body (Preview)

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <rdf:RDF
3.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.   xmlns:oslc="http://open-services.net/ns/core#"
5.   xmlns:dcterms="http://purl.org/dc/terms/"
6.   xmlns:trs="http://jazz.net/ns/trs#"
7.   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
8.   <rdf:Description rdf:about="http://localhost:8085/OSLC4JBugzilla/services/trs/base">
9.     <trs:nextPage rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/trs/base/2"/>
10.    <trs:cutoffEvent rdf:resource="urn:urn-3:cm1.example.com:2012-09-26T01:08:57Z:48"/>
11.    <trdfs:member rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/2/changeRequests/2"/>
12.    <trdfs:member rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/2/changeRequests/3"/>
13.    <trdfs:member rdf:resource="http://localhost:8085/OSLC4JBugzilla/services/2/changeRequests/5"/>
14.    <rdf:type rdf:resource="http://jazz.net/ns/trs#Base"/>
15.  </rdf:Description>
16. </rdf:RDF>
17.
```

1.4 Summary

This lab provided an introduction to defining OSLC resources and REST services using the Eclipse Lyo OSLC4J SDK (<http://eclipse.org/lyo>) for TRS. You now have the basics for exposing the change log of Bugzilla bugs as an OSLC Tracked Resource Set and you are able to retrieve them in a variety of representations which can be used by humans and other tools.

Lab 2 Connect to Jazz Team Server (optional)

Objectives

- Take the implemented adapter and use it for integration, especially with IBM Rational Jazz-based applications.
- This is an optional lab and can be skipped if you are constrained for time.

Description:

Now you have created a TRS provider, you'd like to connect to it with a TRS client. In the Lyo OSLC workshop, you have connected the Bugzilla adapter to Rational Team Concert as OSLC-based Jazz integration. You have also done some additional steps that are needed for the integration.

This lab will explore the steps necessary to achieve those integrations in a TRS provider. It will show some additional requirements that are needed such as the Jazz Root Services document and OAuth.

Steps:

- Understand additional steps needed for integration
 - OAuth
 - Update Jazz Root services

2.1 Enhance OAuth implementation for intra-server communication

You have implemented OAuth 1.0 consumer in “Lab 6. Connecting to Rational Team Concert” in the Lyo OSLC Workshop document. It allows RTC being an OAuth consumer and the user can see a login page to OSLC4JBugzilla in RTC. Here we enhance the OAuth implementation for typical usage of TRS provider and client.

In most cases, a TRS client runs in a back-end server and users might not be involved. For example, a TRS client that keeps the latest clone of a Tracked Resource Set is certainly implemented like a daemon and there is no chance to ask the user to submit credentials via login page. Consequently, we need to extend OAuth communication so that a Provider (implemented in Lab 1), and a Consumer (a TRS client), can communicate without user interaction (a.k.a. two-legged OAuth). The details about this kind of communication are in:

http://oauth.googlecode.com/svn/spec/ext/consumer_request/1.0/drafts/1/spec.html

We designed the two-legged authentication in OSLC4JBugzilla as follow:

- 1) The OAuth consumer used for two-legged authentication must be registered as “trusted” consumer
- 2) A request by the “trusted” consumer is processed using a pre-defined Bugzilla user account (a.k.a. functional user). In this workshop, the user is admin user and is specified in **bugz.properties** file.

2.1.1 Changed Resources

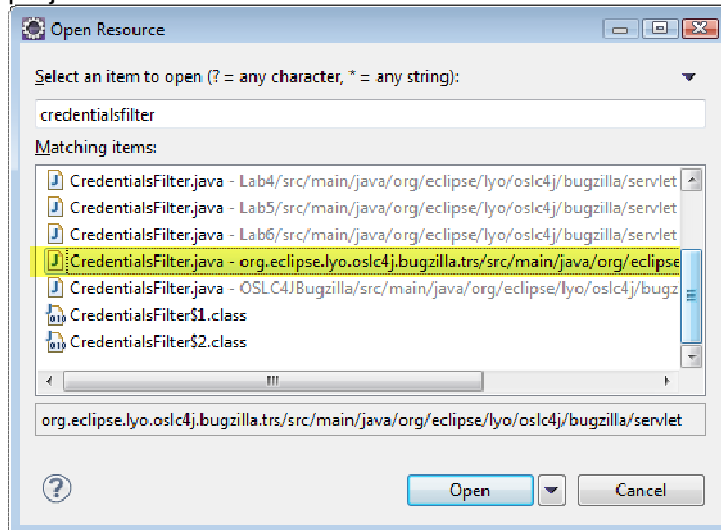
All the changed resources for this section are the following:

- ✓ **/resources/bugz.properties**: Modified. (Add properties)
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.servlet.CredentialsFilter.java**: Modified.
- ✓ **org.eclipse.lyo.oslc4j.bugzilla.BugzillaManager.java**: Modified.
 - Added `adapter_host` and `admin_password` fields.
 - Added statements starting with `admin_password` and `adapter_host` in the static initializer.
 - Added `getAdminCreadentials()` and `getHost()` methods.

2.1.2 Steps

- ___1. You will implement two-legged OAuth beside your existing OAuth implementation. The child sections show how it is implemented in `org.eclipse.lyo.oslc4j.bugzilla.trs` example.

__a. Open **CredentialsFilter.java** in **org.eclipse.lyo.oslc4j.bugzilla.servlet** and then locate the **doFilter()** method. The class is the existing OAuth implementation in **org.eclipse.lyo.oslc4j.bugzilla.trs**. Type **Ctrl+Shift+R** to open “Open Resource” dialog, and in the dialog, type some characters of the file name. Then, select one in **org.eclipse.lyo.oslc4j.bugzilla.trs** project.



__b. The first step for handling two-legged OAuth is to validate the request. The following fragment takes an HTTP request and raises the **isTwoLeggedOAuthRequest** flag when the request is a valid two-legged OAuth request. The flag is considered in the next step to fill the functional user credentials.

```
OAuthMessage message = OAuthServlet.getMessage(request, null);
// test if this is a valid two-legged oauth request
if ("".equals(message.getToken())) {
    validateTwoLeggedOAuthMessage(message);
    isTwoLeggedOAuthRequest = true;
}
```

__c. The **validateTwoLeggedOAuthMessage()** method is to confirm the message being signed using valid consumer key and consumer secret. The fragments above and below are reusable in your component to validate the request.

```
private static void validateTwoLeggedOAuthMessage(OAuthMessage message)
    throws IOException, OAuthException, URISyntaxException {
    OAuthConfiguration config = OAuthConfiguration.getInstance();
    LyoOAuthConsumer consumer = config.getConsumerStore()
        .getConsumer(message.getConsumerKey());
    if (consumer != null && consumer.isTrusted()) {
        // The request can be a two-legged request because it's a trusted consumer
        // Validate the message with an empty token and an empty secret
        OAuthAccessor accessor = new OAuthAccessor(consumer);
        accessor.requestToken = "";
        accessor.tokenSecret = "";
        config.getValidator().validateMessage(message, accessor);
    } else {
        throw new OAuthProblemException(
            OAuth.Problems.TOKEN_REJECTED);
    }
}
```

```

    }
}

```

__d. Once the request is validated, the next step is to grant the request as a (pre-authorized) functional user. In TRS4Bugzilla example, it's done by associating the request with an authorized BugzillaConnector instance.

__i. Open **bugz.properties** in **src/main/resources**. The property named **admin** is for specifying Bugzilla admin user name and **admin_password** property is for the password. The values are used to create a credential for logging in to Bugzilla for two-legged OAuth when the **isTwoLeggedOAuthRequest** flag is set. Go back to the **doFilter()** method and scroll down to the following fragment. In the fragment, ensure that a BugzillaConnector is associated to a token ("") used for two-legged authentication when it's not associated when the flag is **true**, :

```

Credentials credentials;
if (isTwoLeggedOAuthRequest) {
    connector = keyToConnectorCache.get("");
    if (connector == null) {
        credentials = BugzillaManager.getAdminCredentials();
        connector = getBugzillaConnector(credentials);
        keyToConnectorCache.put("", connector);
    }
} else {
    ...
}

```

__2. You have defined two-legged OAuth. The changes are tested in the later section.

2.1.3 Implementing a Credentials Filter

You have implemented the **CredentialsFilter** class for this workshop. The class takes credentials from requests to the adapter, and then passes it to the Bugzilla server. When you create your adapter, your "CredentialsFilter" might be similar. So, there is the base class, **AbstractAdapterCredentialsFilter**, which provides the common part. Please find the javadoc of the class.

In the workshop code, there is also the **BugzillaAdapterCredentialsFilter** class, an example implementation of "CredentialsFilter" based on the base class. You can try the **BugzillaAdapterCredentialsFilter** following the instruction in the javadoc.

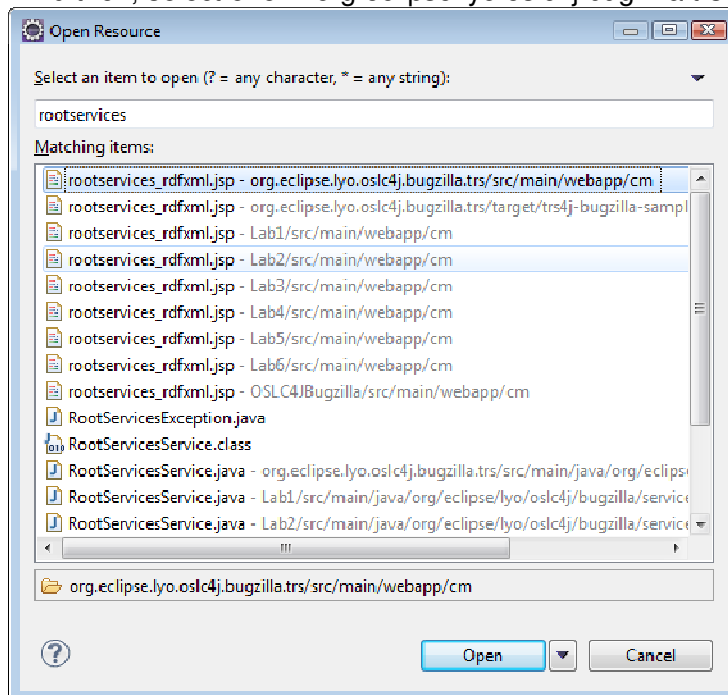
2.2 Enhance Jazz Root Services support

You have added Jazz Rootservices document in "Lab 6. Connecting to Rational Team Concert in Lyo OSLC Workshop document" in the Lyo OSLC Workshop so that the Jazz Team Server and other Jazz-integrated applications query the TRS services your adapter provides. In addition to Lab 6 in the Lyo OSLC Workshop, the following resources provide details:

<https://jazz.net/wiki/bin/view/Main/RootServicesSpec>

<https://jazz.net/wiki/bin/view/Main/RootServicesSpecAddendum2>

- ___1. Open **rootservices_rdfxml.jsp** in org.eclipse.lyo.oslc4j.bugzilla.trs project. The file is a template of the rootservices document in org.eclipse.lyo.oslc4j.bugzilla.trs.
 - ___a. Type Ctrl+Shift+R to open “Open Resource” dialog, and type some characters of the file name in the edit.
 - ___b. And then, select one in org.eclipse.lyo.oslc4j.bugzilla.trs project.



- ___2. Locate an element **bugz:trackedResourceSetProvider**. The element is for a definition of the TRS in rootservices document and points the TRS URI. In org.eclipse.lyo.oslc4j.bugzilla.trs, the URI is “baseUri + “/trs””. In RDF point of view, the rootservices RDF resource has a new predicate, **bugz:trackedResourceSetProvider**, and the object is an anonymous resource whose type is **trs:TrackedResourceSetProvider**. The anonymous resource further has a predicate **trs:trackedResourceSet** and the object is the TRS URI.

```
<!-- Bugzilla Tracked Resource Set Provider -->
<bugz:trackedResourceSetProvider>
  <trs:TrackedResourceSetProvider>
    <trs:trackedResourceSet rdf:resource="<%= baseUri + "/trs" %>" />
  </trs:TrackedResourceSetProvider>
</bugz:trackedResourceSetProvider>
```

- ___a. For the **bugz** and **trs** prefixes in the XML, you'll have added **xmlns:bugz** and **xmlns:trs** attributes to the root **rdf:Description** element.

```
<rdf:Description rdf:about="<%= baseUri + "/rootservices" %>"
  xmlns:bugz="http://www.bugzilla.org/rdf#"
  xmlns:trs="http://jazz.net/ns/trs#"
```

```
xmlns:oslc_cm="http://open-services.net/xmlns/cm/1.0/"  
xmlns:dcterms="http://purl.org/dc/terms/"  
xmlns:jfs="http://jazz.net/xmlns/prod/jazz/jfs/1.0/"  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```



The namespace URI for the TRS may change during the specification finalization process.

Please find the latest version:

<http://open-services.net/wiki/core/TrackedResourceSet-2.0/>

2.3 Summary

This lab provided requires steps to integrate your TRS provider with Jazz Team Server. You now have basic understandings of the steps and how to achieve them.

Appendix A. Troubleshooting

There are always a number of expected things that you might encounter. If there is anything that you may not be able to overcome, please see a proctor or the workshop leader.

In Eclipse, check for compilation errors in the Eclipse project

In Eclipse, Make sure multiple servers aren't running (or right servers are running)

In Eclipse, check Console for Server errors.

Firefox browser has installed an add-in called Firebug that can be used to inspect the web page's DOM and JavaScript.

Ask a proctor for help.

Appendix B. Sources

The sources for this workshop are based on materials that are available in an open source project around OSLC enablement material. Also there are additional materials available to help with consuming OSLC services and more advanced topics are available around authentication and query. See references below for sources:

Eclipse Lyo project <http://eclipse.org/Lyo> and <http://wiki.eclipse.org/Lyo>

Everyone is encouraged to join and contribute to these efforts that can greatly help build the ecosystem of OSLC-based tool integration

OSLC Tutorial

Check <http://open-services.net> for the development of a 2 part tutorial where you can establish your own development environment and follow along with the examples.

Part 1 Consuming OSLC Services

Part 2 Providing OSLC Services

General Resources

As new resources and enablement material are being made available, be sure to check out the OSLC website for these developments at <http://open-services.net>

Appendix C. Legal

1. Notices

This material in this guide is Copyright © IBM Corporation 2011, 2013.

2. About this Content

April 10, 2013

License

The Eclipse Foundation makes available all content in this plug-in (“Content”). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of the Eclipse Public License Version 1.0 (“EPL”) and Eclipse Distribution License Version 1.0 (“EDL”). A copy of the EPL is available at <http://www.eclipse.org/legal/epl-v10.html> and a copy of the EDL is available at <http://www.eclipse.org/org/documents/edl-v10.php>.

For purposes of the EPL, “Program” will mean the Content.

If you did not receive this Content directly from the Eclipse Foundation, the Content is being redistributed by another party (“Redistributor”) and different terms and conditions may apply to your use of any object code in the Content. Check the Redistributor's license that was provided with the Content. If no such license exists, contact the Redistributor. Unless otherwise indicated below, the terms and conditions of the EPL and EDL still apply to any source code in the Content and such source code may be obtained at <http://www.eclipse.org>

NOTES

[illegible]

NOTES

[illegible]

NOTES
